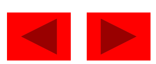
Chapter 1 – Introduction to Computers and C++ Programming

Outline

1.1 Introduction
1.2 What Is a Computer?
1.3 Computer Organization
1.4 Evolution of Operating Systems
1.5 Personal Computing, Distributed Computing and Client/Server Computing
1.6 Machine Languages, Assembly Languages and High-level Languages
1.7 The History of C
1.8 The C Standard Library
1.9 The Key Software Trend: Object Technology
1.10 C++ and C++ How to Program
1.11 Java and Java How to Program
1.12 Other High-level Languages
1.13 Structured Programming
1.14 The Basics of a typical C Program Development Environment

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

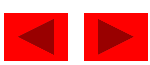


Chapter 1 – Introduction to Computers and C++ Programming

Outline

1.15 Hardware Trends
1.16 History of the Internet
1.17 History of the World Wide Web
1.18 General Notes About C and this Book

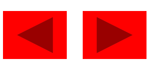
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Objectives

- In this chapter, you will learn:
  - To understand basic computer concepts.
  - To become familiar with different types of programming languages.
  - To become familiar with the history of the C programming language.
  - To become aware of the C standard library.
  - To understand the elements of a typical C program development environment.
  - To appreciate why it is important to learn C in a first programming course.
  - To appreciate why C provides a foundation for further study of programming languages in general and of C++ and Java in particular.

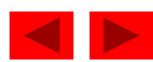
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



1.1 Introduction

- We will learn
  - The C programming language
  - Structured programming and proper programming techniques
- This book also covers
  - C++
    - Chapter 15 – 23 introduce the C++ programming language
  - Java
    - Chapters 24 – 30 introduce the Java programming language
- This course is appropriate for
  - Technically oriented people with little or no programming experience
  - Experienced programmers who want a deep and rigorous treatment of the language

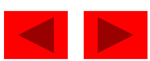
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



1.2 What is a Computer?

- Computer
  - Device capable of performing computations and making logical decisions
  - Computers process data under the control of sets of instructions called computer programs
- Hardware
  - Various devices comprising a computer
  - Keyboard, screen, mouse, disks, memory, CD-ROM, and processing units
- Software
  - Programs that run on a computer

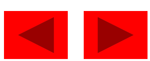
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



1.3 Computer Organization

- Six logical units in every computer:
  1. Input unit
     - Obtains information from input devices (keyboard, mouse)
  2. Output unit
     - Outputs information (to screen, to printer, to control other devices)
  3. Memory unit
     - Rapid access, low capacity, stores input information
  4. Arithmetic and logic unit (ALU)
     - Performs arithmetic calculations and logic decisions
  5. Central processing unit (CPU)
     - Supervises and coordinates the other sections of the computer
  6. Secondary storage unit
     - Cheap, long-term, high-capacity storage
     - Stores inactive programs

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



1.4 Evolution of Operating Systems

- Batch processing
  - Do only one job or task at a time
- Operating systems
  - Manage transitions between jobs
  - Increased throughput
    - Amount of work computers process
- Multiprogramming
  - Computer resources are shared by many jobs or tasks
- Timesharing
  - Computer runs a small portion of one user's job then moves on to service the next user

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

1.5 Personal Computing, Distributed Computing, and Client/Server Computing

- Personal computers
  - Economical enough for individual
- Distributed computing
  - Computing distributed over networks
- Client/server computing
  - Sharing of information across computer networks between file servers and clients (personal computers)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

1.6 Machine Languages, Assembly Languages, and High-level Languages

Three types of programming languages

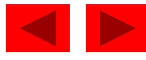
1. Machine languages
   - Strings of numbers giving machine specific instructions
   - Example:

```
+1300042774
+1400593419
+1200274027
```

2. Assembly languages
   - English-like abbreviations representing elementary computer operations (translated via assemblers)
   - Example:

```
LOAD    BASEPAY
ADD     OVERPAY
STORE   GROSSPAY
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



1.6 Machine Languages, Assembly Languages, and High-level Languages

Three types of programming languages (continued)

3. High-level languages
   - Codes similar to everyday English
   - Use mathematical notations (translated via compilers)
   - Example:

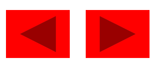
```
grossPay = basePay + overTimePay
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

1.7 History of C

- C
  - Evolved by Ritchie from two previous programming languages, BCPL and B
  - Used to develop UNIX
  - Used to write modern operating systems
  - Hardware independent (portable)
  - By late 1970's C had evolved to "Traditional C"
- Standardization
  - Many slight variations of C existed, and were incompatible
  - Committee formed to create a "unambiguous, machine-independent" definition
  - Standard created in 1989, updated in 1999

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



1.8 The C Standard Library

- C programs consist of pieces/modules called functions
  - A programmer can create his own functions
    - Advantage: the programmer knows exactly how it works
    - Disadvantage: time consuming
  - Programmers will often use the C library functions
    - Use these as building blocks
  - Avoid re-inventing the wheel
    - If a premade function exists, generally best to use it rather than write your own
    - Library functions carefully written, efficient, and portable

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

1.9 The Key Software Trend: Object Technology

- Objects
  - Reusable software components that model items in the real world
  - Meaningful software units
    - Date objects, time objects, paycheck objects, invoice objects, audio objects, video objects, file objects, record objects, etc.
    - Any noun can be represented as an object
  - Very reusable
  - More understandable, better organized, and easier to maintain than procedural programming
  - Favor modularity

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

1.10 C++ and C++ How to Program

- C++
  - Superset of C developed by Bjarne Stroustrup at Bell Labs
  - "Spruces up" C, and provides object-oriented capabilities
  - Object-oriented design very powerful
    - 10 to 100 fold increase in productivity
  - Dominant language in industry and academia
- Learning C++
  - Because C++ includes C, some feel it is best to master C, then learn C++
  - Starting in Chapter 15, we begin our introduction to C++

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

1.11 Java and Java How to Program

- Java is used to
  - Create Web pages with dynamic and interactive content
  - Develop large-scale enterprise applications
  - Enhance the functionality of Web servers
  - Provide applications for consumer devices (such as cell phones, pagers and personal digital assistants)
- Java How to Program
  - Closely followed the development of Java by Sun
  - Teaches first-year programming students the essentials of graphics, images, animation, audio, video, database, networking, multithreading and collaborative computing

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

1.12 Other High-level Languages

- Other high-level languages
  - FORTRAN
    - Used for scientific and engineering applications
  - COBOL
    - Used to manipulate large amounts of data
  - Pascal
    - Intended for academic use

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

1.13 Structured Programming

- Structured programming
  - Disciplined approach to writing programs
  - Clear, easy to test and debug and easy to modify
- Multitasking
  - Specifying that many activities run in parallel

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

1.14 Basics of a Typical C Program Development Environment

- Phases of C++ Programs:

1. Edit

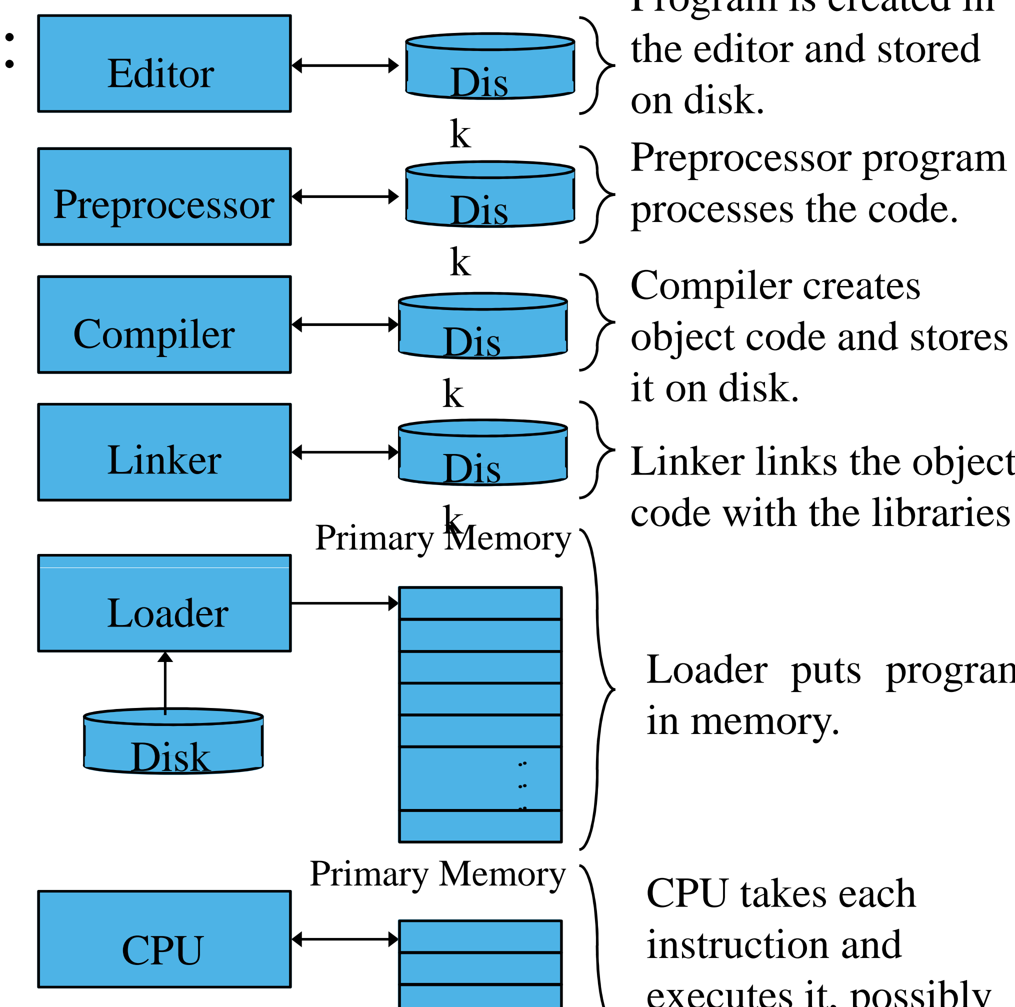
2. Preprocess

3. Compile

4. Link

5. Load

6. Execute



© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Righ

1.15 Hardware Trends

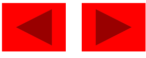
- Every year or two the following approximately double:
  - Amount of memory in which to execute programs
  - Amount of secondary storage (such as disk storage)
    - Used to hold programs and data over the longer term
  - Processor speeds
    - The speeds at which computers execute their programs

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

1.16 History of the Internet

- The Internet enables
  - Quick and easy communication via e-mail
  - International networking of computers
- Packet switching
  - The transfer of digital data via small packets
  - Allows multiple users to send and receive data simultaneously
- No centralized control
  - If one part of the Internet fails, other parts can still operate
- TCP/IP
- Bandwidth
  - Information carrying capacity of communications lines

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



1.17 History of the World Wide Web

- World Wide Web
  - Locate and view multimedia-based documents on almost any subject
  - Makes information instantly and conveniently accessible worldwide
  - Possible for individuals and small businesses to get worldwide exposure
  - Changing the way business is done

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

1.18 General Notes About C and This Book

- Program clarity
  - Programs that are convoluted are difficult to read, understand, and modify
- C is a portable language
  - Programs can run on many different computers
  - However, portability is an elusive goal
- We will do a careful walkthrough of C
  - Some details and subtleties are not covered
  - If you need additional technical details
    - Read the C standard document
    - Read the book by Kernigan and Ritchie

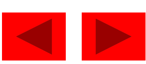
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Chapter 2 - Introduction to C Programming

Outline

2.1 Introduction
2.2 A Simple C Program: Printing a Line of Text
2.3 Another Simple C Program: Adding Two Integers
2.4 Memory Concepts
2.5 Arithmetic in C
2.6 Decision Making: Equality and Relational Operators

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Objectives

- In this chapter, you will learn:
  - To be able to write simple computer programs in C.
  - To be able to use simple input and output statements.
  - To become familiar with fundamental data types.
  - To understand computer memory concepts.
  - To be able to use arithmetic operators.
  - To understand the precedence of arithmetic operators.
  - To be able to write simple decision making statements.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

2.1 Introduction

- C programming language
  - Structured and disciplined approach to program design
- Structured programming
  - Introduced in chapters 3 and 4
  - Used throughout the remainder of the book

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

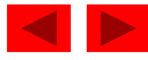
2.2 A Simple C Program: Printing a Line of Text

```
1    /* Fig. 2.1: fig02_01.c
2       A first program in C */
3    #include <stdio.h>
4
5    /* function main begins program execution */
6    int main()
7    {
8       printf( "Welcome to C!\n" );
9
10      return 0; /* indicate that program ended successfully */
11
12   } /* end function main */
```

```
Welcome to C!
```

- Comments
  - Text surrounded by `/*` and `*/` is ignored by computer
  - Used to describe program
- `#include <stdio.h>`
  - Preprocessor directive
    - Tells computer to load contents of a certain file
  - `<stdio.h>` allows standard input/output operations

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



2.2 A Simple C Program: Printing a Line of Text

- `int main()`
  - C++ programs contain one or more functions, exactly one of which must be `main`
  - Parenthesis used to indicate a function
  - `int` means that `main` "returns" an integer value
  - Braces (`{` and `}`) indicate a block
    - The bodies of all functions must be contained in braces

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

2.2 A Simple C Program: Printing a Line of Text

- `printf( "Welcome to C!\n" );`
  - Instructs computer to perform an action
    - Specifically, prints the string of characters within quotes (" ")
  - Entire line called a statement
    - All statements must end with a semicolon (;)
  - Escape character (\)
    - Indicates that printf should do something out of the ordinary
    - `\n` is the newline character

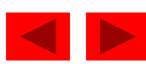
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

2.2 A Simple C Program: Printing a Line of Text

Escape Sequence	Description
`\n`	Newline. Position the cursor at the beginning of the next line.
`\t`	Horizontal tab. Move the cursor to the next tab stop.
`\a`	Alert. Sound the system bell.
`\\`	Backslash. Insert a backslash character in a string.
`\"`	Double quote. Insert a double quote character in a string.

Fig. 2.2 Some common escape sequences.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



2.2 A Simple C Program: Printing a Line of Text

- `return 0;`
  - A way to exit a function
  - `return 0`, in this case, means that the program terminated normally
- Right brace }
  - Indicates end of `main` has been reached
- Linker
  - When a function is called, linker locates it in the library
  - Inserts it into object program
  - If function name is misspelled, the linker will produce an error because it will not be able to find function in the library

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

fig02_03.c

```
1  /* Fig. 2.3: fig02_03.c
2     Printing on one line with two printf statements */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main()
7  {
8     printf( "Welcome " );
9     printf( "to C!\n" );
10
11     return 0; /* indicate that program ended successfully */
12
13 } /* end function main */
```

Program Output

```
Welcome to C!
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

fig02_04.c

```
1  /* Fig. 2.4: fig02_04.c
2     Printing multiple lines with a single printf */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main()
7  {
8     printf( "Welcome\nto\nC!\n" );
9
10    return 0; /* indicate that program ended successfully */
11
12 } /* end function main */
```

```
Welcome
to
C!
```

Program Output

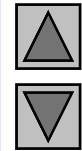
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

fig02_05.c

```
1  /* Fig. 2.5: fig02_05.c
2     Addition program */
3  #include <stdio.h>
4  
5  /* function main begins program execution */
6  int main()
7  {
8     int integer1; /* first number to be input by user  */
9     int integer2; /* second number to be input by user */
10    int sum;      /* variable in which sum will be stored */
11 
12    printf( "Enter first integer\n" );  /* prompt */
13    scanf( "%d", &integer1 );           /* read an integer */
14 
15    printf( "Enter second integer\n" ); /* prompt */
16    scanf( "%d", &integer2 );           /* read an integer */
17 
18    sum = integer1 + integer2;          /* assign total to sum */
19 
20    printf( "Sum is %d\n", sum );       /* print sum */
21 
22    return 0;  /* indicate that program ended successfully */
23 
24 } /* end function main */
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
Enter first integer
45
Enter second integer
72
Sum is 117
```



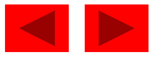
Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

2.3 Another Simple C Program: Adding Two Integers

- As before
  - Comments, `#include <stdio.h>` and `main`
- `int integer1, integer2, sum;`
  - Definition of variables
    - Variables: locations in memory where a value can be stored
  - `int` means the variables can hold integers (`-1, 3, 0, 47`)
  - Variable names (identifiers)
    - `integer1`, `integer2`, `sum`
    - Identifiers: consist of letters, digits (cannot begin with a digit) and underscores(_)
      - Case sensitive
  - Definitions appear before executable statements
    - If an executable statement references and undeclared variable it will produce a syntax (compiler) error

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



2.3 Another Simple C Program: Adding Two Integers

- `scanf( "%d", &integer1 );`
  - Obtains a value from the user
    - `scanf` uses standard input (usually keyboard)
  - This `scanf` statement has two arguments
    - `%d` - indicates data should be a decimal integer
    - `&integer1` - location in memory to store variable
    - `&` is confusing in beginning – for now, just remember to include it with the variable name in `scanf` statements
  - When executing the program the user responds to the `scanf` statement by typing in a number, then pressing the *enter* (return) key

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

2.3 Another Simple C Program: Adding Two Integers

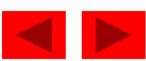
- = (assignment operator)
  - Assigns a value to a variable
  - Is a binary operator (has two operands)

```
sum = variable1 + variable2;
sum gets variable1 + variable2;
```

  - Variable receiving value on left
- `printf( "Sum is %d\n", sum );`
  - Similar to `scanf`
    - `%d` means decimal integer will be printed
    - `sum` specifies what integer will be printed
  - Calculations can be performed inside `printf` statements

```
printf( "Sum is %d\n", integer1 + integer2 );
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

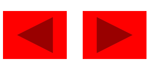


2.4 Memory Concepts

- Variables
  - Variable names correspond to locations in the computer's memory
  - Every variable has a name, a type, a size and a value
  - Whenever a new value is placed into a variable (through `scanf`, for example), it replaces (and destroys) the previous value
  - Reading variables from memory does not change them
- A visual representation

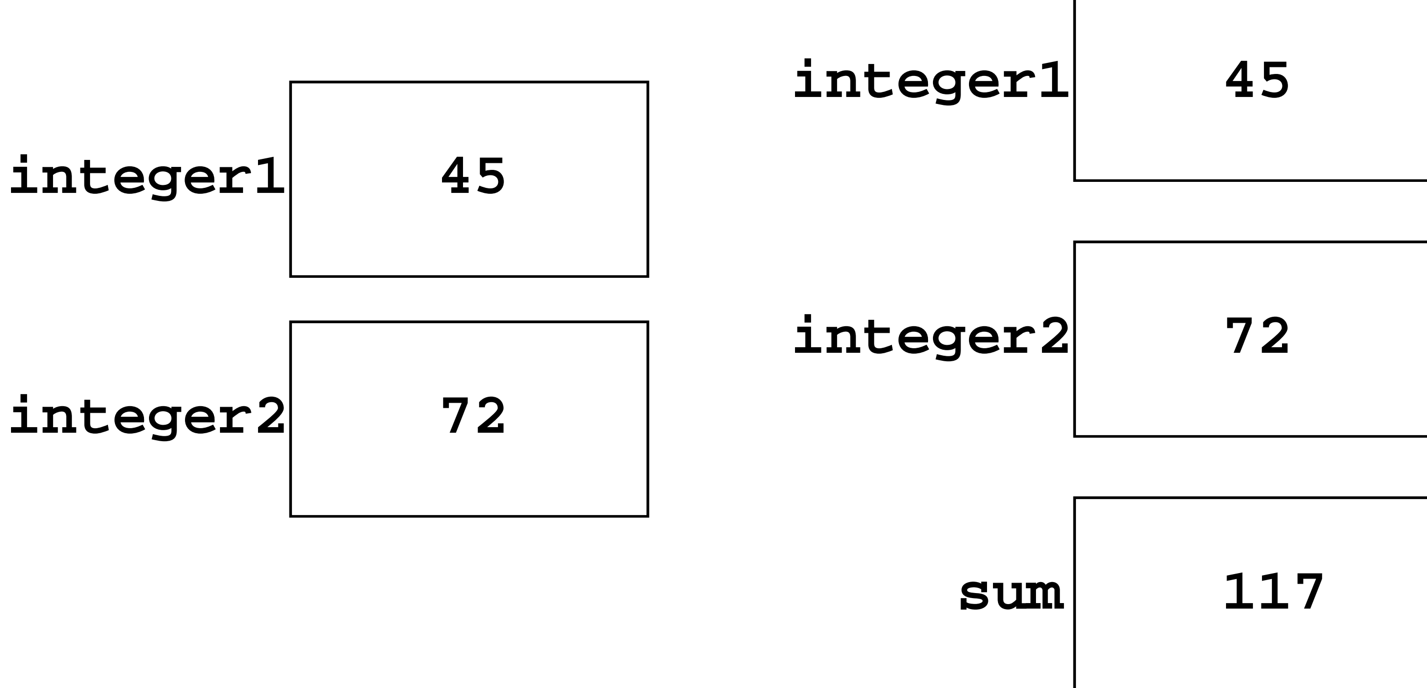
integer1	**45**

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

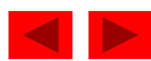


2.4 Memory Concepts

- A visual representation (continued)



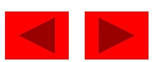
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



2.5 Arithmetic

- Arithmetic calculations
  - Use `*` for multiplication and `/` for division
  - Integer division truncates remainder
    - `7 / 5` evaluates to `1`
  - Modulus operator(%) returns the remainder
    - `7 % 5` evaluates to `2`
- Operator precedence
  - Some arithmetic operators act before others (i.e., multiplication before addition)
    - Use parenthesis when needed
  - Example: Find the average of three variables `a`, `b` and `c`
    - Do not use: `a + b + c / 3`
    - Use: `(a + b + c ) / 3`

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



2.5 Arithmetic

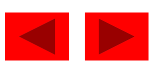
- Arithmetic operators:

C operation	Arithmetic operator	Algebraic expression	C expression
Addition	`+`	$f + 7$	`f + 7`
Subtraction	`-`	$p - c$	`p - c`
Multiplication	`*`	bm	`b * m`
Division	`/`	x / y	`x / y`
Modulus	`%`	$r \bmod s$	`r % s`

- Rules of operator precedence:

Operator(s)	Operation(s)	Order of evaluation (precedence)
`()`	Parentheses	Evaluated first. If the parentheses are nested, the expression in the innermost pair is evaluated first. If there are several pairs of parentheses "on the same level" (i.e., not nested), they are evaluated left to right.
`*`, `/`, or `%`	Multiplication,Division, Modulus	Evaluated second. If there are several, they are evaluated left to right.
`+` or `-`	Addition Subtraction	Evaluated last. If there are several, they are evaluated left to right.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



2.6 Decision Making: Equality and Relational Operators

Step 1. `y = 2 * 5 * 5 + 3 * 5 + 7;` *(Leftmost multiplication)*

`2 * 5 is 10`

Step 2. `y = 10 * 5 + 3 * 5 + 7;` *(Leftmost multiplication)*

`10 * 5 is 50`

Step 3. `y = 50 + 3 * 5 + 7;` *(Multiplication before addition)*

`3 * 5 is 15`

Step 4. `y = 50 + 15 + 7;` *(Leftmost addition)*

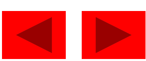
`50 + 15 is 65`

Step 5. `y = 65 + 7;` *(Last addition)*

`65 + 7 is 72`

Step 6. `y = 72;` *(Last operation—place 72 in y)*

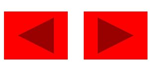
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



2.6 Decision Making: Equality and Relational Operators

- Executable statements
  - Perform actions (calculations, input/output of data)
  - Perform decisions
    - May want to print `"pass"` or `"fail"` given the value of a test grade
- `if` control statement
  - Simple version in this section, more detail later
  - If a condition is `true`, then the body of the `if` statement executed
    - `0` is `false`, non-zero is `true`
  - Control always resumes after the `if` structure
- Keywords
  - Special words reserved for C
  - Cannot be used as identifiers or variable names

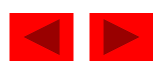
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

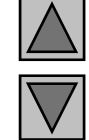


2.6 Decision Making: Equality and Relational Operators

Standard algebraic equality operator or relational operator	C equality or relational operator	Example of C condition	Meaning of C condition
Equality Operators			
=	==	x == y	x is equal to y
≠	!=	x != y	x is not equal to y
Relational Operators			
>	>	x > y	x is greater than y
<	<	x < y	x is less than y
>=	>=	x >= y	x is greater than or equal to y
<=	<=	x <= y	x is less than or equal to y

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.





Outline

fig02_13.c (Part 1 of 2)

```
1  /* Fig. 2.13: fig02_13.c
2     Using if statements, relational
3     operators, and equality operators */
4  #include <stdio.h>
5
6  /* function main begins program execution */
7  int main()
8  {
9     int num1, /* first number to be read from user  */
10     int num2; /* second number to be read from user */
11
12     printf( "Enter two integers, and I will tell you\n" );
13     printf( "the relationships they satisfy: " );
14
15     scanf( "%d%d", &num1, &num2 ); /* read two integers */
16
17     if ( num1 == num2 ) {
18        printf( "%d is equal to %d\n", num1, num2 );
19     } /* end if */
20
21     if ( num1 != num2 ) {
22        printf( "%d is not equal to %d\n", num1, num2 );
23     } /* end if */
24
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Outline

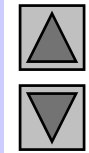
fig02_13.c (Part 2 of 2)

```
25    if ( num1 < num2 ) {
26       printf( "%d is less than %d\n", num1, num2 );
27    } /* end if */
28
29    if ( num1 > num2 ) {
30       printf( "%d is greater than %d\n", num1, num2 );
31    } /* end if */
32
33    if ( num1 <= num2 ) {
34       printf( "%d is less than or equal to %d\n", num1, num2 );
35    } /* end if */
36
37    if ( num1 >= num2 ) {
38       printf( "%d is greater than or equal to %d\n", num1, num2 );
39    } /* end if */
40
41    return 0;   /* indicate that program ended successfully */
42
43 } /* end function main */
```

Program Output

```
Enter two integers, and I will tell you
the relationships they satisfy: 3 7
3 is not equal to 7
3 is less than 7
3 is less than or equal to 7
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Outline

Program Output (continued)

```
Enter two integers, and I will tell you
the relationships they satisfy: 22 12
22 is not equal to 12
22 is greater than 12
22 is greater than or equal to 12
```

```
Enter two integers, and I will tell you
the relationships they satisfy: 7 7
7 is equal to 7
7 is less than or equal to 7
7 is greater than or equal to 7
```

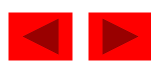
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

2.6 Decision Making: Equality and Relational Operators

Operators				Associativity
*	/	%		left to right
+	-			left to right
<	<=	>	>=	left to right
==	!=			left to right
=				right to left

Fig. 2.14 Precedence and associativity of the operators discussed so far.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

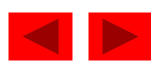


2.6 Decision Making: Equality and Relational Operators

Keywords			
auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Fig. 2.15 C's reserved keywords.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

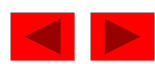


Chapter 3 - Structured Program Development

Outline

3.1 Introduction
3.2 Algorithms
3.3 Pseudocode
3.4 Control Structures
3.5 The If Selection Statement
3.6 The If…Else Selection Statement
3.7 The While Repetition Statement
3.8 Formulating Algorithms: Case Study 1 (Counter-Controlled Repetition)
3.9 Formulating Algorithms with Top-down, Stepwise Refinement: Case Study 2 (Sentinel-Controlled Repetition)
3.10 Formulating Algorithms with Top-down, Stepwise Refinement: Case Study 3 (Nested Control Structures)
3.11 Assignment Operators
3.12 Increment and Decrement Operators

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Objectives

- In this chapter, you will learn:
  - To understand basic problem solving techniques.
  - To be able to develop algorithms through the process of top-down, stepwise refinement.
  - To be able to use the `if` selection statement and `if...else` selection statement to select actions.
  - To be able to use the `while` repetition statement to execute statements in a program repeatedly.
  - To understand counter-controlled repetition and sentinel-controlled repetition.
  - To understand structured programming.
  - To be able to use the increment, decrement and assignment operators.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

3.1 Introduction

- Before writing a program:
  - Have a thorough understanding of the problem
  - Carefully plan an approach for solving it
- While writing a program:
  - Know what “building blocks” are available
  - Use good programming principles

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

3.2 Algorithms

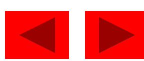
- Computing problems
  - All can be solved by executing a series of actions in a specific order
- Algorithm: procedure in terms of
  - Actions to be executed
  - The order in which these actions are to be executed
- Program control
  - Specify order in which statements are to be executed

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

3.3 Pseudocode

- Pseudocode
  - Artificial, informal language that helps us develop algorithms
  - Similar to everyday English
  - Not actually executed on computers
  - Helps us “think out” a program before writing it
    - Easy to convert into a corresponding C++ program
    - Consists only of executable statements

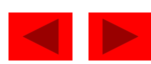
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



3.4 Control Structures

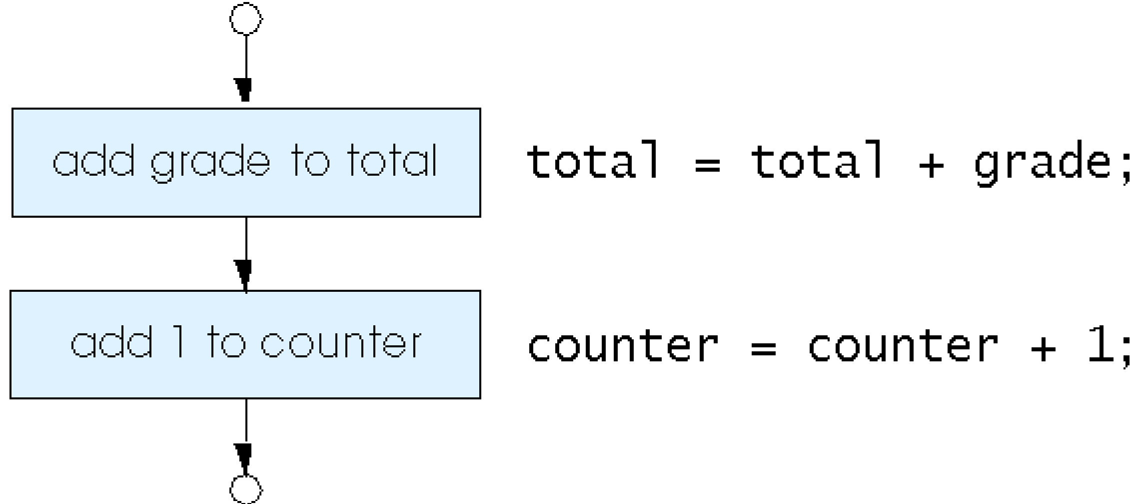
- Sequential execution
  - Statements executed one after the other in the order written
- Transfer of control
  - When the next statement executed is not the next one in sequence
  - Overuse of `goto` statements led to many problems
- Bohm and Jacopini
  - All programs written in terms of 3 control structures
    - Sequence structures: Built into C. Programs executed sequentially by default
    - Selection structures: C has three types: `if`, `if`…`else`, and `switch`
    - Repetition structures: C has three types: `while`, `do`…`while` and `for`

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

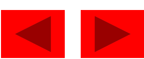


3.4 Control Structures

Figure 3.1 Flowcharting C's sequence structure.



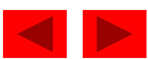
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



3.4 Control Structures

- Flowchart
  - Graphical representation of an algorithm
  - Drawn using certain special-purpose symbols connected by arrows called flowlines
  - Rectangle symbol (action symbol):
    - Indicates any type of action
  - Oval symbol:
    - Indicates the beginning or end of a program or a section of code
- Single-entry/single-exit control structures
  - Connect exit point of one control structure to entry point of the next (control-structure stacking)
  - Makes programs easy to build

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



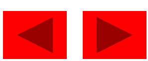
3.5 The `if` Selection Statement

- Selection structure:
  - Used to choose among alternative courses of action
  - Pseudocode:

    If student's grade is greater than or equal to 60
    Print "Passed"

- If condition `true`
  - Print statement executed and program goes on to next statement
  - If `false`, print statement is ignored and the program goes onto the next statement
  - Indenting makes programs easier to read
    - C ignores whitespace characters

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



3.5 The if Selection Statement

- Pseudocode statement in C:

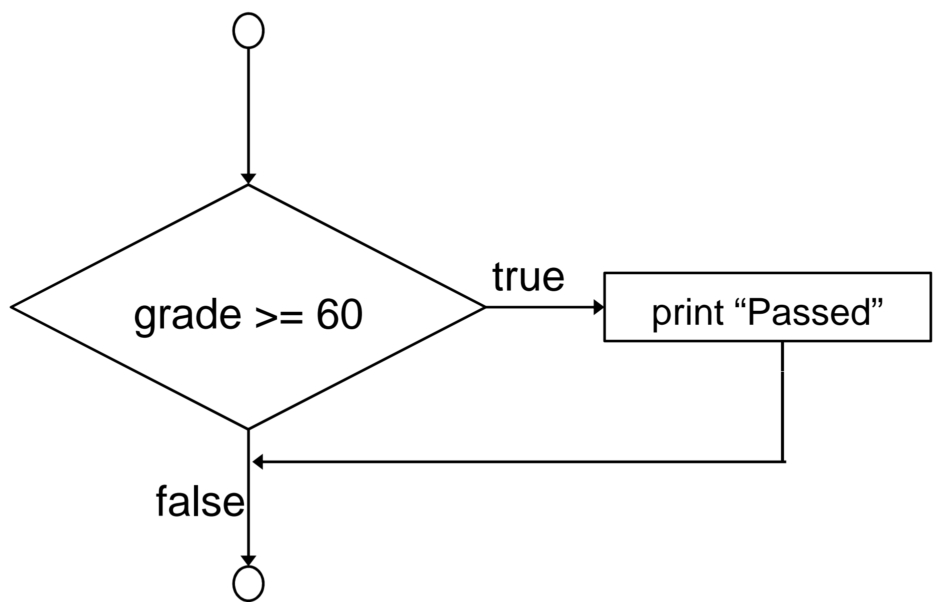
```
if ( grade >= 60 )
   printf( "Passed\n" );
```

  - C code corresponds closely to the pseudocode
- Diamond symbol (decision symbol)
  - Indicates decision is to be made
  - Contains an expression that can be `true` or `false`
  - Test the condition, follow appropriate path

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

3.5 The if Selection Statement

- if statement is a single-entry/single-exit structure



A decision can be made on any expression.

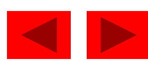
zero - false

nonzero - true

Example:

3 - 4 is true

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



3.6 The `if...else` Selection Statement

- `if`
  - Only performs an action if the condition is `true`
- `if...else`
  - Specifies an action to be performed both when the condition is `true` and when it is `false`
- Psuedocode:

  If student's grade is greater than or equal to 60
  Print "Passed"
  else
  Print "Failed"

  - Note spacing/indentation conventions

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

3.6 The if…else Selection Statement

- C code:

```
if ( grade >= 60 )
   printf( "Passed\n");
else
   printf( "Failed\n");
```

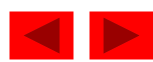
- Ternary conditional operator (?:)
  - Takes three arguments (condition, value if true, value if false)
  - Our pseudocode could be written:

```
printf( "%s\n", grade >= 60 ? "Passed" :
  "Failed" );
```

  - Or it could have been written:

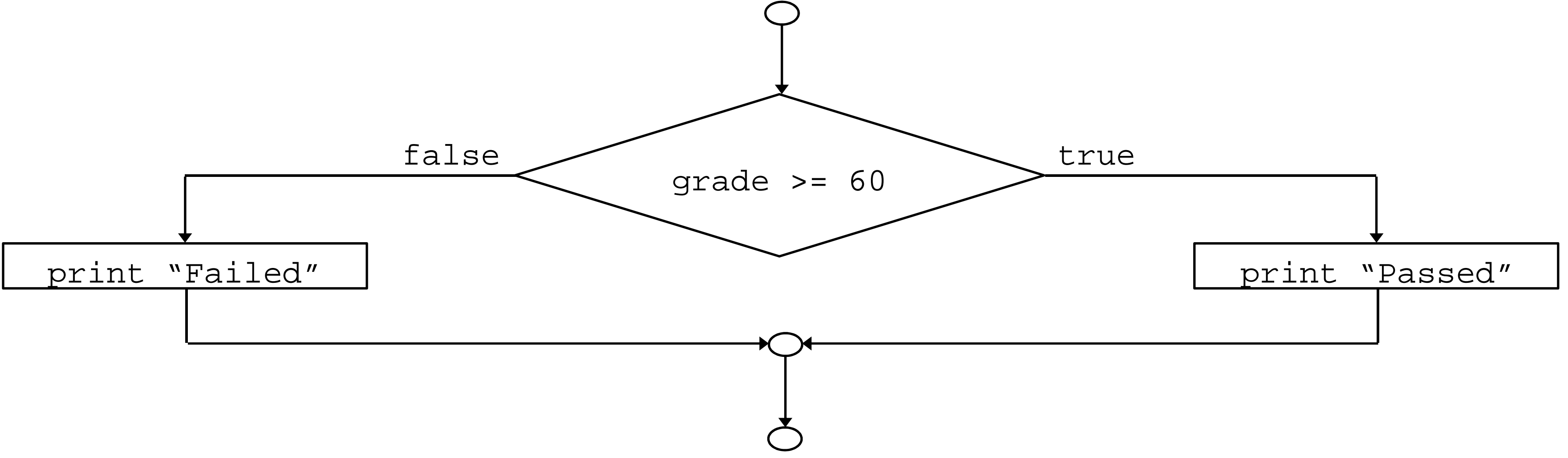
```
grade >= 60 ? printf( “Passed\n” ) : printf(
  “Failed\n” );
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



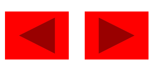
3.6 The if…else Selection Statement

- Flow chart of the if…else selection statement



- Nested if…else statements
  - Test for multiple cases by placing if…else selection statements inside if…else selection statement
  - Once condition is met, rest of statements skipped
  - Deep indentation usually not used in practice

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

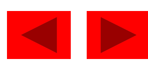


3.6 The `if...else` Selection Statement

- Pseudocode for a nested `if...else` statement

If student's grade is greater than or equal to 90
Print "A"
else
If student's grade is greater than or equal to 80
Print "B"
else
If student's grade is greater than or equal to 70
Print "C"
else
If student's grade is greater than or equal to 60
Print "D"
else
Print "F"

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



3.6 The `if...else` Selection Statement

- Compound statement:
  - Set of statements within a pair of braces
  - Example:

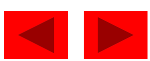
```
if ( grade >= 60 )
   printf( "Passed.\n" );
else {
   printf( "Failed.\n" );
   printf( "You must take this course
      again.\n" );
  }
```

  - Without the braces, the statement

```
printf( "You must take this course
   again.\n" );
```

  would be executed automatically

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



3.6 The `if...else` Selection Statement

- Block:
  - Compound statements with declarations
- Syntax errors
  - Caught by compiler
- Logic errors:
  - Have their effect at execution time
  - Non-fatal: program runs, but has incorrect output
  - Fatal: program exits prematurely

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

3.7 The `while` Repetition Statement

- Repetition structure
  - Programmer specifies an action to be repeated while some condition remains `true`
  - Psuedocode:

    While there are more items on my shopping list

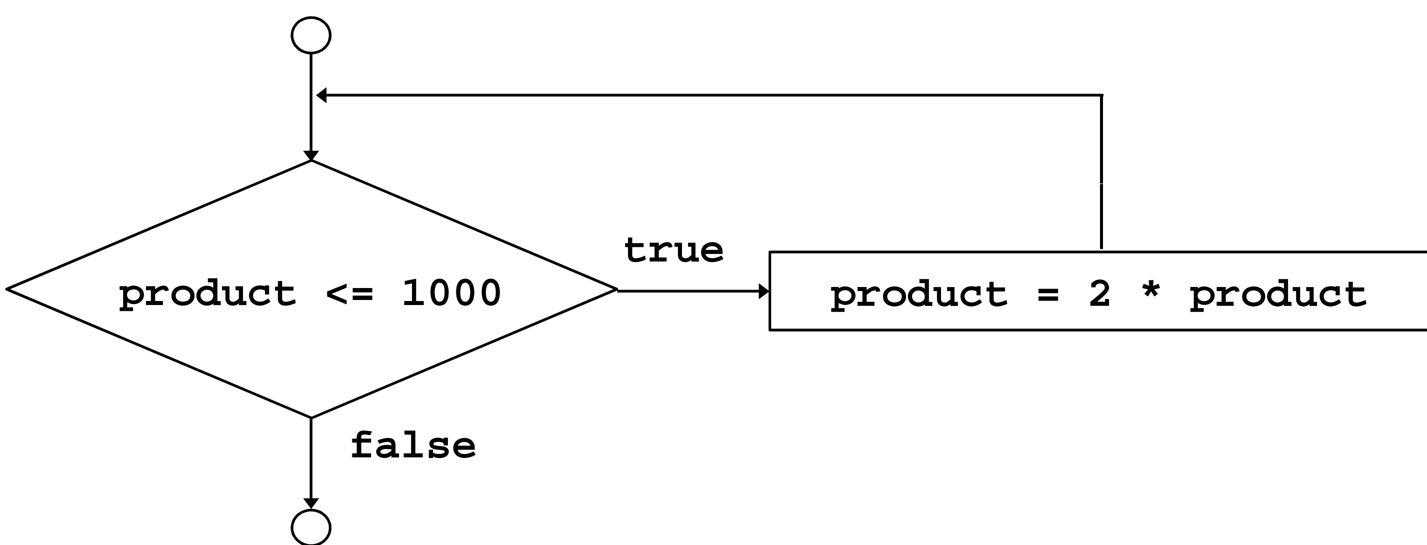
    Purchase next item and cross it off my list
  - `while` loop repeated until condition becomes `false`

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

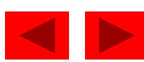
3.7 The `while` Repetition Statement

- Example:

```
int product = 2;
while ( product <= 1000 )
     product = 2 * product;
```



© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



3.8 Formulating Algorithms (Counter-Controlled Repetition)

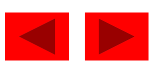
- Counter-controlled repetition
  - Loop repeated until counter reaches a certain value
  - Definite repetition: number of repetitions is known
  - Example: A class of ten students took a quiz. The grades (integers in the range 0 to 100) for this quiz are available to you. Determine the class average on the quiz
  - Pseudocode:

Set total to zero
Set grade counter to one

While grade counter is less than or equal to ten
Input the next grade
Add the grade into the total
Add one to the grade counter

Set the class average to the total divided by ten
Print the class average

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



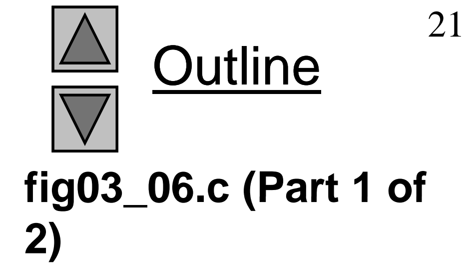


fig03_06.c (Part 1 of 2)

```
1  /* Fig. 3.6: fig03_06.c
2     Class average program with counter-controlled repetition */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main()
7  {
8     int counter; /* number of grade to be entered next */
9     int grade;   /* grade value */
10    int total;   /* sum of grades input by user */
11    int average; /* average of grades */
12
13    /* initialization phase */
14    total = 0;   /* initialize total */
15    counter = 1; /* initialize loop counter */
16
17    /* processing phase */
18    while ( counter <= 10 ) {     /* loop 10 times */
19       printf( "Enter grade: " ); /* prompt for input */
20       scanf( "%d", &grade );     /* read grade from user */
21       total = total + grade;     /* add grade to total */
22       counter = counter + 1;    /* increment counter */
23    } /* end while */
24
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

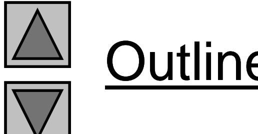


fig03_06.c (Part 2 of 2)

```
25    /* termination phase */
26    average = total / 10;          /* integer division */
27
28    /* display result */
29    printf( "Class average is %d\n", average );
30
31    return 0; /* indicate program ended successfully */
32
33 } /* end function main */
```

Program Output

```
Enter grade: 98
Enter grade: 76
Enter grade: 71
Enter grade: 87
Enter grade: 83
Enter grade: 90
Enter grade: 57
Enter grade: 79
Enter grade: 82
Enter grade: 94
Class average is 81
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

3.9 Formulating Algorithms with Top-Down, Stepwise Refinement

- Problem becomes:

  Develop a class-averaging program that will process an arbitrary number of grades each time the program is run.

  - Unknown number of students
  - How will the program know to end?
- Use sentinel value
  - Also called signal value, dummy value, or flag value
  - Indicates “end of data entry.”
  - Loop ends when user inputs the sentinel value
  - Sentinel value chosen so it cannot be confused with a regular input (such as `-1` in this case)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

3.9 Formulating Algorithms with Top-Down, Stepwise Refinement

- Top-down, stepwise refinement
  - Begin with a pseudocode representation of the *top*:

    Determine the class average for the quiz

  - Divide *top* into smaller tasks and list them in order:

    Initialize variables
    Input, sum and count the quiz grades
    Calculate and print the class average

- Many programs have three phases:
  - Initialization: initializes the program variables
  - Processing: inputs data values and adjusts program variables accordingly
  - Termination: calculates and prints the final results

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

3.9 Formulating Algorithms with Top-Down, Stepwise Refinement

- Refine the initialization phase from *Initialize variables* to:

  Initialize total to zero
  Initialize counter to zero

- Refine *Input, sum and count the quiz grades* to

  Input the first grade (possibly the sentinel)
  While the user has not as yet entered the sentinel
     Add this grade into the running total
     Add one to the grade counter
     Input the next grade (possibly the sentinel)

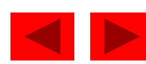
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

3.9 Formulating Algorithms with Top-Down, Stepwise Refinement

- Refine *Calculate and print the class average* to

If the counter is not equal to zero
   Set the average to the total divided by the counter
   Print the average
else
   Print "No grades were entered"

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



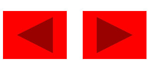
3.9 Formulating Algorithms with Top-Down, Stepwise Refinement

Initialize total to zero
Initialize counter to zero

Input the first grade
While the user has not as yet entered the sentinel
  Add this grade into the running total
  Add one to the grade counter
  Input the next grade (possibly the sentinel)

If the counter is not equal to zero
  Set the average to the total divided by the counter
  Print the average
else
  Print "No grades were entered"

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Outline

fig03_08.c (Part 1 of 2)

```
1  /* Fig. 3.8: fig03_08.c
2     Class average program with sentinel-controlled repetition */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main()
7  {
8     int counter;   /* number of grades entered */
9     int grade;     /* grade value */
10    int total;     /* sum of grades */
11
12    float average; /* number with decimal point for average */
13
14    /* initialization phase */
15    total = 0;     /* initialize total */
16    counter = 0;   /* initialize loop counter */
17
18    /* processing phase */
19    /* get first grade from user */
20    printf( "Enter grade, -1 to end: " );    /* prompt for input */
21    scanf( "%d", &grade );                   /* read grade from user */
22
23    /* loop while sentinel value not yet read from user */
24    while ( grade != -1 ) {
25       total = total + grade;                /* add grade to total */
26       counter = counter + 1;                /* increment counter */
27
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

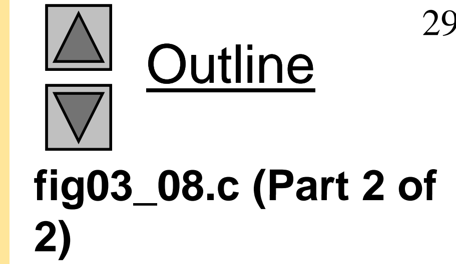


fig03_08.c (Part 2 of 2)

```
28       printf( "Enter grade, -1 to end: " ); /* prompt for input */
29       scanf("%d", &grade);                  /* read next grade */
30    } /* end while */
31
32    /* termination phase */
33    /* if user entered at least one grade */
34    if ( counter != 0 ) {
35
36       /* calculate average of all grades entered */
37       average = ( float ) total / counter;
38
39       /* display average with two digits of precision */
40       printf( "Class average is %.2f\n", average );
41    } /* end if */
42    else { /* if no grades were entered, output message */
43       printf( "No grades were entered\n" );
44    } /* end else */
45
46    return 0; /* indicate program ended successfully */
47
48 } /* end function main */
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Program Output

```
Enter grade, -1 to end: 75
Enter grade, -1 to end: 94
Enter grade, -1 to end: 97
Enter grade, -1 to end: 88
Enter grade, -1 to end: 70
Enter grade, -1 to end: 64
Enter grade, -1 to end: 83
Enter grade, -1 to end: 89
Enter grade, -1 to end: -1
Class average is 82.50
```

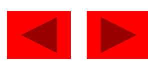
```
Enter grade, -1 to end: -1
No grades were entered
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

3.10 Nested control structures

- Problem
  - A college has a list of test results (1 = pass, 2 = fail) for 10 students
  - Write a program that analyzes the results
    - If more than 8 students pass, print "Raise Tuition"
- Notice that
  - The program must process 10 test results
    - Counter-controlled loop will be used
  - Two counters can be used
    - One for number of passes, one for number of fails
  - Each test result is a number—either a 1 or a 2
    - If the number is not a 1, we assume that it is a 2

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



3.10 Nested control structures

- Top level outline

  Analyze exam results and decide if tuition should be raised

- First Refinement

  Initialize variables

  Input the ten quiz grades and count passes and failures

  Print a summary of the exam results and decide if tuition should be raised

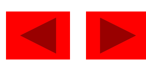
- Refine *Initialize variables* to

  Initialize passes to zero

  Initialize failures to zero

  Initialize student counter to one

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



3.10 Nested control structures

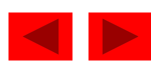
- Refine *Input the ten quiz grades and count passes and failures* to

  While student counter is less than or equal to ten
  Input the next exam result
  If the student passed
  Add one to passes
  else
  Add one to failures
  Add one to student counter

- Refine *Print a summary of the exam results and decide if tuition should be raised* to

  Print the number of passes
  Print the number of failures
  If more than eight students passed
  Print "Raise tuition"

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



3.10 Nested control structures

Initialize passes to zero
Initialize failures to zero
Initialize student to one

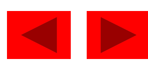
While student counter is less than or equal to ten
    Input the next exam result

    If the student passed
        Add one to passes
    else
        Add one to failures

    Add one to student counter

Print the number of passes
Print the number of failures
If more than eight students passed
    Print "Raise tuition"

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Outline

fig03_10.c (Part 1 of 2)

```
1  /* Fig. 3.10: fig03_10.c
2     Analysis of examination results */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main()
7  {
8     /* initialize variables in definitions */
9     int passes = 0;   /* number of passes */
10    int failures = 0; /* number of failures */
11    int student = 1;  /* student counter */
12    int result;       /* one exam result */
13
14    /* process 10 students using counter-controlled loop */
15    while ( student <= 10 ) {
16
17       /* prompt user for input and obtain value from user */
18       printf( "Enter result ( 1=pass,2=fail ): " );
19       scanf( "%d", &result );
20
21       /* if result 1, increment passes */
22       if ( result == 1 ) {
23          passes = passes + 1;
24       } /* end if */
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

fig03_10.c (Part 2 of 2)

```
25       else { /* otherwise, increment failures */
26          failures = failures + 1;
27       } /* end else */
28
29       student = student + 1; /* increment student counter */
30    } /* end while */
31
32    /* termination phase; display number of passes and failures */
33    printf( "Passed %d\n", passes );
34    printf( "Failed %d\n", failures );
35
36    /* if more than eight students passed, print "raise tuition" */
37    if ( passes > 8 ) {
38       printf( "Raise tuition\n" );
39    } /* end if */
40
41    return 0; /* indicate program ended successfully */
42
43 } /* end function main */
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 2
Enter Result (1=pass,2=fail): 2
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 2
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 2
Passed 6
Failed 4
```

```
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 2
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 1
Passed 9
Failed 1
Raise tuition
```


Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

3.11 Assignment Operators

- Assignment operators abbreviate assignment expressions

  `c = c + 3;`

  can be abbreviated as `c += 3;` using the addition assignment operator

- Statements of the form

  variable = *variable operator expression*;

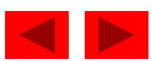
  can be rewritten as

  variable operator= *expression*;

- Examples of other assignment operators:

```
d -= 4      (d = d - 4)
e *= 5      (e = e * 5)
f /= 3      (f = f / 3)
g %= 9      (g = g % 9)
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



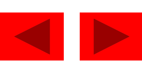
3.11 Assignment Operators

Assume: `int c = 3, d = 5, e = 4, f = 6, g = 12;`

Assignment operator	Sample expression	Explanation	Assigns
`+=`	`c += 7`	`c = c + 7`	10 to c
`-=`	`d -= 4`	`d = d - 4`	1 to d
`*=`	`e *= 5`	`e = e * 5`	20 to e
`/=`	`f /= 3`	`f = f / 3`	2 to `f`
`%=`	`g %= 9`	`g = g % 9`	3 to g

Fig. 3.11 Arithmetic assignment operators.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



3.12 Increment and Decrement Operators

- Increment operator (`++`)
  - Can be used instead of `c+=1`
- Decrement operator (`--`)
  - Can be used instead of `c-=1`
- Preincrement
  - Operator is used before the variable (`++c` or `--c`)
  - Variable is changed before the expression it is in is evaluated
- Postincrement
  - Operator is used after the variable (`c++` or `c--`)
  - Expression executes before the variable is changed

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

3.12 Increment and Decrement Operators

- If `c` equals 5, then

```
printf( "%d", ++c );
```

  – Prints 6

```
printf( "%d", c++ );
```

  – Prints 5

  – In either case, `c` now has the value of 6

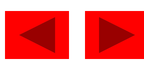
- When variable not in an expression
  – Preincrementing and postincrementing have the same effect

```
++c;
printf( “%d”, c );
```

  – Has the same effect as

```
c++;
printf( “%d”, c );
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



3.12 Increment and Decrement Operators

Operator	Sample expression	Explanation
`++`	`++a`	Increment `a` by 1 then use the new value of `a` in the expression in which `a` resides.
`++`	`a++`	Use the current value of `a` in the expression in which `a` resides, then increment `a` by 1.
`--`	`--b`	Decrement `b` by 1 then use the new value of `b` in the expression in which `b` resides.
`--`	`b--`	Use the current value of `b` in the expression in which `b` resides, then decrement `b` by 1.

Fig. 3.12 The increment and decrement operators

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

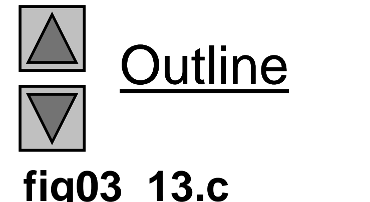


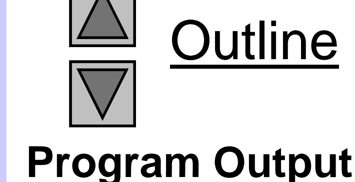
fig03_13.c

```
1  /* Fig. 3.13: fig03_13.c
2     Preincrementing and postincrementing */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main()
7  {
8     int c;                  /* define variable */
9
10    /* demonstrate postincrement */
11    c = 5;                  /* assign 5 to c */
12    printf( "%d\n", c );    /* print 5 */
13    printf( "%d\n", c++ );  /* print 5 then postincrement */
14    printf( "%d\n\n", c );  /* print 6 */
15
16    /* demonstrate preincrement */
17    c = 5;                  /* assign 5 to c */
18    printf( "%d\n", c );    /* print 5 */
19    printf( "%d\n", ++c );  /* preincrement then print 6 */
20    printf( "%d\n", c );    /* print 6 */
21
22    return 0; /* indicate program ended successfully */
23
24 } /* end function main */
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
5
5
6

5
6
6
```



Program Output

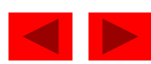
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

3.12 Increment and Decrement Operators

Operators					Associativity	Type
++	--	+	-	(type)	right to left	unary
*	/	%			left to right	multiplicative
+	-				left to right	additive
<	<=	>	>=		left to right	relational
==	!=				left to right	equality
?:					right to left	conditional
=	+=	-=	*=	/=	right to left	assignment

Fig. 3.14 Precedence of the operators encountered so far in the text.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

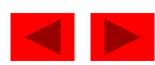


Chapter 4 – C Program Control

Outline

4.1 Introduction
4.2 The Essentials of Repetition
4.3 Counter-Controlled Repetition
4.4 The for Repetition Statement
4.5 The for Statement: Notes and Observations
4.6 Examples Using the for Statement
4.7 The switch Multiple-Selection Statement
4.8 The do…while Repetition Statement
4.9 The break and continue Statements
4.10 Logical Operators
4.11 Confusing Equality (==) and Assignment (=) Operators
4.12 Structured Programming Summary

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Objectives

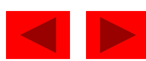
- In this chapter, you will learn:
  - To be able to use the `for` and `do…while` repetition statements.
  - To understand multiple selection using the `switch` selection statement.
  - To be able to use the `break` and `continue` program control statements
  - To be able to use the logical operators.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

4.1 Introduction

- This chapter introduces
  - Additional repetition control structures
    - `for`
    - `Do…while`
  - `switch` multiple selection statement
  - `break` statement
    - Used for exiting immediately and rapidly from certain control structures
  - `continue` statement
    - Used for skipping the remainder of the body of a repetition structure and proceeding with the next iteration of the loop

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



4.2 The Essentials of Repetition

- Loop
  - Group of instructions computer executes repeatedly while some condition remains `true`
- Counter-controlled repetition
  - Definite repetition: know how many times loop will execute
  - Control variable used to count repetitions
- Sentinel-controlled repetition
  - Indefinite repetition
  - Used when number of repetitions not known
  - Sentinel value indicates "end of data"

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

4.3 Essentials of Counter-Controlled Repetition

- Counter-controlled repetition requires
  - The name of a control variable (or loop counter)
  - The initial value of the control variable
  - An increment (or decrement) by which the control variable is modified each time through the loop
  - A condition that tests for the final value of the control variable (i.e., whether looping should continue)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

4.3 Essentials of Counter-Controlled Repetition

- Example:

```
int counter = 1;             // initialization
while ( counter <= 10 ) { // repetition condition
   printf( "%d\n", counter );
   ++counter;                // increment
}
```

  - The statement

    ```
    int counter = 1;
    ```

    - Names `counter`
    - Defines it to be an integer
    - Reserves space for it in memory
    - Sets it to an initial value of `1`

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

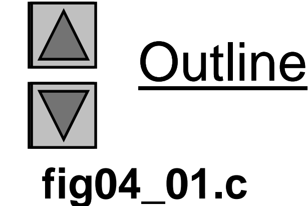


fig04_01.c

```
1  /* Fig. 4.1: fig04_01.c
2     Counter-controlled repetition */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main()
7  {
8     int counter = 1;              /* initialization */
9
10    while ( counter <= 10 ) {     /* repetition condition */
11       printf ( "%d\n", counter ); /* display counter */
12       ++counter;                 /* increment */
13    } /* end while */
14
15    return 0; /* indicate program ended successfully */
16
17 } /* end function main */
```

Program Output

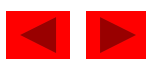
```
1
2
3
4
5
6
7
8
9
10
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

4.3 Essentials of Counter-Controlled Repetition

- Condensed code
  - C Programmers would make the program more concise
  - Initialize `counter` to `0`
    - ```
      while ( ++counter <= 10 )
            printf( "%d\n, counter );
      ```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.


      ```

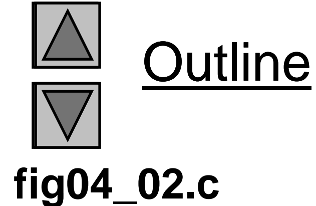
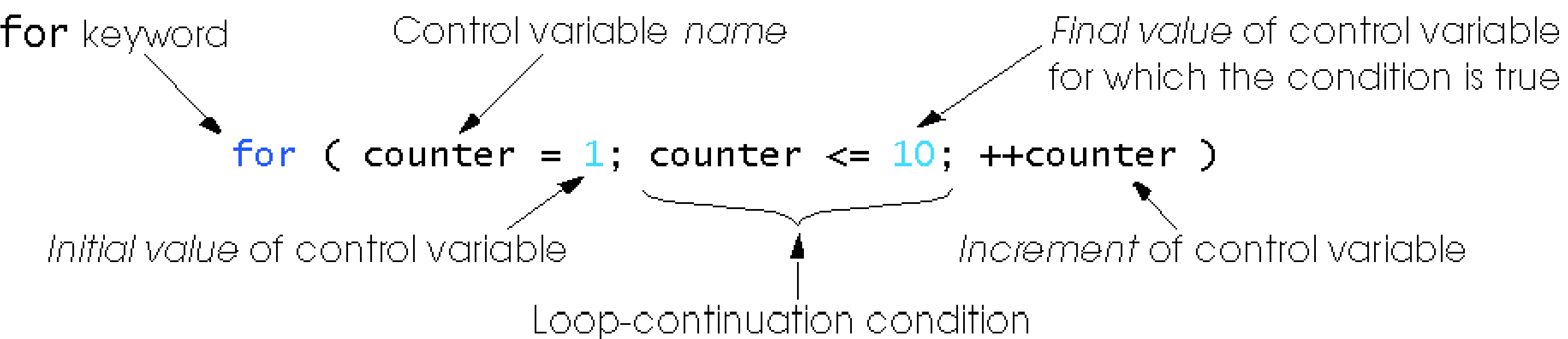


fig04_02.c

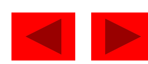
```
1  /* Fig. 4.2: fig04_02.c
2     Counter-controlled repetition with the for statement */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main()
7  {
8     int counter; /* define counter */
9
10    /* initialization, repetition condition, and increment
11       are all included in the for statement header. */
12    for ( counter = 1; counter <= 10; counter++ ) {
13       printf( "%d\n", counter );
14    } /* end for */
15
16    return 0; /* indicate program ended successfully */
17
18 } /* end function main */
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

4.4 The `for` Repetition Statement



© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



4.4 The `for` Repetition Statement

- Format when using `for` loops

  `for` (*initialization*; *loopContinuationTest*; *increment*)
  statement

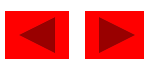
- Example:

```
for( int counter = 1; counter <= 10; counter++ )
   printf( "%d\n", counter );
```

  – Prints the integers from one to ten

No semicolon (`;`) after last expression

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



4.4 The `for` Repetition Statement

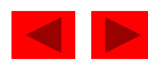
- For loops can usually be rewritten as while loops:

```
initialization;
while ( loopContinuationTest ) {
  statement;
  increment;
}
```

- Initialization and increment
  - Can be comma-separated lists
  - Example:

```
for (int i = 0, j = 0;  j + i <= 10; j++, i++)
   printf( "%d\n", j + i );
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



4.5 The `for` Statement : Notes and Observations

- Arithmetic expressions
  - Initialization, loop-continuation, and increment can contain arithmetic expressions. If `x` equals `2` and `y` equals `10`

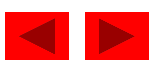
```
for ( j = x; j <= 4 * x * y; j += y / x )
```

is equivalent to

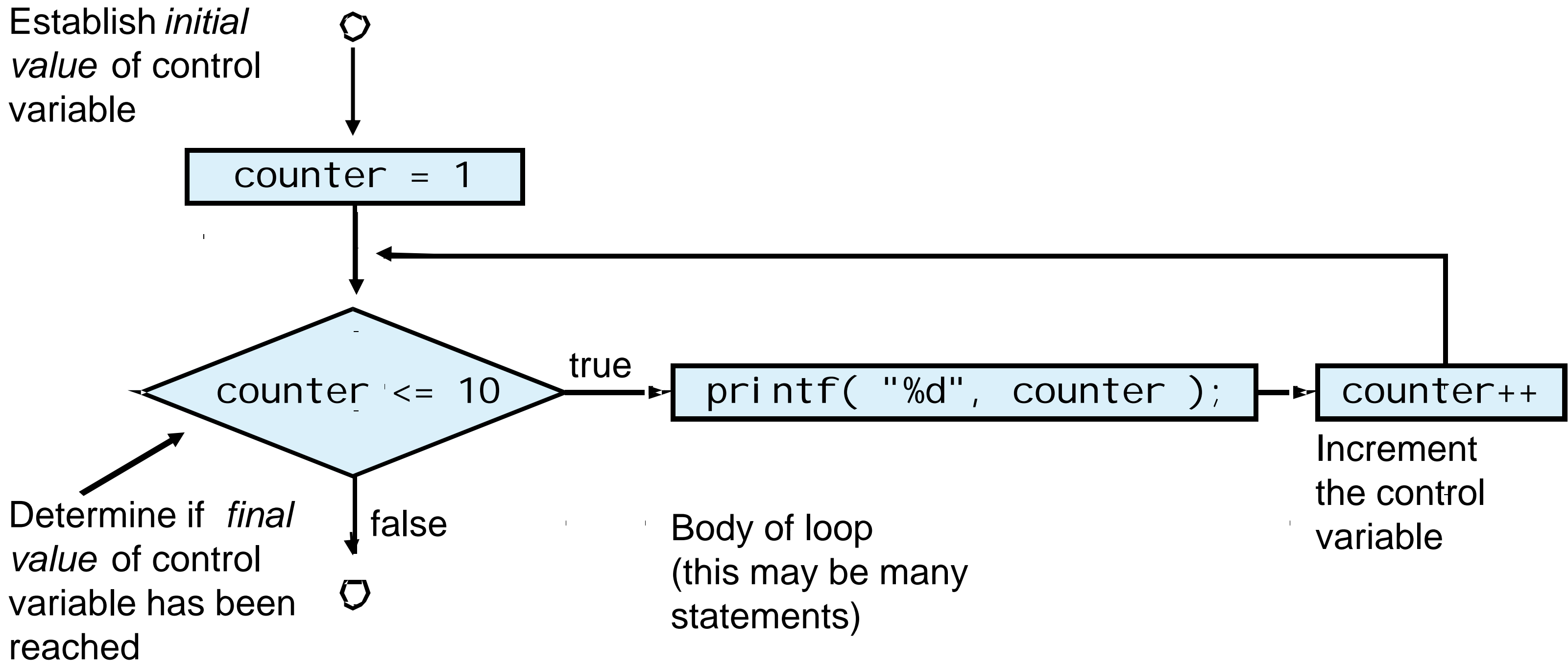
```
for ( j = 2; j <= 80; j += 5 )
```

- Notes about the `for` statement:
  - "Increment" may be negative (decrement)
  - If the loop continuation condition is initially `false`
    - The body of the `for` statement is not performed
    - Control proceeds with the next statement after the `for` statement
  - Control variable
    - Often printed or used inside for body, but not necessary

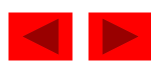
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



4.5 The `for` Statement : Notes and Observations



© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



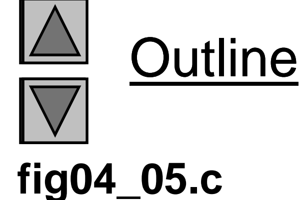


fig04_05.c

```
1  /* Fig. 4.5: fig04_05.c
2     Summation with for */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main()
7  {
8     int sum = 0; /* initialize sum */
9     int number;  /* number to be added to sum */
10
11    for ( number = 2; number <= 100; number += 2 ) {
12       sum += number; /* add number to sum */
13    } /* end for */
14
15    printf( "Sum is %d\n", sum ); /* output sum */
16
17    return 0; /* indicate program ended successfully */
18
19 } /* end function main */
```

Program Output

```
Sum is 2550
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

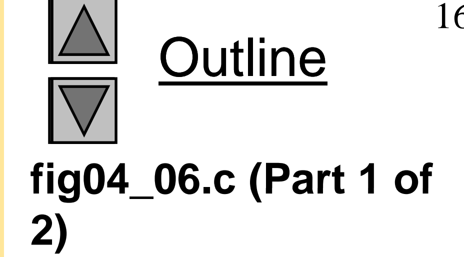


fig04_06.c (Part 1 of 2)

```
1  /* Fig. 4.6: fig04_06.c
2     Calculating compound interest */
3  #include <stdio.h>
4  #include <math.h>
5
6  /* function main begins program execution */
7  int main()
8  {
9     double amount;              /* amount on deposit */
10    double principal = 1000.0; /* starting principal */
11    double rate = .05;         /* interest rate */
12    int year;                  /* year counter */
13
14    /* output table column head */
15    printf( "%4s%21s\n", "Year", "Amount on deposit" );
16
17    /* calculate amount on deposit for each of ten years */
18    for ( year = 1; year <= 10; year++ ) {
19
20       /* calculate new amount for specified year */
21       amount = principal * pow( 1.0 + rate, year );
22
23       /* output one table row */
24       printf( "%4d%21.2f\n", year, amount );
25    } /* end for */
26
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
27    return 0; /* indicate program ended successfully */
28
29 } /* end function main */
```

```
Year     Amount on deposit
   1               1050.00
   2               1102.50
   3               1157.63
   4               1215.51
   5               1276.28
   6               1340.10
   7               1407.10
   8               1477.46
   9               1551.33
  10               1628.89
```


fig04_06.c (Part 2 of 2)

Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

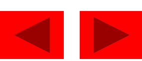
4.7 The switch Multiple-Selection Statement

- switch
  - Useful when a variable or expression is tested for all the values it can assume and different actions are taken
- Format
  - Series of case labels and an optional default case

```
switch ( value ){
    case '1':
        actions
    case '2':
        actions
    default:
        actions
    }
```

  - break; exits from statement

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

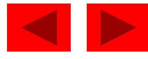


4.7 The `switch` Multiple-Selection Statement

- Flowchart of the `switch` statement

case a
true
case a action(s)
break
false
case b
true
case b action(s)
break
false
.
.
.
case z
true
case z action(s)
break
false
default action(s)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Outline

fig04_07.c (Part 1 of 3)

```
1  /* Fig. 4.7: fig04_07.c
2     Counting letter grades */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main()
7  {
8     int grade;       /* one grade */
9     int aCount = 0; /* number of As */
10    int bCount = 0; /* number of Bs */
11    int cCount = 0; /* number of Cs */
12    int dCount = 0; /* number of Ds */
13    int fCount = 0; /* number of Fs */
14
15    printf( "Enter the letter grades.\n" );
16    printf( "Enter the EOF character to end input.\n" );
17
18    /* loop until user types end-of-file key sequence */
19    while ( ( grade = getchar() ) != EOF ) {
20
21       /* determine which grade was input */
22       switch ( grade ) { /* switch nested in while */
23
24          case 'A':     /* grade was uppercase A */
25          case 'a':     /* or lowercase a */
26             ++aCount; /* increment aCount */
27             break;    /* necessary to exit switch */
28
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Outline

fig04_07.c (Part 2 of 3)

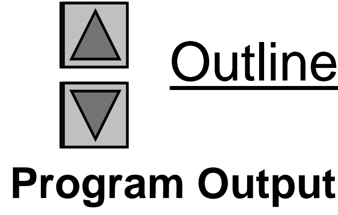
```
29         case 'B':    /* grade was uppercase B */
30         case 'b':    /* or lowercase b */
31            ++bCount; /* increment bCount */
32            break;    /* exit switch */
33
34         case 'C':    /* grade was uppercase C */
35         case 'c':    /* or lowercase c */
36            ++cCount; /* increment cCount */
37            break;    /* exit switch */
38
39         case 'D':    /* grade was uppercase D */
40         case 'd':    /* or lowercase d */
41            ++dCount; /* increment dCount */
42            break;    /* exit switch */
43
44         case 'F':    /* grade was uppercase F */
45         case 'f':    /* or lowercase f */
46            ++fCount; /* increment fCount */
47            break;    /* exit switch */
48
49         case '\n':   /* ignore newlines, */
50         case '\t':   /* tabs, */
51         case ' ':    /* and spaces in input */
52            break;    /* exit switch */
53
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
54          default:     /* catch all other characters */
55             printf( "Incorrect letter grade entered." );
56             printf( " Enter a new grade.\n" );
57             break;    /* optional; will exit switch anyway */
58       } /* end switch */
59
60    } /* end while */
61
62    /* output summary of results */
63    printf( "\nTotals for each letter grade are:\n" );
64    printf( "A: %d\n", aCount ); /* display number of A grades */
65    printf( "B: %d\n", bCount ); /* display number of B grades */
66    printf( "C: %d\n", cCount ); /* display number of C grades */
67    printf( "D: %d\n", dCount ); /* display number of D grades */
68    printf( "F: %d\n", fCount ); /* display number of F grades */
69
70    return 0; /* indicate program ended successfully */
71
72 } /* end function main */
```


fig04_07.c (Part 3 of 3)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Program Output

```
Enter the letter grades.
Enter the EOF character to end input.
a
b
c
C
A
d
f
C
E
Incorrect letter grade entered. Enter a new grade.
D
A
b
^Z

Totals for each letter grade are:
A: 3
B: 2
C: 3
D: 2
F: 1
```

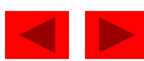
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

4.8 The do…while Repetition Statement

- The do…while repetition statement
  - Similar to the while structure
  - Condition for repetition tested after the body of the loop is performed
    - All actions are performed at least once
  - Format:

```
do {
    statement;
} while ( condition );
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



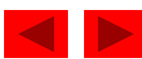
4.8 The do…while Repetition Statement

- Example (letting counter = 1):

```
do {
   printf( "%d  ", counter );
} while (++counter <= 10);
```

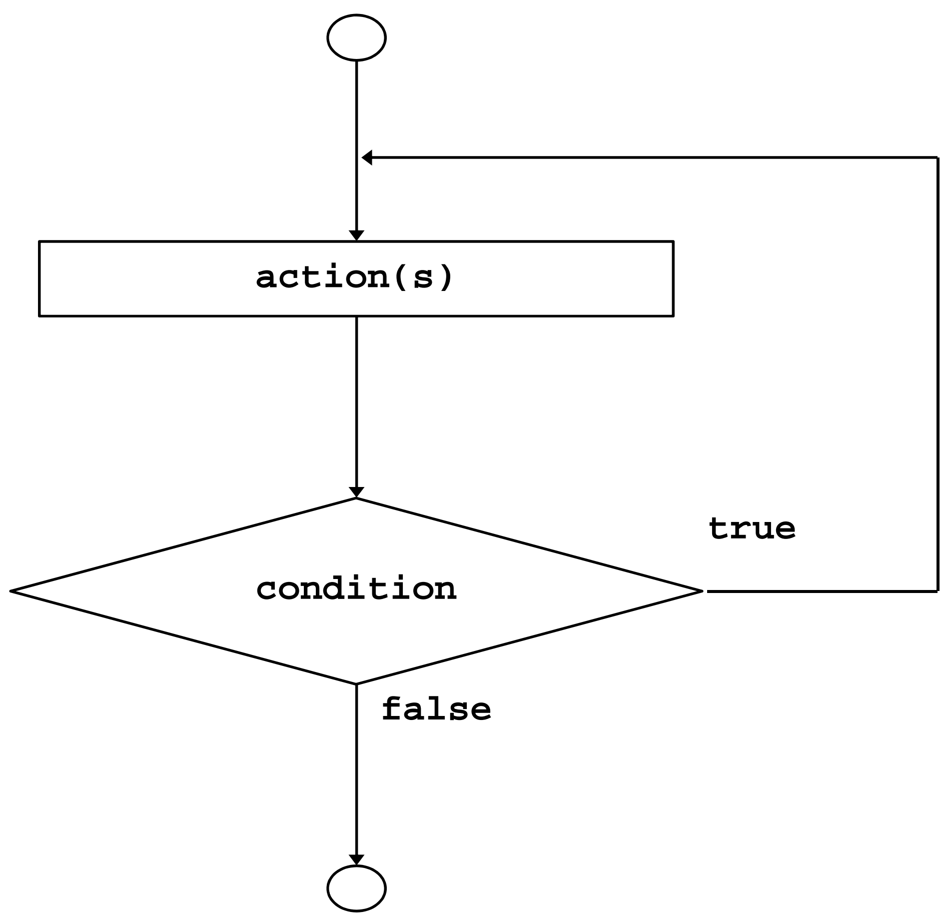
  - Prints the integers from 1 to 10

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

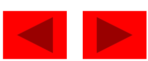


4.8 The do…while Repetition Statement

- Flowchart of the do…while repetition statement



© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



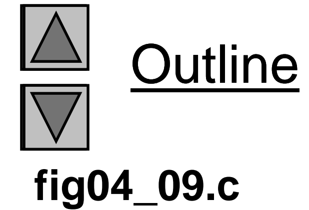


fig04_09.c

```
1  /* Fig. 4.9: fig04_09.c
2     Using the do/while repetition statement */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main()
7  {
8     int counter = 1; /* initialize counter */
9
10    do {
11       printf( "%d  ", counter ); /* display counter */
12    } while ( ++counter <= 10 ); /* end do...while */
13
14    return 0; /* indicate program ended successfully */
15
16 } /* end function main */
```

Program Output

```
1  2  3  4  5  6  7  8  9  10
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

4.9 The `break` and `continue` Statements

- `break`
  - Causes immediate exit from a `while`, `for`, `do…while` or `switch` statement
  - Program execution continues with the first statement after the structure
  - Common uses of the `break` statement
    - Escape early from a loop
    - Skip the remainder of a `switch` statement

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

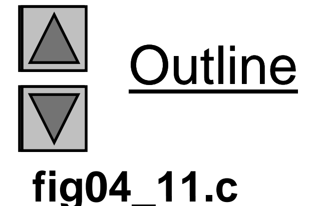


fig04_11.c

```
1  /* Fig. 4.11: fig04_11.c
2     Using the break statement in a for statement */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main()
7  {
8     int x; /* counter */
9
10    /* loop 10 times */
11    for ( x = 1; x <= 10; x++ ) {
12
13       /* if x is 5, terminate loop */
14       if ( x == 5 ) {
15          break; /* break loop only if x is 5 */
16       } /* end if */
17
18       printf( "%d ", x ); /* display value of x */
19    } /* end for */
20
21    printf( "\nBroke out of loop at x == %d\n", x );
22
23    return 0; /* indicate program ended successfully */
24
25 } /* end function main */
```

Program Output

```
1 2 3 4
Broke out of loop at x == 5
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

4.9 The `break` and `continue` Statements

- `continue`
  - Skips the remaining statements in the body of a `while`, `for` or `do…while` statement
    - Proceeds with the next iteration of the loop
  - `while` and `do…while`
    - Loop-continuation test is evaluated immediately after the `continue` statement is executed
  - `for`
    - Increment expression is executed, then the loop-continuation test is evaluated

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

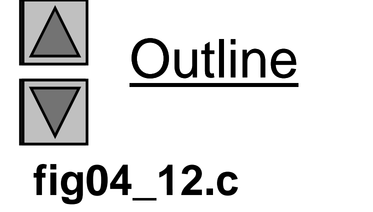


fig04_12.c

```
1  /* Fig. 4.12: fig04_12.c
2     Using the continue statement in a for statement */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main()
7  {
8     int x; /* counter */
9
10    /* loop 10 times */
11    for ( x = 1; x <= 10; x++ ) {
12
13       /* if x is 5, continue with next iteration of loop */
14       if ( x == 5 ) {
15          continue; /* skip remaining code in loop body */
16       } /* end if */
17
18       printf( "%d ", x ); /* display value of x */
19    } /* end for */
20
21    printf( "\nUsed continue to skip printing the value 5\n" );
22
23    return 0; /* indicate program ended successfully */
24
25 } /* end function main */
```

Program Output

```
1 2 3 4 6 7 8 9 10
Used continue to skip printing the value 5
```

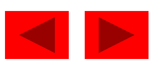
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

4.10 Logical Operators

- && (logical AND)
  - Returns `true` if both conditions are `true`
- || (logical OR)
  - Returns `true` if either of its conditions are `true`
- ! (logical NOT, logical negation)
  - Reverses the truth/falsity of its condition
  - Unary operator, has one operand
- Useful as conditions in loops

Expression	Result
`true && false`	`false`
`true || false`	`true`
`!false`	`true`

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



4.10 Logical Operators

expression1	expression2	expression1 && expression2
0	0	0
0	nonzero	0
nonzero	0	0
nonzero	nonzero	1

Fig. 4.13 Truth table for the && (logical AND) operator.

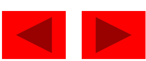
expression1	expression2	expression1 | | expression2
0	0	0
0	nonzero	1
nonzero	0	1
nonzero	nonzero	1

Fig. 4.14 Truth table for the logical OR (||) operator.

expression	! expression
0	1
nonzero	0

Fig. 4.15 Truth table for operator ! (logical negation).

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

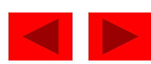


4.10 Logical Operators

Operators						Associativity	Type
++	--	+	-	!	(type)	right to left	unary
*	/	%				left to right	multiplicative
+	-					left to right	additive
<	<=	>	>=			left to right	relational
==	!=					left to right	equality
&&						left to right	logical AND
||						left to right	logical OR
?:						right to left	conditional
=	+=	-=	*=	/=	%=	right to left	assignment
,						left to right	comma

Fig. 4.16 Operator precedence and associativity.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



4.11 Confusing Equality (==) and Assignment (=) Operators

- Dangerous error
  - Does not ordinarily cause syntax errors
  - Any expression that produces a value can be used in control structures
  - Nonzero values are `true`, zero values are `false`
  - Example using ==:

```
if ( payCode == 4 )
   printf( "You get a bonus!\n" );
```

    - Checks `payCode`, if it is 4 then a bonus is awarded

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

4.11 Confusing Equality (==) and Assignment (=) Operators

– Example, replacing == with =:

```
if ( payCode = 4 )
   printf( "You get a bonus!\n" );
```

- This sets payCode to 4
- 4 is nonzero, so expression is true, and bonus awarded no matter what the payCode was

– Logic error, not a syntax error

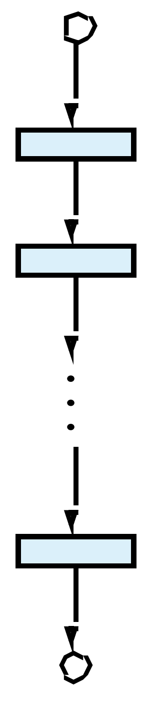
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

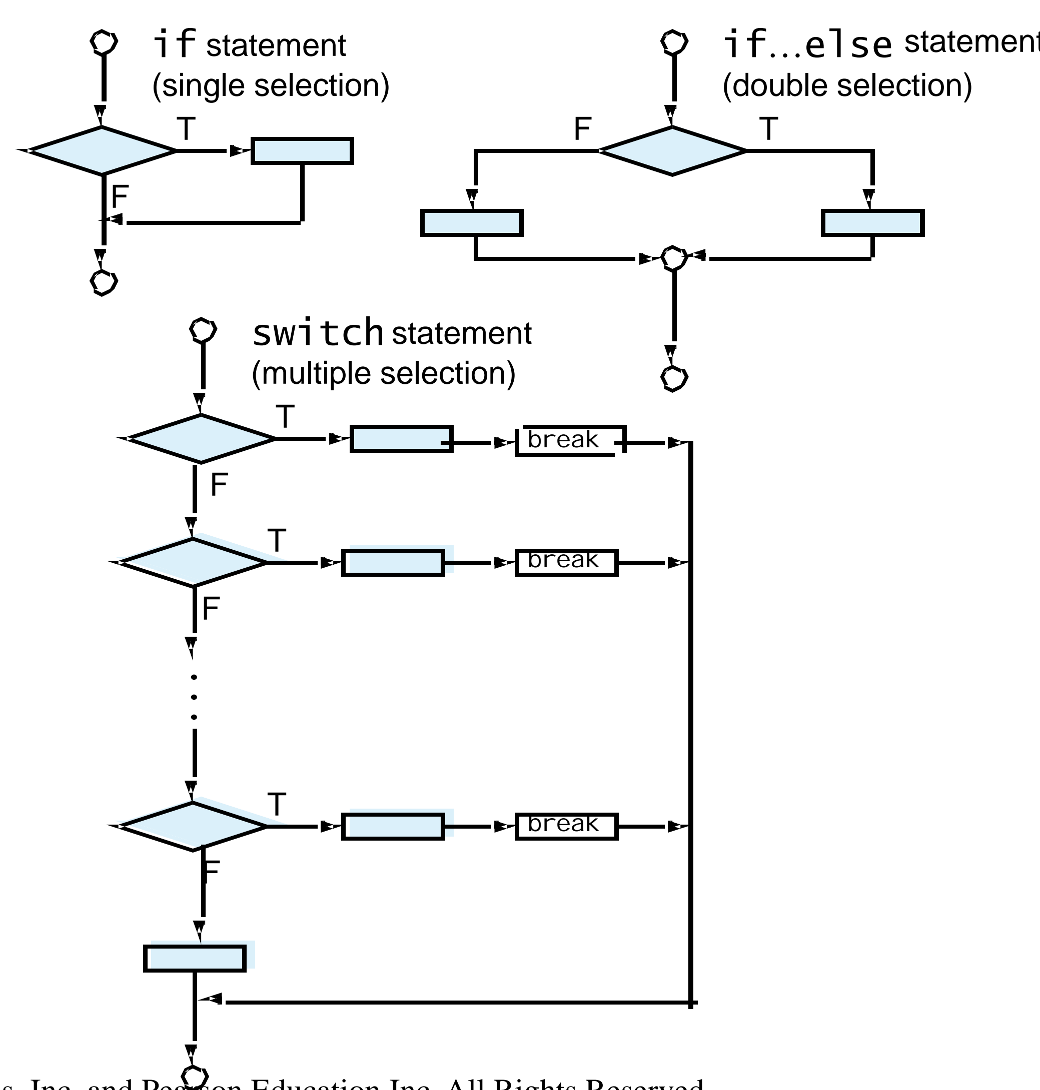
4.11 Confusing Equality (==) and Assignment (=) Operators

- lvalues
  - Expressions that can appear on the left side of an equation
  - Their values can be changed, such as variable names
    - `x = 4;`
- rvalues
  - Expressions that can only appear on the right side of an equation
  - Constants, such as numbers
    - Cannot write `4 = x;`
    - Must write `x = 4;`
  - lvalues can be used as rvalues, but not vice versa
    - `y = x;`

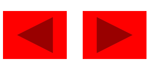
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

4.12 Structured-Programming Summary



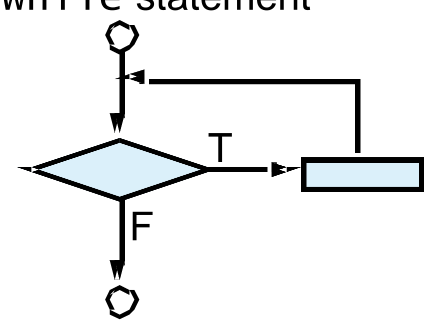


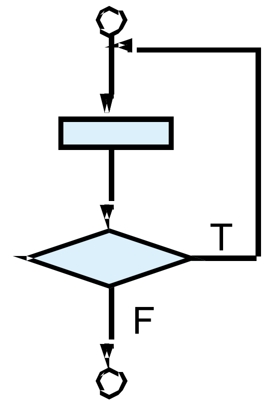
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



4.12 Structured-Programming Summary

Repetition



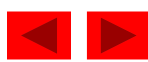


for statement

T

F

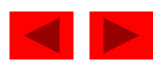
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



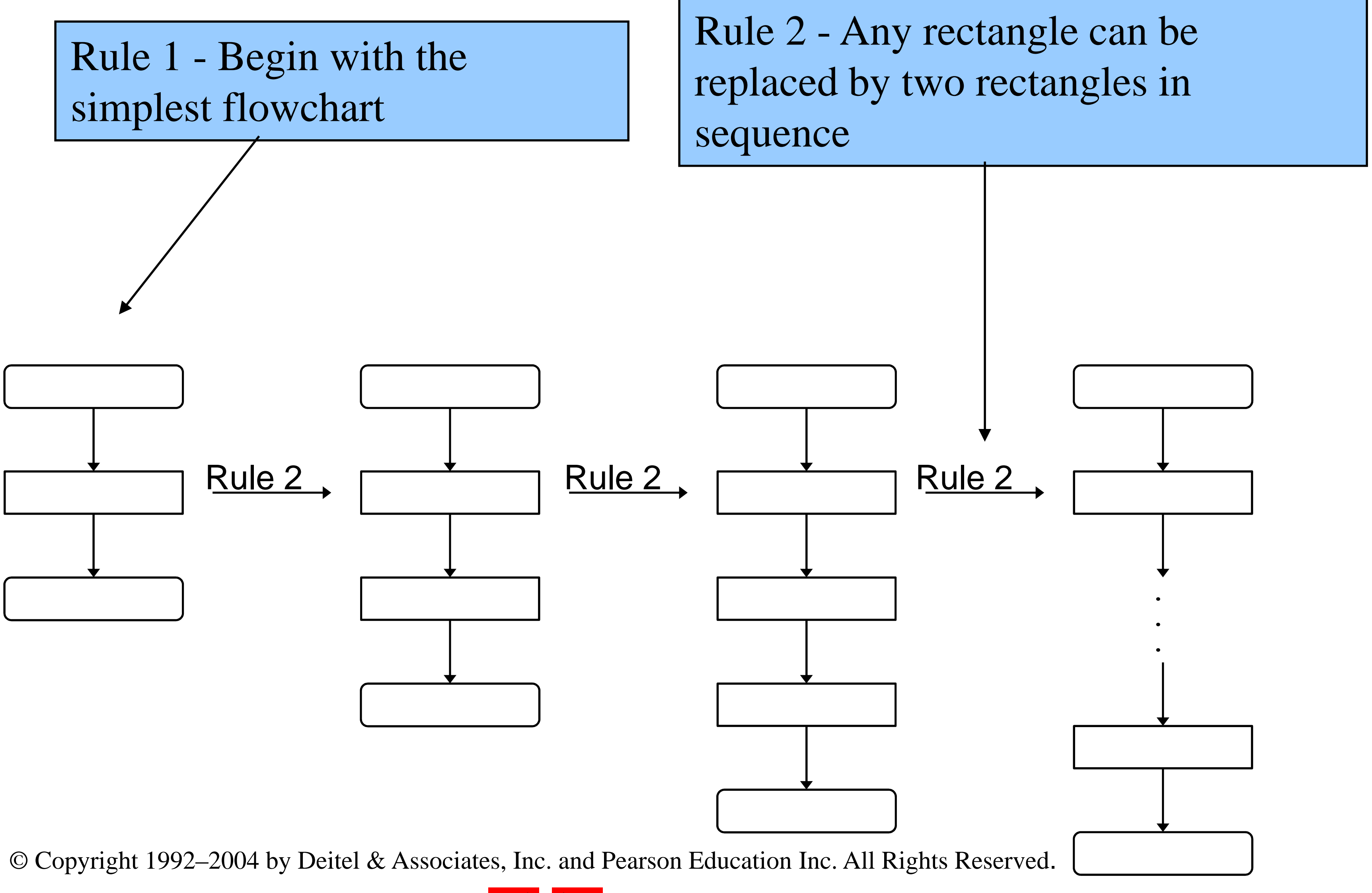
4.12 Structured-Programming Summary

- Structured programming
  - Easier than unstructured programs to understand, test, debug and, modify programs
- Rules for structured programming
  - Rules developed by programming community
  - Only single-entry/single-exit control structures are used
  - Rules:
    1. Begin with the "simplest flowchart"
    2. Stacking rule: Any rectangle (action) can be replaced by two rectangles (actions) in sequence
    3. Nesting rule: Any rectangle (action) can be replaced by any control structure (sequence, `if`, `if…else`, `switch`, `while`, `do…while` or `for`)
    4. Rules 2 and 3 can be applied in any order and multiple times

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



4.12 Structured-Programming Summary



© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

4.12 Structured-Programming Summary

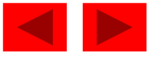
Rule 3 - Replace any rectangle with a control structure

Rule 3

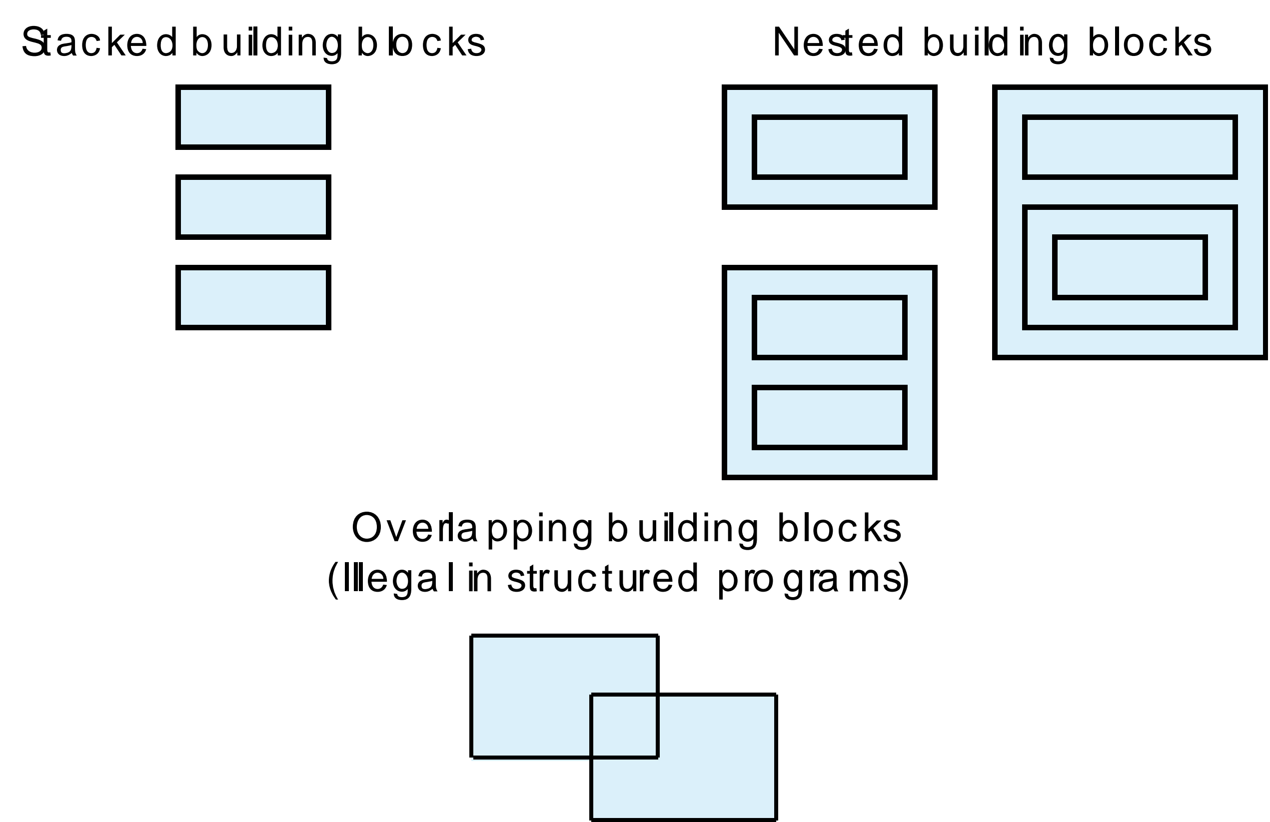
Rule 3

Rule 3

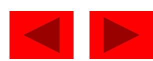
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



4.12 Structured-Programming Summary

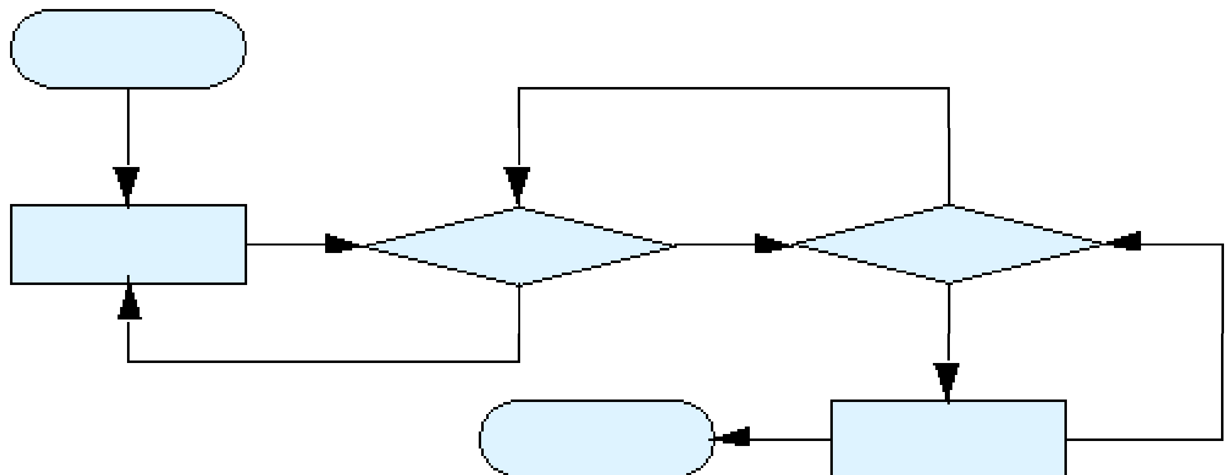


© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

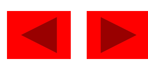


4.12 Structured-Programming Summary

Figure 4.23 An unstructured flowchart.



© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



4.12 Structured-Programming Summary

- All programs can be broken down into 3 controls
  - Sequence – handled automatically by compiler
  - Selection – `if`, `if…else` or `switch`
  - Repetition – `while`, `do…while` or `for`
    - Can only be combined in two ways
      - Nesting (rule 3)
      - Stacking (rule 2)
  - Any selection can be rewritten as an `if` statement, and any repetition can be rewritten as a `while` statement

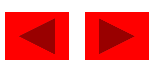
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Chapter 5 - Functions

Outline

5.1 Introduction
5.2 Program Modules in C
5.3 Math Library Functions
5.4 Functions
5.5 Function Definitions
5.6 Function Prototypes
5.7 Header Files
5.8 Calling Functions: Call by Value and Call by Reference
5.9 Random Number Generation
5.10 Example: A Game of Chance
5.11 Storage Classes
5.12 Scope Rules
5.13 Recursion
5.14 Example Using Recursion: The Fibonacci Series
5.15 Recursion vs. Iteration

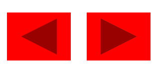
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Objectives

- In this chapter, you will learn:
  - To understand how to construct programs modularly from small pieces called functions..
  - To introduce the common math functions available in the C standard library.
  - To be able to create new functions.
  - To understand the mechanisms used to pass information between functions.
  - To introduce simulation techniques using random number generation.
  - To understand how to write and use functions that call themselves.

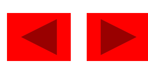
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



5.1 Introduction

- Divide and conquer
  - Construct a program from smaller pieces or components
    - These smaller pieces are called modules
  - Each piece more manageable than the original program

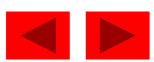
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



5.2 Program Modules in C

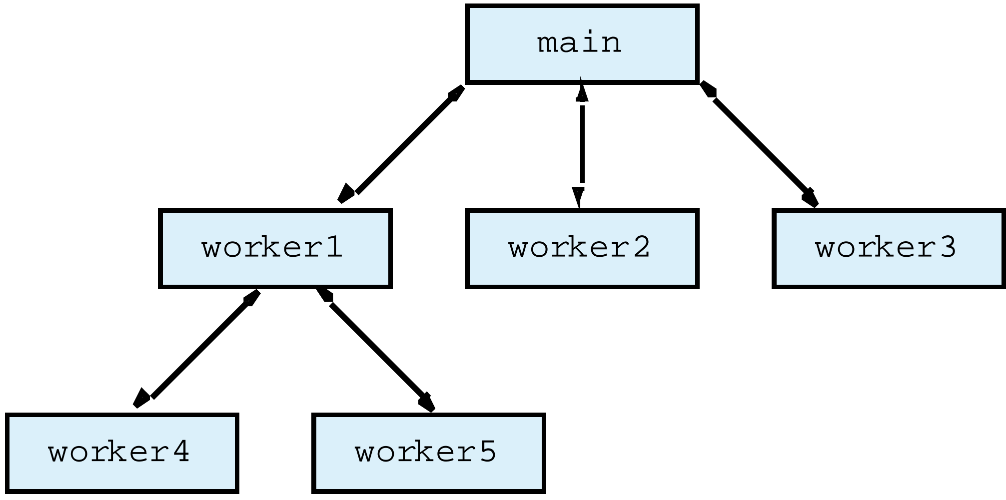
- Functions
  - Modules in C
  - Programs combine user-defined functions with library functions
    - C standard library has a wide variety of functions
- Function calls
  - Invoking functions
    - Provide function name and arguments (data)
    - Function performs operations or manipulations
    - Function returns results
  - Function call analogy:
    - Boss asks worker to complete task
      - Worker gets information, does task, returns result
      - Information hiding: boss does not know details

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

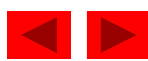


5.2 Program Modules in C

Fig. 5.1 Hierarchical boss function/worker function relationship.



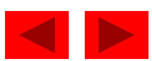
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



5.3 Math Library Functions

- Math library functions
  - perform common mathematical calculations
  - `#include <math.h>`
- Format for calling functions
  - `FunctionName(` *`argument`* `);`
    - If multiple arguments, use comma-separated list
  - `printf( "%.2f", sqrt( 900.0 ) );`
    - Calls function `sqrt`, which returns the square root of its argument
    - All math functions return data type `double`
  - Arguments may be constants, variables, or expressions

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



5.3 Math Library Functions

Function	Description	Example
`sqrt( x )`	square root of x	`sqrt( 900.0 )` is `30.0` `sqrt( 9.0 )` is `3.0`
`exp( x )`	exponential function e^x	`exp( 1.0 )` is `2.718282` `exp( 2.0 )` is `7.389056`
`log( x )`	natural logarithm of x (base e)	`log( 2.718282 )` is `1.0` `log( 7.389056 )` is `2.0`

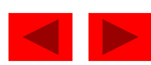
`log10( x )`	logarithm of x (base 10)	`log10( 1.0 )` is `0.0`
		`log10( 10.0 )` is `1.0`
		`log10( 100.0 )` is `2.0`

`fabs( x )`	absolute value of x	`fabs( 5.0 )` is `5.0`
		`fabs( 0.0 )` is `0.0`
		`fabs( -5.0 )` is `5.0`

`ceil( x )`	rounds x to the smallest integer not less than x	`ceil( 9.2 )` is `10.0` `ceil( -9.8 )` is `-9.0`
`floor( x )`	rounds x to the largest integer not greater than x	`floor( 9.2 )` is `9.0` `floor( -9.8 )` is `-10.0`
`pow( x, y )`	x raised to power y (x^y)	`pow( 2, 7 )` is `128.0` `pow( 9, .5 )` is `3.0`
`fmod( x, y )`	remainder of x/y as a floating point number	`fmod( 13.657, 2.333 )` is `1.992`
`sin( x )`	trigonometric sine of x (x in radians)	`sin( 0.0 )` is `0.0`
`cos( x )`	trigonometric cosine of x (x in radians)	`cos( 0.0 )` is `1.0`
`tan( x )`	trigonometric tangent of x (x in radians)	`tan( 0.0 )` is `0.0`

Fig. 5.2 Commonly used math library functions.

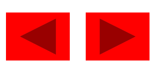
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



5.4 Functions

- Functions
  - Modularize a program
  - All variables defined inside functions are local variables
    - Known only in function defined
  - Parameters
    - Communicate information between functions
    - Local variables
- Benefits of functions
  - Divide and conquer
    - Manageable program development
  - Software reusability
    - Use existing functions as building blocks for new programs
    - Abstraction - hide internal details (library functions)
  - Avoid code repetition

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



5.5 Function Definitions

- Function definition format

```
return-value-type  function-name( parameter-list )
  {
    declarations and statements
  }
```

  – Function-name: any valid identifier
  – Return-value-type: data type of the result (default `int`)
    - `void` – indicates that the function returns nothing
  – Parameter-list: comma separated list, declares parameters
    - A type must be listed explicitly for each parameter unless, the parameter is of type `int`

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

5.5 Function Definitions

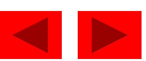
- Function definition format (continued)

  return-value-type function-name(parameter-list)
  `{`
  declarations and statements
  `}`

  – Definitions and statements: function body (block)
    - Variables can be defined inside blocks (can be nested)
    - Functions can not be defined inside other functions

  – Returning control
    - If nothing returned
      – `return;`
      – or, until reaches right brace
    - If something returned
      – `return` *expression* `;`

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



```
1  /* Fig. 5.3: fig05_03.c
2     Creating and using a programmer-defined function */
3  #include <stdio.h>
4
5  int square( int y );   /* function prototype */
6
7  /* function main begins program execution */
8  int main()
9  {
10    int x; /* counter */
11
12    /* loop 10 times and calculate and output square of x each time */
13    for ( x = 1; x <= 10; x++ ) {
14       printf( "%d ", square( x ) ); /* function call */
15    } /* end for */
16
17    printf( "\n" );
18
19    return 0; /* indicates successful termination */
20
21 } /* end main */
22
```

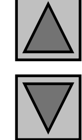


fig05_03.c (Part 1 of 2)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
23 /* square function definition returns square of an integer */
24 int square( int y ) /* y is a copy of argument to function */
25 {
26    return y * y; /* returns square of y as an int */
27
28 } /* end function square */
```

```
1   4   9   16   25   36   49   64   81   100
```


fig05_03.c (Part 2 of 2)

Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
1  /* Fig. 5.4: fig05_04.c
2     Finding the maximum of three integers */
3  #include <stdio.h>
4
5  int maximum( int x, int y, int z ); /* function prototype */
6
7  /* function main begins program execution */
8  int main()
9  {
10    int number1; /* first integer */
11    int number2; /* second integer */
12    int number3; /* third integer */
13
14    printf( "Enter three integers: " );
15    scanf( "%d%d%d", &number1, &number2, &number3 );
16
17    /* number1, number2 and number3 are arguments
18       to the maximum function call */
19    printf( "Maximum is: %d\n", maximum( number1, number2, number3 ) );
20
21    return 0; /* indicates successful termination */
22
23 } /* end main */
24
```

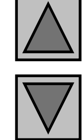


fig05_04.c (Part 1 of 2)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
25 /* Function maximum definition */
26 /* x, y and z are parameters */
27 int maximum( int x, int y, int z )
28 {
29    int max = x;      /* assume x is largest */
30
31    if ( y > max ) { /* if y is larger than max, assign y to max */
32       max = y;
33    } /* end if */
34
35    if ( z > max ) { /* if z is larger than max, assign z to max */
36       max = z;
37    } /* end if */
38
39    return max;       /* max is largest value */
40
41 } /* end function maximum */
```

```
Enter three integers: 22 85 17
Maximum is: 85
Enter three integers: 85 22 17
Maximum is: 85
Enter three integers: 22 17 85
Maximum is: 85
```

Outline

fig05_04.c (Part 2 of 2)

Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

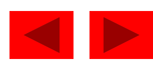
5.6 Function Prototypes

- Function prototype
  - Function name
  - Parameters – what the function takes in
  - Return type – data type function returns (default `int`)
  - Used to validate functions
  - Prototype only needed if function definition comes after use in program
  - The function with the prototype

```
int maximum( int x, int y, int z );
```

    - Takes in 3 `int`s
    - Returns an `int`
- Promotion rules and conversions
  - Converting to lower types can lead to errors

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

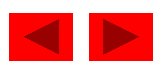


5.6 Function Prototypes

Data types	printf conversion specifications	scanf conversion specifications
long double	%Lf	%Lf
double	%f	%lf
float	%f	%f
unsigned long int	%lu	%lu
long int	%ld	%ld
unsigned int	%u	%u
int	%d	%d
short	%hd	%hd
char	%c	%c

Fig. 5.5 Promotion hierarchy for data types.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



5.7 Header Files

- Header files
  - Contain function prototypes for library functions
  - `<stdlib.h>` , `<math.h>` , etc
  - Load with `#include <filename>`

    `#include <math.h>`
- Custom header files
  - Create file with functions
  - Save as `filename.h`
  - Load in other files with `#include "filename.h"`
  - Reuse functions

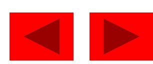
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

5.7 Header Files

Standard library header	Explanation
<assert.h>	Contains macros and information for adding diagnostics that aid program debugging.
<ctype.h>	Contains function prototypes for functions that test characters for certain properties, and function prototypes for functions that can be used to convert lowercase letters to uppercase letters and vice versa.
<errno.h>	Defines macros that are useful for reporting error conditions.
<float.h>	Contains the floating point size limits of the system.
<limits.h>	Contains the integral size limits of the system.
<locale.h>	Contains function prototypes and other information that enables a program to be modified for the current locale on which it is running. The notion of locale enables the computer system to handle different conventions for expressing data like dates, times, dollar amounts and large numbers throughout the world.
<math.h>	Contains function prototypes for math library functions.
<setjmp.h>	Contains function prototypes for functions that allow bypassing of the usual function call and return sequence.
<signal.h>	Contains function prototypes and macros to handle various conditions that may arise during program execution.
<stdarg.h>	Defines macros for dealing with a list of arguments to a function whose number and types are unknown.
<stddef.h>	Contains common definitions of types used by C for performing certain calculations.
<stdio.h>	Contains function prototypes for the standard input/output library functions, and information used by them.
<stdlib.h>	Contains function prototypes for conversions of numbers to text and text to numbers, memory allocation, random numbers, and other utility functions.
<string.h>	Contains function prototypes for string processing functions.
<time.h>	Contains function prototypes and types for manipulating the time and date.

Fig. 5.6 Some of the standard library header.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



5.8 Calling Functions: Call by Value and Call by Reference

- Call by value
  - Copy of argument passed to function
  - Changes in function do not effect original
  - Use when function does not need to modify argument
    - Avoids accidental changes
- Call by reference
  - Passes original argument
  - Changes in function effect original
  - Only used with trusted functions
- For now, we focus on call by value

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

5.9 Random Number Generation

- `rand` function
  - Load `<stdlib.h>`
  - Returns "random" number between `0` and `RAND_MAX` (at least `32767`)

```
i = rand();
```

  - Pseudorandom
    - Preset sequence of "random" numbers
    - Same sequence for every function call
- Scaling
  - To get a random number between `1` and `n`

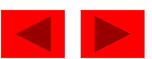
```
1 + ( rand() % n )
```

    - `rand() % n` returns a number between `0` and `n - 1`
    - Add `1` to make random number between `1` and `n`

```
1 + ( rand() % 6)
```

      - number between `1` and `6`

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



5.9 Random Number Generation

- `srand` function
  - `<stdlib.h>`
  - Takes an integer seed and jumps to that location in its "random" sequence

    `srand(` *seed* **`);`**
  - `srand( time( NULL ) );/*load <time.h> */`
    - `time( NULL )`
      - Returns the time at which the program was compiled in seconds
      - “Randomizes" the seed

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

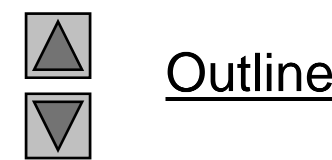


fig05_07.c

```
1  /* Fig. 5.7: fig05_07.c
2     Shifted, scaled integers produced by 1 + rand() % 6 */
3  #include <stdio.h>
4  #include <stdlib.h>
5
6  /* function main begins program execution */
7  int main()
8  {
9     int i; /* counter */
10
11    /* loop 20 times */
12    for ( i = 1; i <= 20; i++ ) {
13
14       /* pick random number from 1 to 6 and output it */
15       printf( "%10d", 1 + ( rand() % 6 ) );
16
17       /* if counter is divisible by 5, begin new line of output */
18       if ( i % 5 == 0 ) {
19          printf( "\n" );
20       } /* end if */
21
22    } /* end for */
23
24    return 0; /* indicates successful termination */
25
26 } /* end main */
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
6          6          5          5          6
5          1          1          5          3
6          6          2          4          2
6          2          3          4          1
```

Outline

Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
1  /* Fig. 5.8: fig05_08.c
2     Roll a six-sided die 6000 times */
3  #include <stdio.h>
4  #include <stdlib.h>
5
6  /* function main begins program execution */
7  int main()
8  {
9     int frequency1 = 0; /* rolled 1 counter */
10    int frequency2 = 0; /* rolled 2 counter */
11    int frequency3 = 0; /* rolled 3 counter */
12    int frequency4 = 0; /* rolled 4 counter */
13    int frequency5 = 0; /* rolled 5 counter */
14    int frequency6 = 0; /* rolled 6 counter */
15
16    int roll; /* roll counter */
17    int face; /* represents one roll of the die, value 1 to 6 */
18
19    /* loop 6000 times and summarize results */
20    for ( roll = 1; roll <= 6000; roll++ ) {
21       face = 1 + rand() % 6; /* random number from 1 to 6 */
22
```

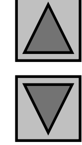


fig05_08.c (Part 1 of 3)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
23     /* determine face value and increment appropriate counter */
24     switch ( face ) {
25
26        case 1:          /* rolled 1 */
27           ++frequency1;
28           break;
29
30        case 2:          /* rolled 2 */
31           ++frequency2;
32           break;
33
34        case 3:          /* rolled 3 */
35           ++frequency3;
36           break;
37
38        case 4:          /* rolled 4 */
39           ++frequency4;
40           break;
41
42        case 5:          /* rolled 5 */
43           ++frequency5;
44           break;
45
```

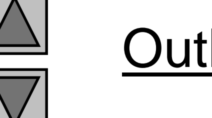
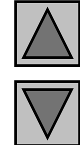


fig05_08.c (Part 2 of 3)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Outline

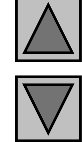
fig05_08.c (Part 3 of 3)

```
45
46          case 6:          /* rolled 6 */
47             ++frequency6;
48             break;
49       } /* end switch */
50
51    } /* end for */
52
53    /* display results in tabular format */
54    printf( "%s%13s\n", "Face", "Frequency" );
55    printf( "   1%13d\n", frequency1 );
56    printf( "   2%13d\n", frequency2 );
57    printf( "   3%13d\n", frequency3 );
58    printf( "   4%13d\n", frequency4 );
59    printf( "   5%13d\n", frequency5 );
60    printf( "   6%13d\n", frequency6 );
61
62    return 0; /* indicates successful termination */
63
64 } /* end main */
```

Program Output

```
Face    Frequency
   1         1003
   2         1017
   3          983
   4          994
   5         1004
   6          999
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Outline

fig05_09.c (Part 1 of 2)

```
1  /* Fig. 5.9: fig05_09.c
2     Randomizing die-rolling program */
3  #include <stdlib.h>
4  #include <stdio.h>
5
6  /* function main begins program execution */
7  int main()
8  {
9     int i;          /* counter */
10    unsigned seed; /* number used to seed random number generator */
11
12    printf( "Enter seed: " );
13    scanf( "%u", &seed );
14
15    srand( seed ); /* seed random number generator */
16
17    /* loop 10 times */
18    for ( i = 1; i <= 10; i++ ) {
19
20       /* pick a random number from 1 to 6 and output it */
21       printf( "%10d", 1 + ( rand() % 6 ) );
22
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
23      /* if counter is divisible by 5, begin a new line of output */
24      if ( i % 5 == 0 ) {
25         printf( "\n" );
26      } /* end if */
27
28   } /* end for */
29
30   return 0; /* indicates successful termination */
31
32 } /* end main */
```

```
Enter seed: 67
         6         1         4         6         2
         1         6         1         6         4
Enter seed: 867
         2         4         6         1         6
         1         1         3         6         2
Enter seed: 67
         6         1         4         6         2
         1         6         1         6         4
```

Outline

fig05_09.c (Part 2 of 2)

Program Output

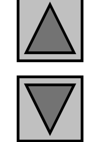
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

5.10 Example: A Game of Chance

- Craps simulator
- Rules
  - Roll two dice
    - 7 or 11 on first throw, player wins
    - 2, 3, or 12 on first throw, player loses
    - 4, 5, 6, 8, 9, 10 - value becomes player's "point"
  - Player must roll his point before rolling 7 to win

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
1  /* Fig. 5.10: fig05_10.c
2     Craps */
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include <time.h> /* contains prototype for function time */
6
7  /* enumeration constants represent game status */
8  enum Status { CONTINUE, WON, LOST };
9
10 int rollDice( void ); /* function prototype */
11
12 /* function main begins program execution */
13 int main()
14 {
15    int sum;        /* sum of rolled dice */
16    int myPoint;    /* point earned */
17
18    enum Status gameStatus; /* can contain CONTINUE, WON, or LOST */
19
20    /* randomize random number generator using current time */
21    srand( time( NULL ) );
22
23    sum = rollDice(  ); /* first roll of the dice */
24
```



Outline

fig05_10.c (Part 1 of 4)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
25    /* determine game status based on sum of dice */
26    switch( sum ) {
27
28       /* win on first roll */
29       case 7:
30       case 11:
31          gameStatus = WON;
32          break;
33
34       /* lose on first roll */
35       case 2:
36       case 3:
37       case 12:
38          gameStatus = LOST;
39          break;
40
41       /* remember point */
42       default:
43          gameStatus = CONTINUE;
44          myPoint = sum;
45          printf( "Point is %d\n", myPoint );
46          break; /* optional */
47    } /* end switch */
48
```

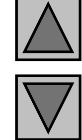


fig05_10.c (Part 2 of 4)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
49    /* while game not complete */
50    while ( gameStatus == CONTINUE ) {
51       sum = rollDice(  ); /* roll dice again */
52
53       /* determine game status */
54       if ( sum == myPoint ) { /* win by making point */
55          gameStatus = WON;
56       } /* end if */
57       else {
58
59          if ( sum == 7 ) { /* lose by rolling 7 */
60             gameStatus = LOST;
61          } /* end if */
62
63       } /* end else */
64
65    } /* end while */
66
67    /* display won or lost message */
68    if ( gameStatus == WON ) {
69       printf( "Player wins\n" );
70    } /* end if */
71    else {
72       printf( "Player loses\n" );
73    } /* end else */
74
```

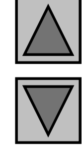


fig05_10.c (Part 3 of 4)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
75    return 0; /* indicates successful termination */
76
77 } /* end main */
78
79 /* roll dice, calculate sum and display results */
80 int rollDice( void )
81 {
82    int die1;    /* first die */
83    int die2;    /* second die */
84    int workSum; /* sum of dice */
85
86    die1 = 1 + ( rand() % 6 ); /* pick random die1 value */
87    die2 = 1 + ( rand() % 6 ); /* pick random die2 value */
88    workSum = die1 + die2;     /* sum die1 and die2 */
89
90    /* display results of this roll */
91    printf( "Player rolled %d + %d = %d\n", die1, die2, workSum );
92
93    return workSum; /* return sum of dice */
94
95 } /* end function rollRice */
```

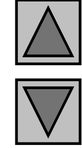


fig05_10.c (Part 4 of 4)

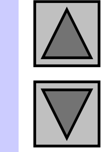
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
Player rolled 5 + 6 = 11
Player wins

Player rolled 4 + 1 = 5
Point is 5
Player rolled 6 + 2 = 8
Player rolled 2 + 1 = 3
Player rolled 3 + 2 = 5
Player wins

Player rolled 1 + 1 = 2
Player loses

Player rolled 1 + 4 = 5
Point is 5
Player rolled 3 + 4 = 7
Player loses
```



Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

5.11 Storage Classes

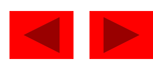
- Storage class specifiers
  - Storage duration – how long an object exists in memory
  - Scope – where object can be referenced in program
  - Linkage – specifies the files in which an identifier is known (more in Chapter 14)
- Automatic storage
  - Object created and destroyed within its block
  - `auto`: default for local variables

```
auto double x, y;
```

  - `register`: tries to put variable into high-speed registers
    - Can only be used for automatic variables

```
register int counter = 1;
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



5.11 Storage Classes

- Static storage
  - Variables exist for entire program execution
  - Default value of zero
  - `static`: local variables defined in functions.
    - Keep value after function ends
    - Only known in their own function
  - `extern`: default for global variables and functions
    - Known in any function

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

5.12 Scope Rules

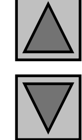
- File scope
  - Identifier defined outside function, known in all functions
  - Used for global variables, function definitions, function prototypes
- Function scope
  - Can only be referenced inside a function body
  - Used only for labels (`start:`, `case:` , etc.)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

5.12 Scope Rules

- Block scope
  - Identifier declared inside a block
    - Block scope begins at definition, ends at right brace
  - Used for variables, function parameters (local variables of function)
  - Outer blocks "hidden" from inner blocks if there is a variable with the same name in the inner block
- Function prototype scope
  - Used for identifiers in parameter list

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Outline

fig05_12.c (Part 1 of 3)

```
1  /* Fig. 5.12: fig05_12.c
2     A scoping example */
3  #include <stdio.h>
4
5  void useLocal( void );       /* function prototype */
6  void useStaticLocal( void ); /* function prototype */
7  void useGlobal( void );      /* function prototype */
8
9  int x = 1; /* global variable */
10
11 /* function main begins program execution */
12 int main()
13 {
14    int x = 5; /* local variable to main */
15
16    printf("local x in outer scope of main is %d\n", x );
17
18    { /* start new scope */
19       int x = 7; /* local variable to new scope */
20
21       printf( "local x in inner scope of main is %d\n", x );
22    } /* end new scope */
23
24    printf( "local x in outer scope of main is %d\n", x );
25
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

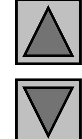
```
26   useLocal();       /* useLocal has automatic local x */
27   useStaticLocal(); /* useStaticLocal has static local x */
28   useGlobal();      /* useGlobal uses global x */
29   useLocal();       /* useLocal reinitializes automatic local x */
30   useStaticLocal(); /* static local x retains its prior value */
31   useGlobal();      /* global x also retains its value */
32
33   printf( "local x in main is %d\n", x );
34
35   return 0; /* indicates successful termination */
36
37 } /* end main */
38
39 /* useLocal reinitializes local variable x during each call */
40 void useLocal( void )
41 {
42   int x = 25;  /* initialized each time useLocal is called */
43
44   printf( "\nlocal x in a is %d after entering a\n", x );
45   x++;
46   printf( "local x in a is %d before exiting a\n", x );
47 } /* end function useLocal */
48
```

Outline

fig05_12.c (Part 2 of 3)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

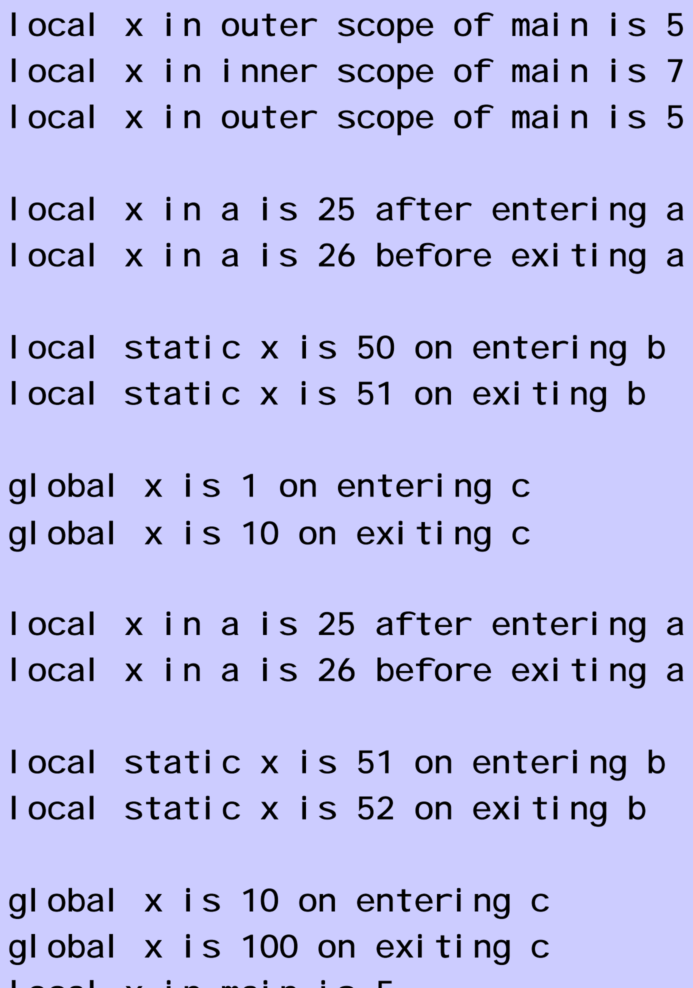
```
49 /* useStaticLocal initializes static local variable x only the first time
50    the function is called; value of x is saved between calls to this
51    function */
52 void useStaticLocal( void )
53 {
54    /* initialized only first time useStaticLocal is called */
55    static int x = 50;
56
57    printf( "\nlocal static x is %d on entering b\n", x );
58    x++;
59    printf( "local static x is %d on exiting b\n", x );
60 } /* end function useStaticLocal */
61
62 /* function useGlobal modifies global variable x during each call */
63 void useGlobal( void )
64 {
65    printf( "\nglobal x is %d on entering c\n", x );
66    x *= 10;
67    printf( "global x is %d on exiting c\n", x );
68 } /* end function useGlobal */
```



Outline

fig05_12.c (Part 3 of 3)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



```
local x in outer scope of main is 5
local x in inner scope of main is 7
local x in outer scope of main is 5

local x in a is 25 after entering a
local x in a is 26 before exiting a

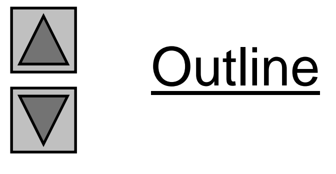
local static x is 50 on entering b
local static x is 51 on exiting b

global x is 1 on entering c
global x is 10 on exiting c

local x in a is 25 after entering a
local x in a is 26 before exiting a

local static x is 51 on entering b
local static x is 52 on exiting b

global x is 10 on entering c
global x is 100 on exiting c
local x in main is 5
```



Outline

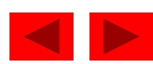
Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

5.13 Recursion

- Recursive functions
  - Functions that call themselves
  - Can only solve a base case
  - Divide a problem up into
    - What it can do
    - What it cannot do
      - What it cannot do resembles original problem
      - The function launches a new copy of itself (recursion step) to solve what it cannot do
  - Eventually base case gets solved
    - Gets plugged in, works its way up and solves whole problem

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

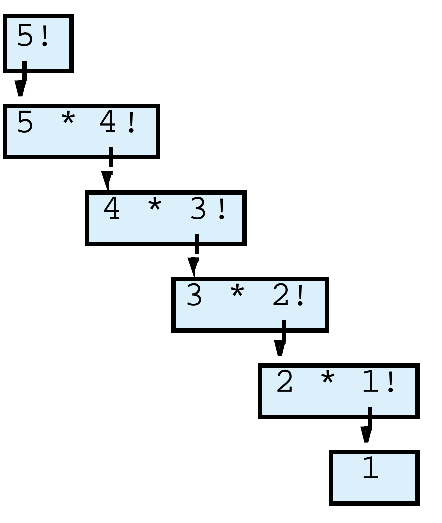


5.13 Recursion

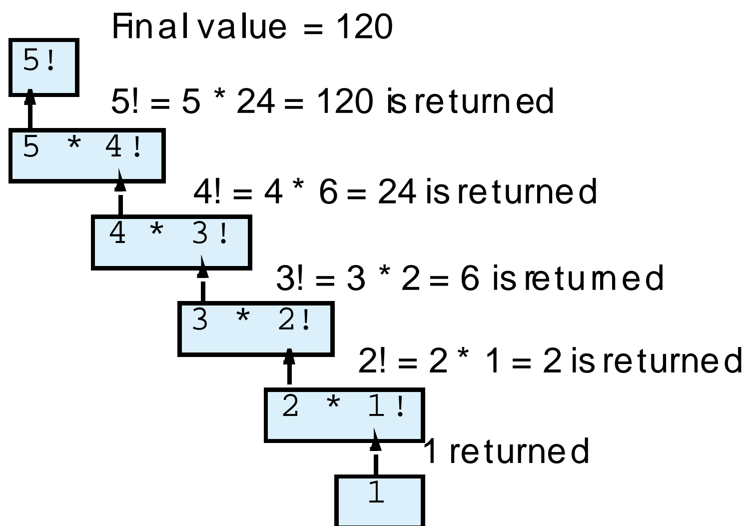
- Example: factorials
  - `5! = 5 * 4 * 3 * 2 * 1`
  - Notice that
    - `5! = 5 * 4!`
    - `4! = 4 * 3! ...`
  - Can compute factorials recursively
  - Solve base case (`1! = 0! = 1`) then plug in
    - `2! = 2 * 1! = 2 * 1 = 2;`
    - `3! = 3 * 2! = 3 * 2 = 6;`

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

5.13 Recursion

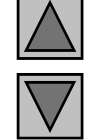


(a) Sequence of recursive calls.



(b) Values returned from each recursive call.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Outline

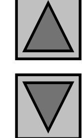
fig05_14.c (Part 1 of 2)

```
1  /* Fig. 5.14: fig05_14.c
2     Recursive factorial function */
3  #include <stdio.h>
4
5  long factorial( long number ); /* function prototype */
6
7  /* function main begins program execution */
8  int main()
9  {
10    int i; /* counter */
11
12    /* loop 10 times.  During each iteration, calculate
13       factorial( i ) and display result */
14    for ( i = 1; i <= 10; i++ ) {
15       printf( "%2d! = %ld\n", i, factorial( i ) );
16    } /* end for */
17
18    return 0; /* indicates successful termination */
19
20 } /* end main */
21
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
22 /* recursive definition of function factorial */
23 long factorial( long number )
24 {
25    /* base case */
26    if ( number <= 1 ) {
27       return 1;
28    } /* end if */
29    else { /* recursive step */
30       return ( number * factorial( number - 1 ) );
31    } /* end else */
32 
33 } /* end function factorial */
```

```
 1! = 1
 2! = 2
 3! = 6
 4! = 24
 5! = 120
 6! = 720
 7! = 5040
 8! = 40320
 9! = 362880
10! = 3628800
```



Outline

fig05_14.c (Part 2 of 2)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

5.14 Example Using Recursion: The Fibonacci Series

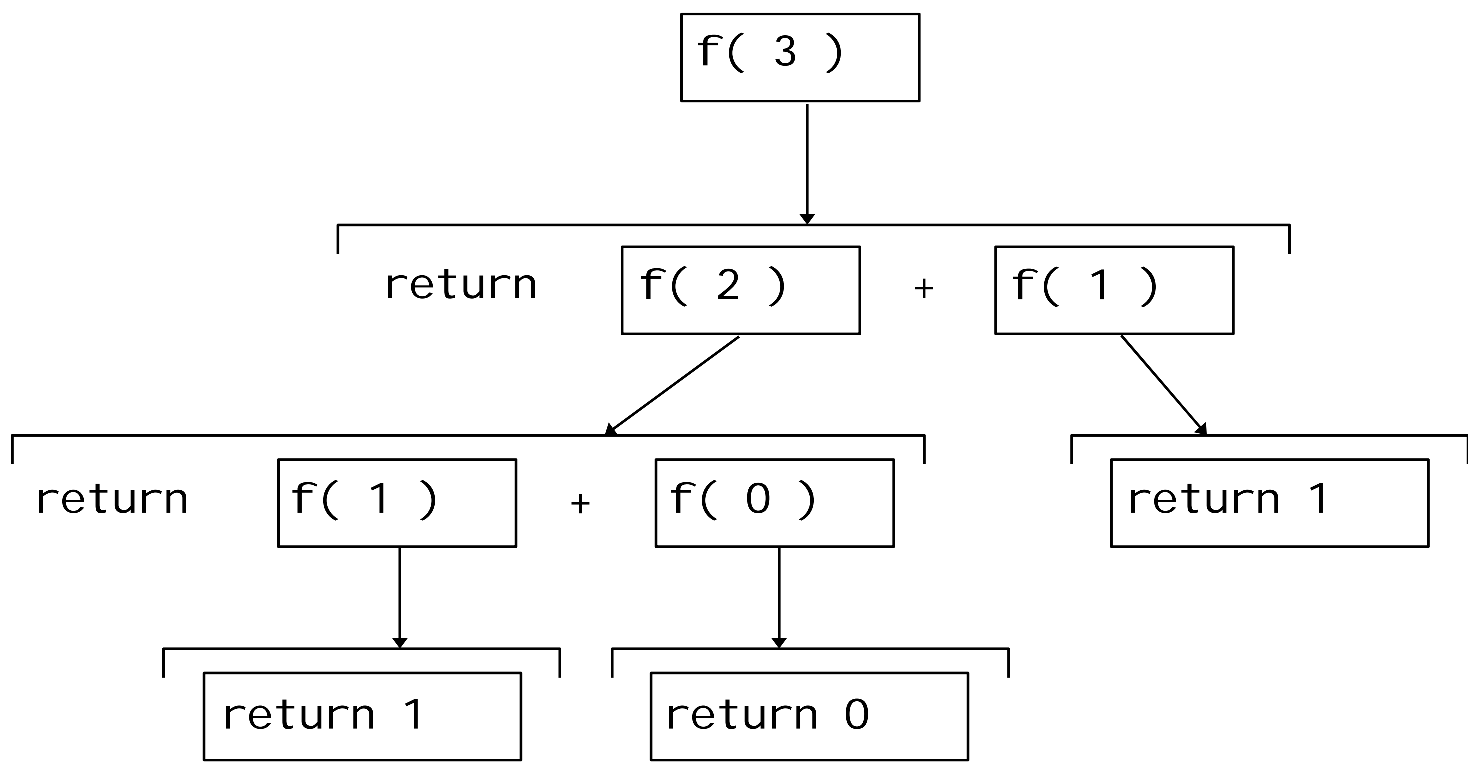
- Fibonacci series: 0, 1, 1, 2, 3, 5, 8...
  - Each number is the sum of the previous two
  - Can be solved recursively:
    - `fib( n ) = fib( n - 1 ) + fib( n – 2 )`
  - Code for the `fibonacci` function

```
long fibonacci( long n )
{
  if (n == 0 || n == 1)  // base case
     return n;
  else
     return fibonacci( n - 1) +
        fibonacci( n – 2 );
}
```

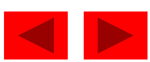
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

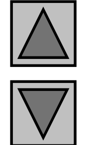
5.14 Example Using Recursion: The Fibonacci Series

- Set of recursive calls to function `fibonacci`



© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



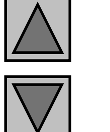


Outline

fig05_15.c (Part 1 of 2)

```
1  /* Fig. 5.15: fig05_15.c
2     Recursive fibonacci function */
3  #include <stdio.h>
4
5  long fibonacci( long n ); /* function prototype */
6
7  /* function main begins program execution */
8  int main()
9  {
10    long result; /* fibonacci value */
11    long number; /* number input by user */
12
13    /* obtain integer from user */
14    printf( "Enter an integer: " );
15    scanf( "%ld", &number );
16
17    /* calculate fibonacci value for number input by user */
18    result = fibonacci( number );
19
20    /* display result */
21    printf( "Fibonacci( %ld ) = %ld\n", number, result );
22
23    return 0; /* indicates successful termination */
24
25 } /* end main */
26
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Outline

fig05_15.c (Part 2 of 2)

```
27 /* Recursive definition of function fibonacci */
28 long fibonacci( long n )
29 {
30    /* base case */
31    if ( n == 0 || n == 1 ) {
32       return n;
33    } /* end if */
34    else { /* recursive step */
35       return fibonacci( n - 1 ) + fibonacci( n - 2 );
36    } /* end else */
37
38 } /* end function fibonacci */
```

Program Output

```
Enter an integer: 0
Fibonacci( 0 ) = 0

Enter an integer: 1
Fibonacci( 1 ) = 1

Enter an integer: 2
Fibonacci( 2 ) = 1

Enter an integer: 3
Fibonacci( 3 ) = 2

Enter an integer: 4
Fibonacci( 4 ) = 3
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
Enter an integer: 5
Fibonacci( 5 ) = 5

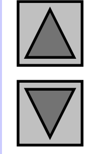
Enter an integer: 6
Fibonacci( 6 ) = 8

Enter an integer: 10
Fibonacci( 10 ) = 55

Enter an integer: 20
Fibonacci( 20 ) = 6765

Enter an integer: 30
Fibonacci( 30 ) = 832040

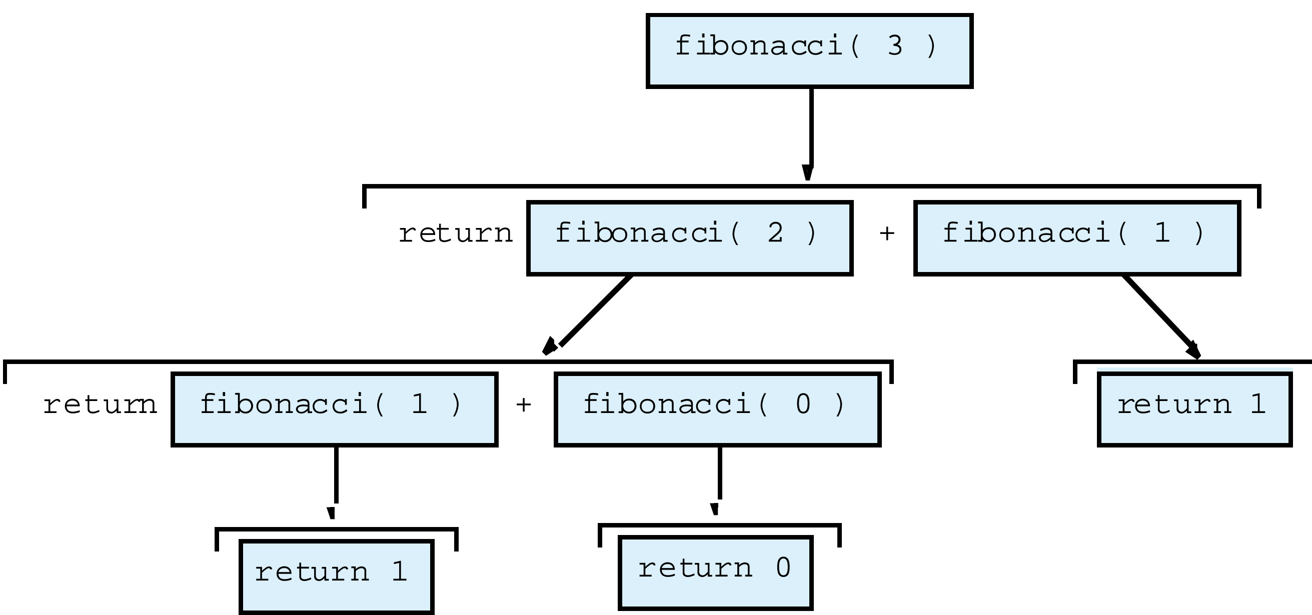
Enter an integer: 35
Fibonacci( 35 ) = 9227465
```



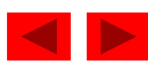
Program Output (continued)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

5.14 Example Using Recursion: The Fibonacci Series



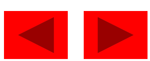
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



5.15 Recursion vs. Iteration

- Repetition
  - Iteration: explicit loop
  - Recursion: repeated function calls
- Termination
  - Iteration: loop condition fails
  - Recursion: base case recognized
- Both can have infinite loops
- Balance
  - Choice between performance (iteration) and good software engineering (recursion)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

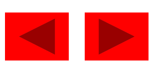


5.15 Recursion vs. Iteration

Chapter	Recursion Examples and Exercises
Chapter 5	Factorial function Fibonacci functions Greatest common divisor Sum of two integers Multiply two integers Raising an integer to an integer power Towers of Hanoi Recursive main Printing keyboard inputs in reverse Visualizing recursion
Chapter 6	Sum the elements of an array Print an array Print an array backwards Print a string backwards Check if a string is a palindrome Minimum value in an array Selection sort Quicksort Linear search Binary search
Chapter 7	Eight Queens Maze traversal
Chapter 8	Printing a string input at the keyboard backwards
Chapter 12	Linked list insert Linked list delete Search a linked list Print a linked list backwards Binary tree insert Preorder traversal of a binary tree Inorder traversal of a binary tree Postorder traversal of a binary tree

Fig. 5.17 Summary of recursion examples and exercises in the text.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

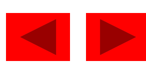


Chapter 6 - Arrays

Outline

6.1 Introduction
6.2 Arrays
6.3 Declaring Arrays
6.4 Examples Using Arrays
6.5 Passing Arrays to Functions
6.6 Sorting Arrays
6.7 Case Study: Computing Mean, Median and Mode Using Arrays
6.8 Searching Arrays
6.9 Multiple-Subscripted Arrays

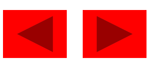
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Objectives

- In this chapter, you will learn:
  - To introduce the array data structure.
  - To understand the use of arrays to store, sort and search lists and tables of values.
  - To understand how to define an array, initialize an array and refer to individual elements of an array.
  - To be able to pass arrays to functions.
  - To understand basic sorting techniques.
  - To be able to define and manipulate multiple subscript arrays.

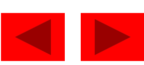
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



6.1 Introduction

- Arrays
  - Structures of related data items
  - Static entity – same size throughout program
  - Dynamic data structures discussed in Chapter 12

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



6.2 Arrays

- Array
  - Group of consecutive memory locations
  - Same name and type
- To refer to an element, specify
  - Array name
  - Position number
- Format:

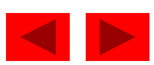
  arrayname[*position number*]
  - First element at position 0
  - n element array named c:
    - c[0], c[1]...c[n – 1]

Name of array (Note that all elements of this array have the same name, c)

c[0]	-45
c[1]	6
c[2]	0
c[3]	72
c[4]	1543
c[5]	-89
c[6]	0
c[7]	62
c[8]	-3
c[9]	1
c[10]	6453
c[11]	78

Position number of the element within array c

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



6.2 Arrays

- Array elements are like normal variables

```
c[ 0 ] =  3;
printf( "%d", c[ 0 ] );
```

  - Perform operations in subscript. If `x` equals 3

```
c[ 5 - 2 ] == c[ 3 ] == c[ x ]
```

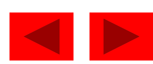
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

6.2 Arrays

Operators						Associativity	Type
[]	()					left to right	highest
++	--	!	*(type)*			right to left	unary
*	/	%				left to right	multiplicative
+	-					left to right	additive
<	<=	>	>=			left to right	relational
==	!=					left to right	equality
&&						left to right	logical and
||						left to right	logical or
?:						right to left	conditional
=	+=	-=	*=	/=	%=	right to left	assignment
,						left to right	comma

Fig. 6.2 Operator precedence.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



6.3 Defining Arrays

- When defining arrays, specify
  - Name
  - Type of array
  - Number of elements

```
arrayType arrayName[ numberOfElements ];
```

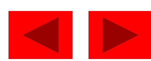
  - Examples:

```
int c[ 10 ];
float myArray[ 3284 ];
```

- Defining multiple arrays of same type
  - Format similar to regular variables
  - Example:

```
int b[ 100 ], x[ 27 ];
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



6.4 Examples Using Arrays

- Initializers

```
int n[ 5 ] = { 1, 2, 3, 4, 5 };
```

  - If not enough initializers, rightmost elements become `0`

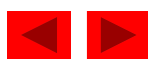
```
int n[ 5 ] = { 0 }
```

    - All elements 0
  - If too many a syntax error is produced syntax error
  - C arrays have no bounds checking
- If size omitted, initializers determine it

```
int n[ ] = { 1, 2, 3, 4, 5 };
```

  - 5 initializers, therefore 5 element array

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



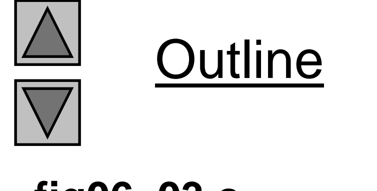
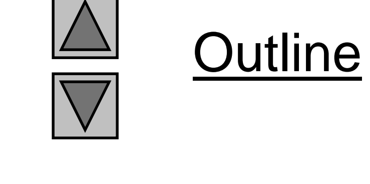


fig06_03.c

```
1  /* Fig. 6.3: fig06_03.c
2     initializing an array */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main()
7  {
8     int n[ 10 ]; /* n is an array of 10 integers */
9     int i;       /* counter */
10
11    /* initialize elements of array n to 0 */
12    for ( i = 0; i < 10; i++ ) {
13       n[ i ] = 0; /* set element at location i to 0 */
14    } /* end for */
15
16    printf( "%s%13s\n", "Element", "Value" );
17
18    /* output contents of array n in tabular format */
19    for ( i = 0; i < 10; i++ ) {
20       printf( "%7d%13d\n", i, n[ i ] );
21    } /* end for */
22
23    return 0; /* indicates successful termination */
24
25 } /* end main */
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Program Output

```
Element    Value
      0        0
      1        0
      2        0
      3        0
      4        0
      5        0
      6        0
      7        0
      8        0
      9        0
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

6.4 Examples Using Arrays

- Character arrays
  - String “first” is really a static array of characters
  - Character arrays can be initialized using string literals

```
char string1[] = "first";
```

    - Null character '\0' terminates strings
    - string1 actually has 6 elements
      - It is equivalent to

```
char string1[] = { 'f', 'i', 'r', 's', 't', '\0' };
```

  - Can access individual characters

```
string1[ 3 ] is character 's'
```

  - Array name is address of array, so & not needed for scanf

```
scanf( "%s", string2 );
```

    - Reads characters until whitespace encountered
    - Can write beyond end of array, be careful

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

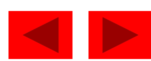


fig06_04.c

```
1  /* Fig. 6.4: fig06_04.c
2     Initializing an array with an initializer list */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main()
7  {
8     /* use initializer list to initialize array n */
9     int n[ 10 ] = { 32, 27, 64, 18, 95, 14, 90, 70, 60, 37 };
10    int i; /* counter */
11
12    printf( "%s%13s\n", "Element", "Value" );
13
14    /* output contents of array in tabular format */
15    for ( i = 0; i < 10; i++ ) {
16       printf( "%7d%13d\n", i, n[ i ] );
17    } /* end for */
18
19    return 0; /* indicates successful termination */
20
21 } /* end main */
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Program Output

```
Element        Value
      0           32
      1           27
      2           64
      3           18
      4           95
      5           14
      6           90
      7           70
      8           60
      9           37
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

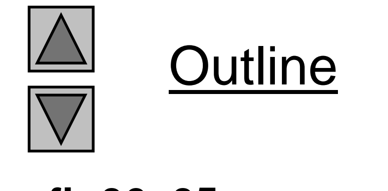


fig06_05.c

```
1  /* Fig. 6.5: fig06_05.c
2     Initialize the elements of array s to the even integers from 2 to 20 */
3  #include <stdio.h>
4  #define SIZE 10
5
6  /* function main begins program execution */
7  int main()
8  {
9     /* symbolic constant SIZE can be used to specify array size */
10    int s[ SIZE ]; /* array s has 10 elements */
11    int j;         /* counter */
12
13    for ( j = 0; j < SIZE; j++ ) { /* set the values */
14       s[ j ] = 2 + 2 * j;
15    } /* end for */
16
17    printf( "%s%13s\n", "Element", "Value" );
18
19    /* output contents of array s in tabular format */
20    for ( j = 0; j < SIZE; j++ ) {
21       printf( "%7d%13d\n", j, s[ j ] );
22    } /* end for */
23
24    return 0; /* indicates successful termination */
25
26 } /* end main */
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
Element         Value
      0             2
      1             4
      2             6
      3             8
      4            10
      5            12
      6            14
      7            16
      8            18
      9            20
```


Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

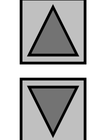
fig06_06.c

```
1  /* Fig. 6.6: fig06_06.c
2     Compute the sum of the elements of the array */
3  #include <stdio.h>
4  #define SIZE 12
5  
6  /* function main begins program execution */
7  int main()
8  {
9     /* use initializer list to initialize array */
10    int a[ SIZE ] = { 1, 3, 5, 4, 7, 2, 99, 16, 45, 67, 89, 45 };
11    int i;        /* counter */
12    int total = 0; /* sum of array */
13 
14    /* sum contents of array a */
15    for ( i = 0; i < SIZE; i++ ) {
16       total += a[ i ];
17    } /* end for */
18 
19    printf( "Total of array element values is %d\n", total );
20 
21    return 0; /* indicates successful termination */
22 
23 } /* end main */
```

Program Output

```
Total of array element values is 383
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Outline

fig06_07.c (Part 1 of 2)

```
1  /* Fig. 6.7: fig06_07.c
2     Student poll program */
3  #include <stdio.h>
4  #define RESPONSE_SIZE 40 /* define array sizes */
5  #define FREQUENCY_SIZE 11
6
7  /* function main begins program execution */
8  int main()
9  {
10    int answer; /* counter */
11    int rating; /* counter */
12
13    /* initialize frequency counters to 0 */
14    int frequency[ FREQUENCY_SIZE ] = { 0 };
15
16    /* place survey responses in array responses */
17    int responses[ RESPONSE_SIZE ] = { 1, 2, 6, 4, 8, 5, 9, 7, 8, 10,
18         1, 6, 3, 8, 6, 10, 3, 8, 2, 7, 6, 5, 7, 6, 8, 6, 7, 5, 6, 6,
19         5, 6, 7, 5, 6, 4, 8, 6, 8, 10 };
20
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

fig06_07.c (Part 2 of 2)

```
21    /* for each answer, select value of an element of array responses
22       and use that value as subscript in array frequency to
23       determine element to increment */
24    for ( answer = 0; answer < RESPONSE_SIZE; answer++ ) {
25       ++frequency[ responses [ answer ] ];
26    } /* end for */
27
28    /* display results */
29    printf( "%s%17s\n", "Rating", "Frequency" );
30
31    /* output frequencies in tabular format */
32    for ( rating = 1; rating < FREQUENCY_SIZE; rating++ ) {
33       printf( "%6d%17d\n", rating, frequency[ rating ] );
34    } /* end for */
35
36    return 0; /* indicates successful termination */
37
38 } /* end main */
```

Program Output

```
Rating        Frequency
     1                2
     2                2
     3                2
     4                2
     5                5
     6               11
     7                5
     8                7
     9                1
    10                3
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Outline

fig06_08.c (Part 1 of 2)

```
1  /* Fig. 6.8: fig06_08.c
2     Histogram printing program */
3  #include <stdio.h>
4  #define SIZE 10
5
6  /* function main begins program execution */
7  int main()
8  {
9     /* use initializer list to initialize array n */
10    int n[ SIZE ] = { 19, 3, 15, 7, 11, 9, 13, 5, 17, 1 };
11    int i; /* outer counter */
12    int j; /* inner counter */
13
14    printf( "%s%13s%17s\n", "Element", "Value", "Histogram" );
15
16    /* for each element of array n, output a bar in histogram */
17    for ( i = 0; i < SIZE; i++ ) {
18       printf( "%7d%13d        ", i, n[ i ]) ;
19
20       for ( j = 1; j <= n[ i ]; j++ ) { /* print one bar */
21          printf( "%c", '*' );
22         } /* end inner for */
23
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

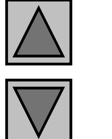
```
24       printf( "\n" ); /* start next line of output */
25    } /* end outer for */
26
27    return 0; /* indicates successful termination */
28
29 } /* end main */
```

```
Element        Value          Histogram
      0           19          *******************
      1            3          ***
      2           15          ***************
      3            7          *******
      4           11          ***********
      5            9          *********
      6           13          *************
      7            5          *****
      8           17          *****************
      9            1          *
```


fig06_08.c (Part 2 of 2)

Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Outline

fig06_09.c (Part 1 of 2)

```
1  /* Fig. 6.9: fig06_09.c
2     Roll a six-sided die 6000 times */
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include <time.h>
6  #define SIZE 7
7
8  /* function main begins program execution */
9  int main()
10 {
11    int face;                     /* random number with value 1 - 6 */
12    int roll;                     /* roll counter */
13    int frequency[ SIZE ] = { 0 }; /* initialize array to 0 */
14
15    srand( time( NULL ) ); /* seed random-number generator */
16
17    /* roll die 6000 times */
18    for ( roll = 1; roll <= 6000; roll++ ) {
19       face = rand() % 6 + 1;
20       ++frequency[ face ]; /* replaces 26-line switch of Fig. 5.8 */
21    } /* end for */
22
23    printf( "%s%17s\n", "Face", "Frequency" );
24
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Outline

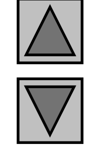
fig06_09.c (Part 2 of 2)

```
25    /* output frequency elements 1-6 in tabular format */
26    for ( face = 1; face < SIZE; face++ ) {
27       printf( "%4d%17d\n", face, frequency[ face ] );
28    } /* end for */
29
30    return 0; /* indicates successful termination */
31
32 } /* end main */
```

Program Output

```
Face      Frequency
   1           1029
   2            951
   3            987
   4           1033
   5           1010
   6            990
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

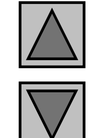


Outline

fig06_10.c (Part 1 of 2)

```
1  /* Fig. 6.10: fig06_10.c
2     Treating character arrays as strings */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main()
7  {
8     char string1[ 20 ];                /* reserves 20 characters */
9     char string2[] = "string literal"; /* reserves 15 characters */
10    int i;                             /* counter */
11
12    /* read string from user into array string2 */
13    printf("Enter a string: ");
14    scanf( "%s", string1 );
15
16    /* output strings */
17    printf( "string1 is: %s\nstring2 is: %s\n"
18            "string1 with spaces between characters is:\n",
19            string1, string2 );
20
21    /* output characters until null character is reached */
22    for ( i = 0; string1[ i ] != '\0'; i++ ) {
23       printf( "%c ", string1[ i ] );
24    } /* end for */
25
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

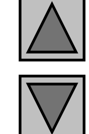



```
26    printf( "\n" );
27
28    return 0; /* indicates successful termination */
29
30 } /* end main */
```

```
Enter a string: Hello there
string1 is: Hello
string2 is: string literal
string1 with spaces between characters is:
H e l l o
```

fig06_10.c (Part 2 of 2)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Outline

fig06_11.c (Part 1 of 3)

```
1  /* Fig. 6.11: fig06_11.c
2     Static arrays are initialized to zero */
3  #include <stdio.h>
4
5  void staticArrayInit( void );    /* function prototype */
6  void automaticArrayInit( void ); /* function prototype */
7
8  /* function main begins program execution */
9  int main()
10 {
11    printf( "First call to each function:\n" );
12    staticArrayInit();
13    automaticArrayInit();
14
15    printf( "\n\nSecond call to each function:\n" );
16    staticArrayInit();
17    automaticArrayInit();
18
19    return 0; /* indicates successful termination */
20
21 } /* end main */
22
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
23 /* function to demonstrate a static local array */
24 void staticArrayInit( void )
25 {
26    /* initializes elements to 0 first time function is called */
27    static int array1[ 3 ];
28    int i; /* counter */
29
30    printf( "\nValues on entering staticArrayInit:\n" );
31
32    /* output contents of array1 */
33    for ( i = 0; i <= 2; i++ ) {
34       printf( "array1[ %d ] = %d  ", i, array1[ i ] );
35    } /* end for */
36
37    printf( "\nValues on exiting staticArrayInit:\n" );
38
39    /* modify and output contents of array1 */
40    for ( i = 0; i <= 2; i++ ) {
41       printf( "array1[ %d ] = %d  ", i, array1[ i ] += 5 );
42    } /* end for */
43
44 } /* end function staticArrayInit */
45
```


fig06_11.c (Part 2 of 3)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
46 /* function to demonstrate an automatic local array */
47 void automaticArrayInit( void )
48 {
49    /* initializes elements each time function is called */
50    int array2[ 3 ] = { 1, 2, 3 };
51    int i; /* counter */
52
53    printf( "\n\nValues on entering automaticArrayInit:\n" );
54
55    /* output contents of array2 */
56    for ( i = 0; i <= 2; i++ ) {
57       printf("array2[ %d ] = %d  ", i, array2[ i ] );
58    } /* end for */
59
60    printf( "\nValues on exiting automaticArrayInit:\n" );
61
62    /* modify and output contents of array2 */
63    for ( i = 0; i <= 2; i++ ) {
64       printf( "array2[ %d ] = %d  ", i, array2[ i ] += 5 );
65    } /* end for */
66
67 } /* end function automaticArrayInit */
```


fig06_11.c (Part 3 of 3)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
First call to each function:

Values on entering staticArrayInit:
array1[ 0 ] = 0  array1[ 1 ] = 0  array1[ 2 ] = 0
Values on exiting staticArrayInit:
array1[ 0 ] = 5  array1[ 1 ] = 5  array1[ 2 ] = 5

Values on entering automaticArrayInit:
array2[ 0 ] = 1  array2[ 1 ] = 2  array2[ 2 ] = 3
Values on exiting automaticArrayInit:
array2[ 0 ] = 6  array2[ 1 ] = 7  array2[ 2 ] = 8

Second call to each function:

Values on entering staticArrayInit:
array1[ 0 ] = 5  array1[ 1 ] = 5  array1[ 2 ] = 5
Values on exiting staticArrayInit:
array1[ 0 ] = 10  array1[ 1 ] = 10  array1[ 2 ] = 10

Values on entering automaticArrayInit:
array2[ 0 ] = 1  array2[ 1 ] = 2  array2[ 2 ] = 3
Values on exiting automaticArrayInit:
array2[ 0 ] = 6  array2[ 1 ] = 7  array2[ 2 ] = 8
```


Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

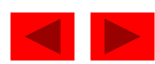
6.5 Passing Arrays to Functions

- Passing arrays
  - To pass an array argument to a function, specify the name of the array without any brackets

```
int myArray[ 24 ];
myFunction( myArray, 24 );
```

    - Array size usually passed to function
  - Arrays passed call-by-reference
  - Name of array is address of first element
  - Function knows where the array is stored
    - Modifies original memory locations
- Passing array elements
  - Passed by call-by-value
  - Pass subscripted name (i.e., `myArray[ 3 ]`) to function

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



6.5 Passing Arrays to Functions

- Function prototype

```
void modifyArray( int b[], int arraySize );
```

  - Parameter names optional in prototype
    - `int b[]` could be written `int []`
    - `int arraySize` could be simply `int`

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

fig06_12.c

```
1  /* Fig. 6.12: fig06_12.c
2     The name of an array is the same as &array[ 0 ] */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main()
7  {
8     char array[ 5 ]; /* define an array of size 5 */
9
10     printf( "    array = %p\n&array[0] = %p\n"
11        "   &array = %p\n",
12        array, &array[ 0 ], &array );
13
14     return 0; /* indicates successful termination */
15
16  } /* end main */
```

Program Output

```
    array = 0012FF78
&array[0] = 0012FF78
   &array = 0012FF78
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

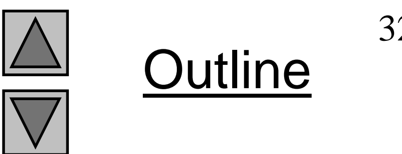


fig06_13.c (Part 1 of 3)

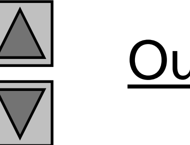
```
1  /* Fig. 6.13: fig06_13.c
2     Passing arrays and individual array elements to functions */
3  #include <stdio.h>
4  #define SIZE 5
5
6  /* function prototypes */
7  void modifyArray( int b[], int size );
8  void modifyElement( int e );
9
10 /* function main begins program execution */
11 int main()
12 {
13    int a[ SIZE ] = { 0, 1, 2, 3, 4 }; /* initialize a */
14    int i; /* counter */
15
16    printf( "Effects of passing entire array by reference:\n\nThe "
17           "values of the original array are:\n" );
18
19    /* output original array */
20    for ( i = 0; i < SIZE; i++ ) {
21       printf( "%3d", a[ i ] );
22    } /* end for */
23
24    printf( "\n" );
25
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
26     /* pass array a to modifyArray by reference */
27     modifyArray( a, SIZE );
28
29     printf( "The values of the modified array are:\n" );
30
31     /* output modified array */
32     for ( i = 0; i < SIZE; i++ ) {
33        printf( "%3d", a[ i ] );
34     } /* end for */
35
36     /* output value of a[ 3 ] */
37     printf( "\n\n\nEffects of passing array element "
38             "by value:\n\nThe value of a[3] is %d\n", a[ 3 ] );
39
40     modifyElement( a[ 3 ] ); /* pass array element a[ 3 ] by value */
41
42     /* output value of a[ 3 ] */
43     printf( "The value of a[ 3 ] is %d\n", a[ 3 ] );
44
45     return 0; /* indicates successful termination */
46
47  } /* end main */
48
```


fig06_13.c (Part 2 of 3)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Outline

fig06_13.c (Part 3 of 3)

```
49 /* in function modifyArray, "b" points to the original array "a"
50    in memory */
51 void modifyArray( int b[], int size )
52 {
53    int j; /* counter */
54
55    /* multiply each array element by 2 */
56    for ( j = 0; j < size; j++ ) {
57       b[ j ] *= 2;
58    } /* end for */
59
60 } /* end function modifyArray */
61
62 /* in function modifyElement, "e" is a local copy of array element
63    a[ 3 ] passed from main */
64 void modifyElement( int e )
65 {
66    /* multiply parameter by 2 */
67    printf( "Value in modifyElement is %d\n", e *= 2 );
68 } /* end function modifyElement */
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
Effects of passing entire array by reference:

The values of the original array are:
  0  1  2  3  4
The values of the modified array are:
  0  2  4  6  8


Effects of passing array element by value:

The value of a[3] is 6
Value in modifyElement is 12
The value of a[ 3 ] is 6
```


Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

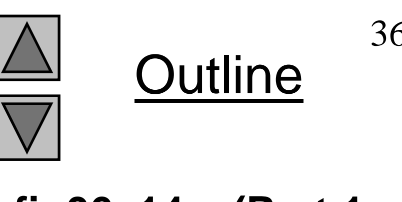
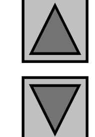


fig06_14.c (Part 1 of 2)

```
1  /* Fig. 6.14: fig06_14.c
2     Demonstrating the const type qualifier with arrays */
3  #include <stdio.h>
4
5  void tryToModifyArray( const int b[] ); /* function prototype */
6
7  /* function main begins program execution */
8  int main()
9  {
10    int a[] = { 10, 20, 30 }; /* initialize a */
11
12    tryToModifyArray( a );
13
14    printf("%d %d %d\n", a[ 0 ], a[ 1 ], a[ 2 ] );
15
16    return 0; /* indicates successful termination */
17
18 } /* end main */
19
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Outline

fig06_14.c (Part 2 of 2)

```
20 /* in function tryToModifyArray, array b is const, so it cannot be
21    used to modify the original array a in main. */
22 void tryToModifyArray( const int b[] )
23 {
24    b[ 0 ] /= 2;    /* error */
25    b[ 1 ] /= 2;    /* error */
26    b[ 2 ] /= 2;    /* error */
27 } /* end function tryToModifyArray */
```

Program Output

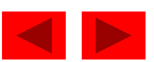
```
Compiling...
FIG06_14.C
fig06_14.c(24) : error C2166: l-value specifies const object
fig06_14.c(25) : error C2166: l-value specifies const object
fig06_14.c(26) : error C2166: l-value specifies const object
```

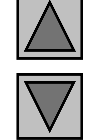
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

6.6 Sorting Arrays

- Sorting data
  - Important computing application
  - Virtually every organization must sort some data
- Bubble sort (sinking sort)
  - Several passes through the array
  - Successive pairs of elements are compared
    - If increasing order (or identical), no change
    - If decreasing order, elements exchanged
  - Repeat
- Example:
  - original: 3 4 2 6 7
  - pass 1: 3 2 4 6 7
  - pass 2: 2 3 4 6 7
  - Small elements "bubble" to the top

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.





Outline

fig06_15.c (Part 1 of 3)

```
1  /* Fig. 6.15: fig06_15.c
2     This program sorts an array's values into ascending order */
3  #include <stdio.h>
4  #define SIZE 10
5
6  /* function main begins program execution */
7  int main()
8  {
9     /* initialize a */
10    int a[ SIZE ] = { 2, 6, 4, 8, 10, 12, 89, 68, 45, 37 };
11    int i;    /* inner counter */
12    int pass; /* outer counter */
13    int hold; /* temporary location used to swap array elements */
14
15    printf( "Data items in original order\n" );
16
17    /* output original array */
18    for ( i = 0; i < SIZE; i++ ) {
19       printf( "%4d", a[ i ] );
20    } /* end for */
21
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
22    /* bubble sort */
23    /* loop to control number of passes */
24    for ( pass = 1; pass < SIZE; pass++ ) {
25
26       /* loop to control number of comparisons per pass */
27       for ( i = 0; i < SIZE - 1; i++ ) {
28
29          /* compare adjacent elements and swap them if first
30          element is greater than second element */
31          if ( a[ i ] > a[ i + 1 ] ) {
32             hold = a[ i ];
33             a[ i ] = a[ i + 1 ];
34             a[ i + 1 ] = hold;
35          } /* end if */
36
37       } /* end inner for */
38
39    } /* end outer for */
40
41    printf( "\nData items in ascending order\n" );
42
```


fig06_15.c (Part 2 of 3)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
43    /* output sorted array */
44    for ( i = 0; i < SIZE; i++ ) {
45       printf( "%4d", a[ i ] );
46    } /* end for */
47
48    printf( "\n" );
49
50    return 0; /* indicates successful termination */
51
```

```
Data items in original order
   2   6   4   8  10  12  89  68  45  37
Data items in ascending order
   2   4   6   8  10  12  37  45  68  89
```

fig06_15.c (Part 3 of 3)

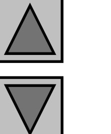
Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

6.7 Case Study: Computing Mean, Median and Mode Using Arrays

- Mean – average
- Median – number in middle of sorted list
  - 1, 2, 3, 4, 5
  - 3 is the median
- Mode – number that occurs most often
  - 1, 1, 1, 2, 3, 3, 4, 5
  - 1 is the mode

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Outline

fig06_16.c (Part 1 of 8)

```
1  /* Fig. 6.16: fig06_16.c
2     This program introduces the topic of survey data analysis.
3     It computes the mean, median, and  mode of the data */
4  #include <stdio.h>
5  #define SIZE 99
6
7  /* function prototypes */
8  void mean( const int answer[] );
9  void median( int answer[] );
10 void mode( int freq[], const int answer[] ) ;
11 void bubbleSort( int a[] );
12 void printArray( const int a[] );
13
14 /* function main begins program execution */
15 int main()
16 {
17    int frequency[ 10 ] = { 0 }; /* initialize array frequency */
18
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
19    /* initialize array response */
20    int response[ SIZE ] =
21       { 6, 7, 8, 9, 8, 7, 8, 9, 8, 9,
22         7, 8, 9, 5, 9, 8, 7, 8, 7, 8,
23         6, 7, 8, 9, 3, 9, 8, 7, 8, 7,
24         7, 8, 9, 8, 9, 8, 9, 7, 8, 9,
25         6, 7, 8, 7, 8, 7, 9, 8, 9, 2,
26         7, 8, 9, 8, 9, 8, 9, 7, 5, 3,
27         5, 6, 7, 2, 5, 3, 9, 4, 6, 4,
28         7, 8, 9, 6, 8, 7, 8, 9, 7, 8,
29         7, 4, 4, 2, 5, 3, 8, 7, 5, 6,
30         4, 5, 6, 1, 6, 5, 7, 8, 7 };
31
32    /* process responses */
33    mean( response );
34    median( response );
35    mode( frequency, response );
36
37    return 0; /* indicates successful termination */
38
39 } /* end main */
40
```

fig06_16.c (Part 2 of 8)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
41 /* calculate average of all response values */
42 void mean( const int answer[] )
43 {
44    int j;         /* counter */
45    int total = 0; /* variable to hold sum of array elements */
46
47    printf( "%s\n%s\n%s\n", "********", "  Mean", "********" );
48
49    /* total response values */
50    for ( j = 0; j < SIZE; j++ ) {
51       total += answer[ j ];
52    } /* end for */
53
54    printf( "The mean is the average value of the data\n"
55            "items. The mean is equal to the total of\n"
56            "all the data items divided by the number\n"
57            "of data items ( %d ). The mean value for\n"
58            "this run is: %d / %d = %.4f\n\n",
59            SIZE, total, SIZE, ( double ) total / SIZE );
60 } /* end function mean */
61
```

fig06_16.c (Part 3 of 8)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

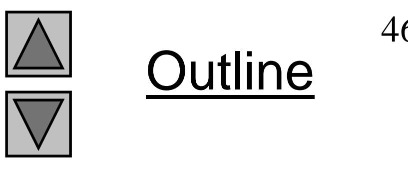


fig06_16.c (Part 4 of 8)

```
62 /* sort array and determine median element's value */
63 void median( int answer[] )
64 {
65    printf( "\n%s\n%s\n%s\n%s",
66            "********", " Median", "********",
67            "The unsorted array of responses is" );
68
69    printArray( answer ); /* output unsorted array */
70
71    bubbleSort( answer ); /* sort array */
72
73    printf( "\n\nThe sorted array is" );
74    printArray( answer ); /* output sorted array */
75
76    /* display median element */
77    printf( "\n\nThe median is element %d of\n"
78            "the sorted %d element array.\n"
79            "For this run the median is %d\n\n",
80            SIZE / 2, SIZE, answer[ SIZE / 2 ] );
81 } /* end function median */
82
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

fig06_16.c (Part 5 of 8)

```
83 /* determine most frequent response */
84 void mode( int freq[], const int answer[] )
85 {
86    int rating;        /* counter */
87    int j;             /* counter */
88    int h;             /* counter */
89    int largest = 0;   /* represents largest frequency */
90    int modeValue = 0; /* respesents most frequent response */
91
92    printf( "\n%s\n%s\n%s\n",
93            "********", "  Mode", "********" );
94
95    /* initialize frequencies to 0 */
96    for ( rating = 1; rating <= 9; rating++ ) {
97       freq[ rating ] = 0;
98    } /* end for */
99
100    /* summarize frequencies */
101   for ( j = 0; j < SIZE; j++ ) {
102      ++freq[ answer[ j ] ];
103   } /* end for */
104
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
105 /* output headers for result columns */
106 printf( "%s%11s%19s\n\n%54s\n%54s\n\n",
107         "Response", "Frequency", "Histogram",
108         "1    1    2    2", "5    0    5    0    5" );
109
110 /* output results */
111 for ( rating = 1; rating <= 9; rating++ ) {
112    printf( "%8d%11d          ", rating, freq[ rating ] );
113
114    /* keep track of mode value and largest frequency value */
115    if ( freq[ rating ] > largest ) {
116       largest = freq[ rating ];
117       modeValue = rating;
118    } /* end if */
119
120    /* output histogram bar representing frequency value */
121    for ( h = 1; h <= freq[ rating ]; h++ ) {
122       printf( "*" );
123    } /* end inner for */
124
125    printf( "\n" ); /* being new line of output */
126 } /* end outer for */
127
```

fig06_16.c (Part 6 of 8)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

fig06_16.c (Part 7 of 8)

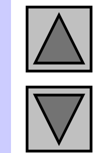
```
128    /* display the mode value */
129    printf( "The mode is the most frequent value.\n"
130            "For this run the mode is %d which occurred"
131            " %d times.\n", modeValue, largest );
132 } /* end function mode */
133
134 /* function that sorts an array with bubble sort algorithm */
135 void bubbleSort( int a[] )
136 {
137    int pass; /* counter */
138    int j;    /* counter */
139    int hold; /* temporary location used to swap elements */
140
141    /* loop to control number of passes */
142    for ( pass = 1; pass < SIZE; pass++ ) {
143
144       /* loop to control number of comparisons per pass */
145       for ( j = 0; j < SIZE - 1; j++ ) {
146
147          /* swap elements if out of order */
148          if ( a[ j ] > a[ j + 1 ] ) {
149             hold = a[ j ];
150             a[ j ] = a[ j + 1 ];
151             a[ j + 1 ] = hold;
152          } /* end if */
153
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
154       } /* end inner for */
155
156    } /* end outer for */
157
158} /* end function bubbleSort */
159
160/* output array contents (20 values per row) */
161void printArray( const int a[] )
162{
163    int j; /* counter */
164
165    /* output array contents */
166    for ( j = 0; j < SIZE; j++ ) {
167
168       if ( j % 20 == 0 ) { /* begin new line every 20 values */
169          printf( "\n" );
170       } /* end if */
171
172       printf( "%2d", a[ j ] );
173    } /* end for */
174
175} /* end function printArray */
```


fig06_16.c (Part 8 of 8)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Outline

Program Output

```
********
  Mean
********
The mean is the average value of the data
items. The mean is equal to the total of
all the data items divided by the number
of data items ( 99 ). The mean value for
this run is: 681 / 99 = 6.8788


********
 Median
********
The unsorted array of responses is
 6 7 8 9 8 7 8 9 8 9 7 8 9 5 9 8 7 8 7 8
 6 7 8 9 3 9 8 7 8 7 7 8 9 8 9 8 9 7 8 9
 6 7 8 7 8 7 9 8 9 2 7 8 9 8 9 8 9 7 5 3
 5 6 7 2 5 3 9 4 6 4 7 8 9 6 8 7 8 9 7 8
 7 4 4 2 5 3 8 7 5 6 4 5 6 1 6 5 7 8 7

The sorted array is
 1 2 2 2 3 3 3 3 4 4 4 4 4 5 5 5 5 5 5 5
 5 6 6 6 6 6 6 6 6 6 7 7 7 7 7 7 7 7 7 7
 7 7 7 7 7 7 7 7 7 7 7 7 7 8 8 8 8 8 8 8
 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Outline

Program Output (continued)

```
The median is element 49 of
the sorted 99 element array.
For this run the median is 7


********
  Mode
********
Response  Frequency          Histogram

                                     1    1    2    2
                                5    0    5    0    5

       1          1          *
       2          3          ***
       3          4          ****
       4          5          *****
       5          8          ********
       6          9          *********
       7         23          ***********************
       8         27          ***************************
       9         19          *******************
The mode is the most frequent value.
For this run the mode is 8 which occurred 27 times.
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

6.8 Searching Arrays: Linear Search and Binary Search

- Search an array for a *key value*
- Linear search
  - Simple
  - Compare each element of array with key value
  - Useful for small and unsorted arrays

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

6.8 Searching Arrays: Linear Search and Binary Search

- Binary search
  - For sorted arrays
  - Compares `middle` element with `key`
    - If equal, match found
    - If `key < middle`, looks in first half of array
    - If `key > middle`, looks in last half
    - Repeat
  - Very fast; at most n steps, where $2^n >$ number of elements
    - 30 element array takes at most 5 steps
      - $2^5 > 30$ so at most 5 steps

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
1  /* Fig. 6.18: fig06_18.c
2     Linear search of an array */
3  #include <stdio.h>
4  #define SIZE 100
5
6  /* function prototype */
7  int linearSearch( const int array[], int key, int size );
8
9  /* function main begins program execution */
10 int main()
11 {
12    int a[ SIZE ]; /* create array a */
13    int x;         /* counter */
14    int searchKey; /* value to locate in a */
15    int element;   /* variable to hold location of searchKey or -1 */
16
17    /* create data */
18    for ( x = 0; x < SIZE; x++ ) {
19       a[ x ] = 2 * x;
20    } /* end for */
21
22    printf( "Enter integer search key:\n" );
23    scanf( "%d", &searchKey );
24
```

Outline

fig06_18.c (Part 1 of 3)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

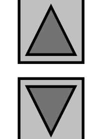


fig06_18.c (Part 2 of 3)

```
25    /* attempt to locate searchKey in array a */
26    element = linearSearch( a, searchKey, SIZE );
27
28    /* display results */
29    if ( element != -1 ) {
30       printf( "Found value in element %d\n", element );
31    } /* end if */
32    else {
33       printf( "Value not found\n" );
34    } /* end else */
35
36    return 0; /* indicates successful termination */
37
38 } /* end main */
39
40 /* compare key to every element of array until the location is found
41    or until the end of array is reached; return subscript of element
42    if key or -1 if key is not found */
43 int linearSearch( const int array[], int key, int size )
44 {
45    int n; /* counter */
46
47    /* loop through array */
48    for ( n = 0; n < size; ++n ) {
49
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
50        if ( array[ n ] == key ) {
51           return n; /* return location of key */
52        } /* end if */
53
54     } /* end for */
55
56     return -1; /* key not found */
57
58  } /* end function linearSearch */
```

```
Enter integer search key:
36
Found value in element 18


Enter integer search key:
37
Value not found
```


fig06_18.c (Part 3 of 3)

Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

fig06_19.c (Part 1 of 5)

```
1  /* Fig. 6.19: fig06_19.c
2     Binary search of an array */
3  #include <stdio.h>
4  #define SIZE 15
5
6  /* function prototypes */
7  int binarySearch( const int b[], int searchKey, int low, int high );
8  void printHeader( void );
9  void printRow( const int b[], int low, int mid, int high );
10
11 /* function main begins program execution */
12 int main()
13 {
14    int a[ SIZE ]; /* create array a */
15    int i;         /* counter */
16    int key;       /* value to locate in array a */
17    int result;    /* variable to hold location of key or -1 */
18
19    /* create data */
20    for ( i = 0; i < SIZE; i++ ) {
21       a[ i ] = 2 * i;
22    } /* end for */
23
24    printf( "Enter a number between 0 and 28: " );
25    scanf( "%d", &key );
26
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Outline

fig06_19.c (Part 2 of 5)

```
27    printHeader();
28
29    /* search for key in array a */
30    result = binarySearch( a, key, 0, SIZE - 1 );
31
32    /* display results */
33    if ( result != -1 ) {
34       printf( "\n%d found in array element %d\n", key, result );
35    } /* end if */
36    else {
37       printf( "\n%d not found\n", key );
38    } /* end else */
39
40    return 0; /* indicates successful termination */
41
42 } /* end main */
43
44 /* function to perform binary search of an array */
45 int binarySearch( const int b[], int searchKey, int low, int high )
46 {
47    int middle; /* variable to hold middle element of array */
48
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

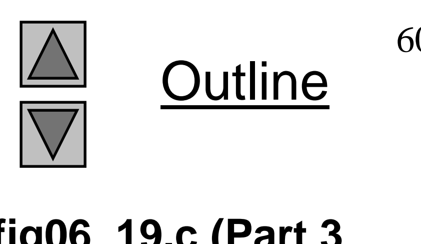
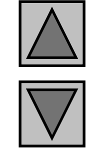


fig06_19.c (Part 3 of 5)

```
49    /* loop until low subscript is greater than high subscript */
50    while ( low <= high ) {
51
52       /* determine middle element of subarray being searched */
53       middle = ( low + high ) / 2;
54
55       /* display subarray used in this loop iteration */
56       printRow( b, low, middle, high );
57
58       /* if searchKey matched middle element, return middle */
59       if ( searchKey == b[ middle ] ) {
60          return middle;
61       } /* end if */
62
63       /* if searchKey less than middle element, set new high */
64       else if ( searchKey < b[ middle ] ) {
65          high = middle - 1; /* search low end of array */
66       } /* end else if */
67
68       /* if searchKey greater than middle element, set new low */
69       else {
70          low = middle + 1; /* search high end of array */
71       } /* end else */
72
73    } /* end while */
74
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Outline

fig06_19.c (Part 4 of 5)

```
75    return -1;   /* searchKey not found */
76
77 } /* end function binarySearch */
78
79 /* Print a header for the output */
80 void printHeader( void )
81 {
82    int i; /* counter */
83
84    printf( "\nSubscripts:\n" );
85
86    /* output column head */
87    for ( i = 0; i < SIZE; i++ ) {
88       printf( "%3d ", i );
89    } /* end for */
90
91    printf( "\n" ); /* start new line of output */
92
93    /* output line of - characters */
94    for ( i = 1; i <= 4 * SIZE; i++ ) {
95       printf( "-" );
96    } /* end for */
97
98    printf( "\n" ); /* start new line of output */
99 } /* end function printHeader */
100
```

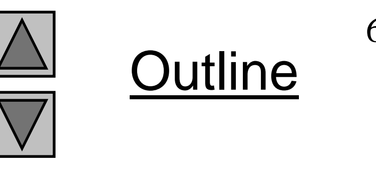
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Outline

fig06_19.c (Part 5 of 5)

```
101/* Print one row of output showing the current
102   part of the array being processed. */
103void printRow( const int b[], int low, int mid, int high )
104{
105   int i; /* counter */
106
107   /* loop through entire array */
108   for ( i = 0; i < SIZE; i++ ) {
109
110      /* display spaces if outside current subarray range */
111      if ( i < low || i > high ) {
112         printf( "    " );
113      } /* end if */
114      else if ( i == mid ) { /* display middle element */
115         printf( "%3d*", b[ i ] ); /* mark middle value */
116      } /* end else if */
117      else { /* display other elements in subarray */
118         printf( "%3d ", b[ i ] );
119      } /* end else */
120
121   } /* end for */
122
123   printf( "\n" ); /* start new line of output */
124} /* end function printRow */
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Program Output

```
Enter a number between 0 and 28: 25

Subscripts:
  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14
------------------------------------------------------------
  0   2   4   6   8  10  12  14* 16  18  20  22  24  26  28
                                 16  18  20  22* 24  26  28
                                             24  26* 28
                                             24*

25 not found

Enter a number between 0 and 28: 8

Subscripts:
  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14
------------------------------------------------------------
  0   2   4   6   8  10  12  14* 16  18  20  22  24  26  28
  0   2   4   6*  8  10  12
                  8  10* 12
                  8*

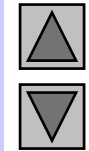
8 found in array element 4
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
Enter a number between 0 and 28: 6

Subscripts:
  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14
------------------------------------------------------------
  0   2   4   6   8  10  12  14* 16  18  20  22  24  26  28
  0   2   4   6*  8  10  12

6 found in array element 3
```

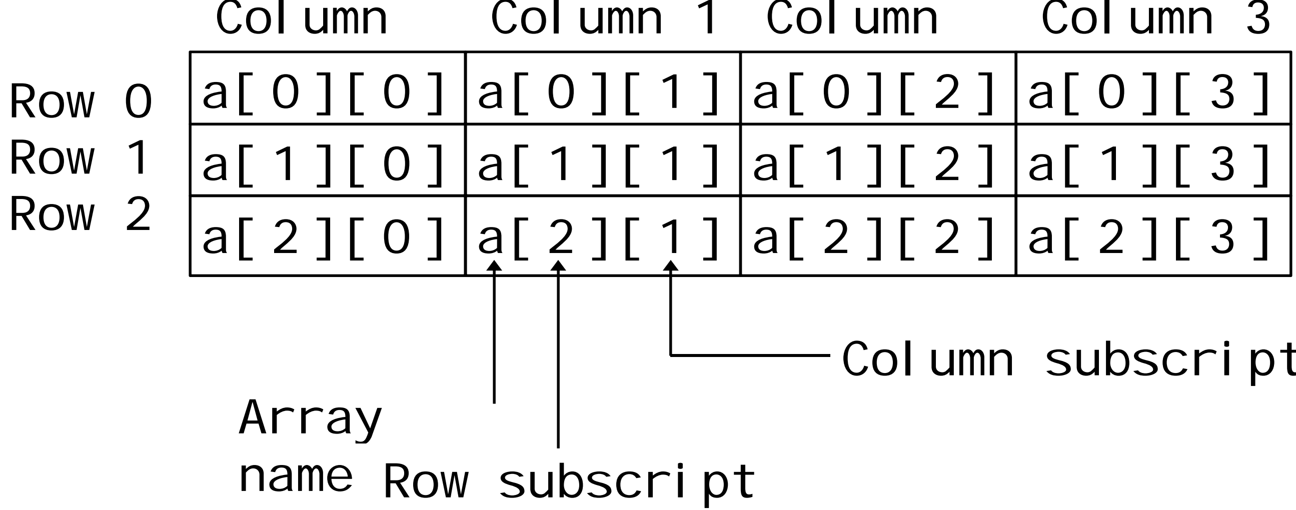


Program Output (continued)

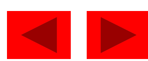
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

6.9 Multiple-Subscripted Arrays

- Multiple subscripted arrays
  - Tables with rows and columns (`m` by `n` array)
  - Like matrices: specify row, then column



© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



6.9 Multiple-Subscripted Arrays

- Initialization
  - `int b[ 2 ][ 2 ] = { { 1, 2 }, { 3, 4 } };`
  - Initializers grouped by row in braces
  - If not enough, unspecified elements set to zero

```
int b[ 2 ][ 2 ] = { { 1 }, { 3, 4 } };
```

1	2
3	4

1	0
3	4

- Referencing elements
  - Specify row, then column

```
printf( "%d", b[ 0 ][ 1 ] );
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Outline

fig06_21.c (Part 1 of 2)

```
1  /* Fig. 6.21: fig06_21.c
2     Initializing multidimensional arrays */
3  #include <stdio.h>
4
5  void printArray( const int a[][ 3 ] ); /* function prototype */
6
7  /* function main begins program execution */
8  int main()
9  {
10    /* initialize array1, array2, array3 */
11    int array1[ 2 ][ 3 ] = { { 1, 2, 3 }, { 4, 5, 6 } };
12    int array2[ 2 ][ 3 ] = { 1, 2, 3, 4, 5 };
13    int array3[ 2 ][ 3 ] = { { 1, 2 }, { 4 } };
14
15    printf( "Values in array1 by row are:\n" );
16    printArray( array1 );
17
18    printf( "Values in array2 by row are:\n" );
19    printArray( array2 );
20
21    printf( "Values in array3 by row are:\n" );
22    printArray( array3 );
23
24    return 0; /* indicates successful termination */
25
26 } /* end main */
27
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
28 /* function to output array with two rows and three columns */
29 void printArray( const int a[][ 3 ] )
30 {
31    int i; /* counter */
32    int j; /* counter */
33 
34    /* loop through rows */
35    for ( i = 0; i <= 1; i++ ) {
36 
37       /* output column values */
38       for ( j = 0; j <= 2; j++ ) {
39          printf( "%d ", a[ i ][ j ] );
40       } /* end inner for */
41 
42       printf( "\n" ); /* start new line of output */
43    } /* end outer for */
44 
45 } /* end function printArray */
```

```
Values in array1 by row are:
1 2 3
4 5 6
Values in array2 by row are:
1 2 3
4 5 0
Values in array3 by row are:
1 2 0
4 0 0
```


fig06_21.c (Part 2 of 2)

Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

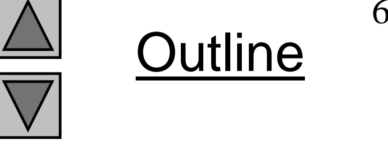


fig06_22.c (Part 1 of 6)

```
1  /* Fig. 6.22: fig06_22.c
2     Double-subscripted array example */
3  #include <stdio.h>
4  #define STUDENTS 3
5  #define EXAMS 4
6
7  /* function prototypes */
8  int minimum( const int grades[][ EXAMS ], int pupils, int tests );
9  int maximum( const int grades[][ EXAMS ], int pupils, int tests );
10 double average( const int setOfGrades[], int tests );
11 void printArray( const int grades[][ EXAMS ], int pupils, int tests );
12
13 /* function main begins program execution */
14 int main()
15 {
16    int student; /* counter */
17
18    /* initialize student grades for three students (rows) */
19    const int studentGrades[ STUDENTS ][ EXAMS ] =
20       { { 77, 68, 86, 73 },
21         { 96, 87, 89, 78 },
22         { 70, 90, 86, 81 } };
23
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

fig06_22.c (Part 2 of 6)

```
24    /* output array studentGrades */
25    printf( "The array is:\n" );
26    printArray( studentGrades, STUDENTS, EXAMS );
27
28    /* determine smallest and largest grade values */
29    printf( "\n\nLowest grade: %d\nHighest grade: %d\n",
30       minimum( studentGrades, STUDENTS, EXAMS ),
31       maximum( studentGrades, STUDENTS, EXAMS ) );
32
33    /* calculate average grade for each student */
34    for ( student = 0; student <= STUDENTS - 1; student++ ) {
35       printf( "The average grade for student %d is %.2f\n",
36          student, average( studentGrades[ student ], EXAMS ) );
37    } /* end for */
38
39    return 0; /* indicates successful termination */
40
41 } /* end main */
42
43 /* Find the minimum grade */
44 int minimum( const int grades[][ EXAMS ], int pupils, int tests )
45 {
46    int i;              /* counter */
47    int j;              /* counter */
48    int lowGrade = 100; /* initialize to highest possible grade */
49
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
50    /* loop through rows of grades */
51    for ( i = 0; i < pupils; i++ ) {
52
53       /* loop through columns of grades */
54       for ( j = 0; j < tests; j++ ) {
55
56          if ( grades[ i ][ j ] < lowGrade ) {
57             lowGrade = grades[ i ][ j ];
58          } /* end if */
59
60       } /* end inner for */
61
62    } /* end outer for */
63
64    return lowGrade; /* return minimum grade */
65
66 } /* end function minimum */
67
68 /* Find the maximum grade */
69 int maximum( const int grades[][ EXAMS ], int pupils, int tests )
70 {
71    int i;              /* counter */
72    int j;              /* counter */
73    int highGrade = 0; /* initialize to lowest possible grade */
74
```

Outline

fig06_22.c (Part 3 of 6)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
75    /* loop through rows of grades */
76    for ( i = 0; i < pupils; i++ ) {
77
78       /* loop through columns of grades */
79       for ( j = 0; j < tests; j++ ) {
80
81          if ( grades[ i ][ j ] > highGrade ) {
82             highGrade = grades[ i ][ j ];
83          } /* end if */
84
85       } /* end inner for */
86
87    } /* end outer for */
88
89    return highGrade; /* return maximum grade */
90
91 } /* end function maximum */
92
93 /* Determine the average grade for a particular student */
94 double average( const int setOfGrades[], int tests )
95 {
96    int i;          /* counter */
97    int total = 0; /* sum of test grades */
98
```


fig06_22.c (Part 4 of 6)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

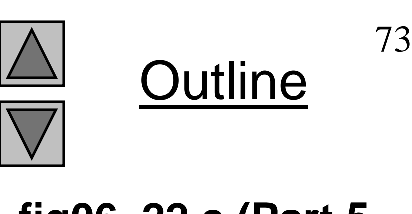


fig06_22.c (Part 5 of 6)

```
99     /* total all grades for one student */
100     for ( i = 0; i < tests; i++ ) {
101        total += setOfGrades[ i ];
102    } /* end for */
103
104    return ( double ) total / tests; /* average */
105
106} /* end function average */
107
108/* Print the array */
109void printArray( const int grades[][ EXAMS ], int pupils, int tests )
110{
111    int i; /* counter */
112    int j; /* counter */
113
114    /* output column heads */
115    printf( "                 [0]  [1]  [2]  [3]" );
116
117    /* output grades in tabular format */
118    for ( i = 0; i < pupils; i++ ) {
119
120       /* output label for row */
121       printf( "\nstudentGrades[%d] ", i );
122
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

fig06_22.c (Part 6 of 6)

```
123       /* output grades for one student */
124       for ( j = 0; j < tests; j++ ) {
125          printf( "%-5d", grades[ i ][ j ] );
126       } /* end inner for */
127
128    } /* end outer for */
129
130} /* end function printArray */
```

```
The array is:
                 [0]  [1]  [2]  [3]
studentGrades[0] 77   68   86   73
studentGrades[1] 96   87   89   78
studentGrades[2] 70   90   86   81

Lowest grade: 68
Highest grade: 96
The average grade for student 0 is 76.00
The average grade for student 1 is 87.50
The average grade for student 2 is 81.75
```

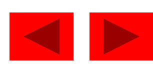
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Chapter 7 - Pointers

Outline

7.1 Introduction
7.2 Pointer Variable Definitions and Initialization
7.3 Pointer Operators
7.4 Calling Functions by Reference
7.5 Using the const Qualifier with Pointers
7.6 Bubble Sort Using Call by Reference
7.7 Pointer Expressions and Pointer Arithmetic
7.8 The Relationship between Pointers and Arrays
7.9 Arrays of Pointers
7.10 Case Study: A Card Shuffling and Dealing Simulation
7.11 Pointers to Functions

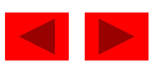
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Objectives

- In this chapter, you will learn:
  - To be able to use pointers.
  - To be able to use pointers to pass arguments to functions using call by reference.
  - To understand the close relationships among pointers, arrays and strings.
  - To understand the use of pointers to functions.
  - To be able to define and use arrays of strings.

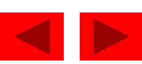
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



7.1 Introduction

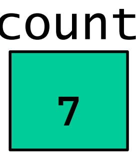
- Pointers
  - Powerful, but difficult to master
  - Simulate call-by-reference
  - Close relationship with arrays and strings

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

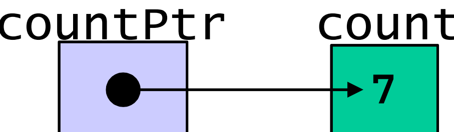


7.2 Pointer Variable Definitions and Initialization

- Pointer variables
  - Contain memory addresses as their values
  - Normal variables contain a specific value (direct reference)



  - Pointers contain address of a variable that has a specific value (indirect reference)
  - Indirection – referencing a pointer value



© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

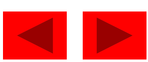
7.2 Pointer Variable Definitions and Initialization

- Pointer definitions
  - `*` used with pointer variables

    ```
    int *myPtr;
    ```
  - Defines a pointer to an `int` (pointer of type `int *`)
  - Multiple pointers require using a `*` before each variable definition

    ```
    int *myPtr1, *myPtr2;
    ```
  - Can define pointers to any data type
  - Initialize pointers to `0`, `NULL`, or an address
    - `0` or `NULL` – points to nothing (`NULL` preferred)

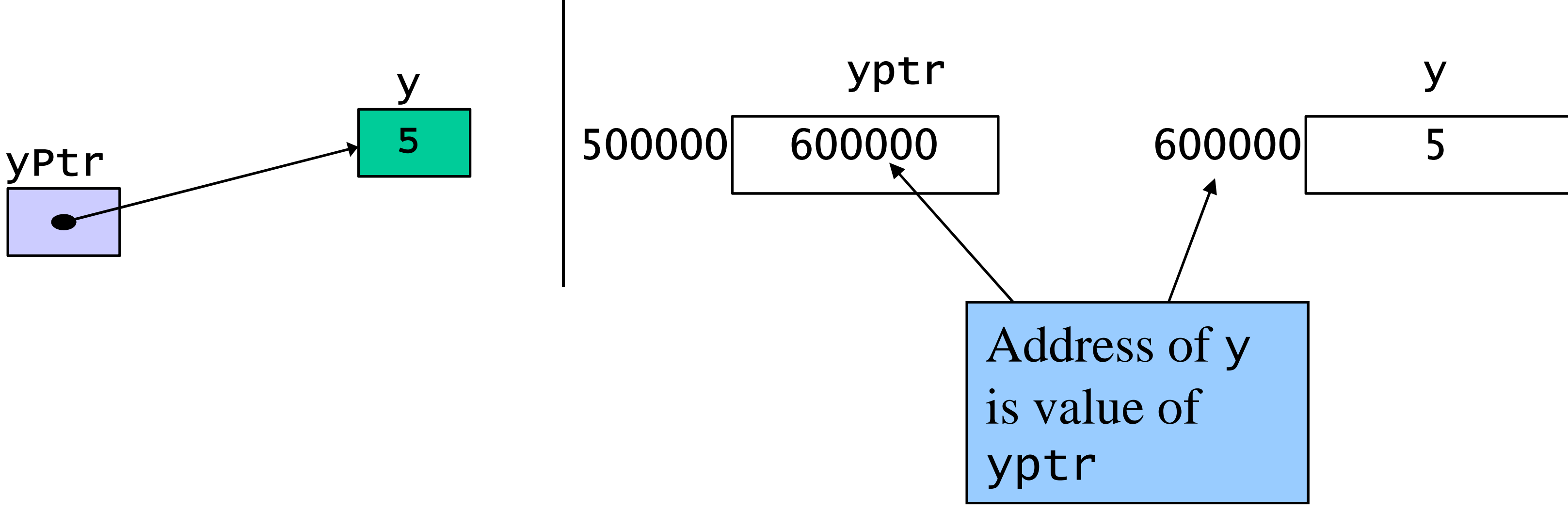
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



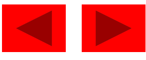
7.3 Pointer Operators

- & (address operator)
  - Returns address of operand

```
int y = 5;
int *yPtr;
yPtr = &y;     /* yPtr gets address of y */
yPtr “points to” y
```



© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



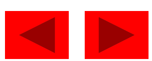
7.3 Pointer Operators

- `*` (indirection/dereferencing operator)
  – Returns a synonym/alias of what its operand points to
  – `*yptr` returns `y` (because `yptr` points to `y`)
  – `*` can be used for assignment
    - Returns alias to an object

```
*yptr = 7;  /* changes y to 7 */
```

  – Dereferenced pointer (operand of `*`) must be an lvalue (no constants)
- `*` and `&` are inverses
  – They cancel each other out

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Outline

fig07_04.c

```
1  /* Fig. 7.4: fig07_04.c
2     Using the & and * operators */
3  #include <stdio.h>
4
5  int main()
6  {
7     int a;        /* a is an integer */
8     int *aPtr;    /* aPtr is a pointer to an integer */
9
10    a = 7;
11    aPtr = &a;    /* aPtr set to address of a */
12
13    printf( "The address of a is %p"
14            "\nThe value of aPtr is %p", &a, aPtr );
15
16    printf( "\n\nThe value of a is %d"
17            "\nThe value of *aPtr is %d", a, *aPtr );
18
19    printf( "\n\nShowing that * and & are complements of "
20            "each other\n&*aPtr = %p"
21            "\n*&aPtr = %p\n", &*aPtr, *&aPtr );
22
23    return 0; /* indicates successful termination */
24
25 } /* end main */
```

The address of a is the value of aPtr.

The * operator returns an alias to what its operand points to. aPtr points to a, so *aPtr returns a.

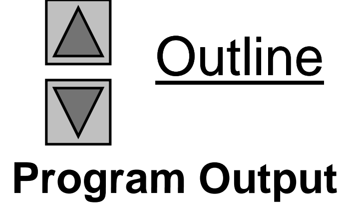
Notice how * and & are inverses

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
The address of a is 0012FF7C
The value of aPtr is 0012FF7C

The value of a is 7
The value of *aPtr is 7

Showing that * and & are complements of each other.
&*aPtr = 0012FF7C
*&aPtr = 0012FF7C
```



Program Output

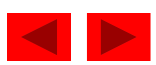
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

7.3 Pointer Operators

Operators								Associativity	Type
()	[]							left to right	highest
+	-	++	--	!	*	&	(type)	right to left	unary
*	/	%						left to right	multiplicative
+	-							left to right	additive
<	<=	>	>=					left to right	relational
==	!=							left to right	equality
&&								left to right	logical and
||								left to right	logical or
?:								right to left	conditional
=	+=	-=	*=	/=	%=			right to left	assignment
,								left to right	comma

Fig. 7.5 Operator precedence.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



7.4 Calling Functions by Reference

- Call by reference with pointer arguments
  - Pass address of argument using & operator
  - Allows you to change actual location in memory
  - Arrays are not passed with & because the array name is already a pointer
- * operator
  - Used as alias/nickname for variable inside of function

```
void double( int *number )
 {
  *number = 2 * ( *number );
 }
```

  - `*number` used as nickname for the variable passed

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

fig07_06.c

```
1  /* Fig. 7.6: fig07_06.c
2     Cube a variable using call-by-value */
3  #include <stdio.h>
4
5  int cubeByValue( int n ); /* prototype */
6
7  int main()
8  {
9     int number = 5; /* initialize number */
10
11    printf( "The original value of number is %d", number );
12
13    /* pass number by value to cubeByValue */
14    number = cubeByValue( number );
15
16    printf( "\nThe new value of number is %d\n", number );
17
18    return 0; /* indicates successful termination */
19
20 } /* end main */
21
22 /* calculate and return cube of integer argument */
23 int cubeByValue( int n )
24 {
25    return n * n * n;   /* cube local variable n and return result */
26
27 } /* end function cubeByValue */
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
The original value of number is 5
The new value of number is 125
```

Outline

Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Outline

fig07_07.c

```
1  /* Fig. 7.7: fig07_07.c
2     Cube a variable using call-by-reference with a pointer argument */
3
4  #include <stdio.h>
5
6  void cubeByReference( int *nPtr ); /* prototype */
7
8  int main()
9  {
10    int number = 5; /* initialize number */
11
12    printf( "The original value of number is %d", number );
13
14    /* pass address of number to cubeByReference */
15    cubeByReference( &number );
16
17    printf( "\nThe new value of number is %d\n", number );
18
19    return 0; /* indicates successful termination */
20
21 } /* end main */
22
23 /* calculate cube of *nPtr; modifies variable number in main */
24 void cubeByReference( int *nPtr )
25 {
26    *nPtr = *nPtr * *nPtr * *nPtr;  /* cube *nPtr */
27 } /* end function cubeByReference */
```

Notice that the function prototype takes a pointer to an integer.

Notice how the address of `number` is given - `cubeByReference` expects a pointer (an address of a variable).

Inside `cubeByReference`, `*nPtr` is used (`*nPtr` is `number`).

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
The original value of number is 5
The new value of number is 125
```

Outline

Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Before `main` calls `cubeByValue` :

```
int main()
{
    int number = 5;

    number=cubeByValue(number);
}
```

number: 5

```
int cubeByValue( int n )
{
    return n * n * n;
}
```

n: undefined

After `cubeByValue` receives the call:

```
int main()
{
    int number = 5;

    number = cubeByValue( number );
}
```

number: 5

```
int cubeByValue( int n )
{
    return n * n * n;
}
```

n: 5

After `cubeByValue` cubes parameter `n` and before `cubeByValue` returns to `main` :

```
int main()
{
    int number = 5;

    number = cubeByValue( number );
}
```

number: 5

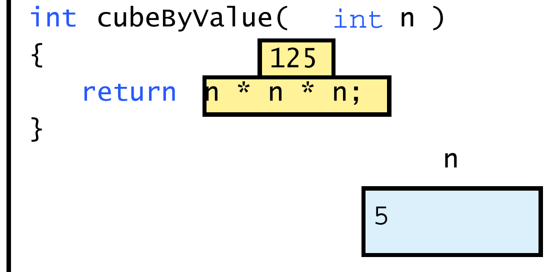
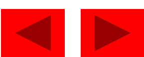


Fig. 7.8 Analysis of a typical call-by-value. (Part 1 of 2.)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



After `cubeByValue` returns to `main` and before assigning the result to `number`:

```
int main()                                  number
{                                           [ 5 ]
   int number = 5;
                         125
      number = cubeByValue( number );
}
```

```
int cubeByValue( int n )
{
   return n * n * n;
}
                                            n
                                      [ undefined ]
```

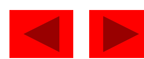
After `main` completes the assignment to `number`:

```
int main()                                  number
{                                           [ 125 ]
   int number = 5;
      125                125
   number = cubeByValue( number );
}
```

```
int cubeByValue( int n )
{
   return n * n * n;
}
                                            n
                                      [ undefined ]
```

Fig. 7.8 Analysis of a typical call-by-value. (Part 2 of 2.)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Before `main` calls `cubeByReference` :

```
int main()                              number
{                                        5
   int number = 5;

   cubeByReference( &number );
}
```

```
void cubeByReference( int *nPtr )
{
   *nPtr = *nPtr * *nPtr * *nPtr;
}
                                nPtr
                             undefined
```

After `cubeByReference` receives the call and before `*nPtr` is cubed:

```
int main()                              number
{                                        5
   int number = 5;

   cubeByReference( &number );
}
```

```
void cubeByReference( int *nPtr )
{
   *nPtr = *nPtr * *nPtr * *nPtr;
}
                                nPtr
```

call establishes this pointer

After `*nPtr` is cubed and before program control returns to `main` :

```
int main()                              number
{                                        125
   int number = 5;

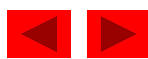
   cubeByReference( &number );
}
```

```
void cubeByReference( int *nPtr )
{                      125
   *nPtr = *nPtr * *nPtr * *nPtr;
}
                                nPtr
```

called function modifies caller's variable

Fig. 7.9 Analysis of a typical call-by-reference with a pointer argument.

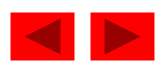
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



7.5 Using the `const` Qualifier with Pointers

- `const` qualifier
  - Variable cannot be changed
  - Use `const` if function does not need to change a variable
  - Attempting to change a `const` variable produces an error
- `const` pointers
  - Point to a constant memory location
  - Must be initialized when defined
  - `int *const myPtr = &x;`
    - Type `int *const` – constant pointer to an `int`
  - `const int *myPtr = &x;`
    - Regular pointer to a `const int`
  - `const int *const Ptr = &x;`
    - `const` pointer to a `const int`
    - `x` can be changed, but not `*Ptr`

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



```c
1  /* Fig. 7.10: fig07_10.c
2     Converting lowercase letters to uppercase letters
3     using a non-constant pointer to non-constant data */
4
5  #include <stdio.h>
6  #include <ctype.h>
7
8  void convertToUppercase( char *sPtr ); /* prototype */
9
10 int main()
11 {
12    char string[] = "characters and $32.98"; /* initialize char array */
13
14    printf( "The string before conversion is: %s", string );
15    convertToUppercase( string );
16    printf( "\nThe string after conversion is: %s\n", string );
17
18    return 0; /* indicates successful termination */
19
20 } /* end main */
21
```

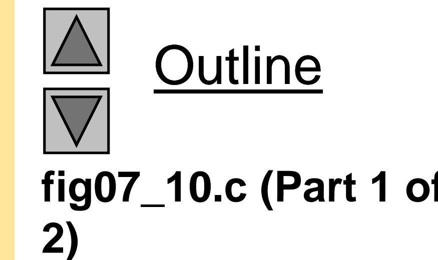


fig07_10.c (Part 1 of 2)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
22 /* convert string to uppercase letters */
23 void convertToUppercase( char *sPtr )
24 {
25    while ( *sPtr != '\0' ) { /* current character is not '\0' */
26
27       if ( islower( *sPtr ) ) {    /* if character is lowercase, */
28          *sPtr = toupper( *sPtr ); /* convert to uppercase */
29       } /* end if */
30
31       ++sPtr;  /* move sPtr to the next character */
32    } /* end while */
33
34 } /* end function convertToUppercase */
```

```
The string before conversion is: characters and $32.98
The string after conversion is: CHARACTERS AND $32.98
```

Outline

fig07_10.c (Part 2 of 2)

Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
1  /* Fig. 7.11: fig07_11.c
2     Printing a string one character at a time using
3     a non-constant pointer to constant data */
4
5  #include <stdio.h>
6
7  void printCharacters( const char *sPtr );
8
9  int main()
10 {
11    /* initialize char array */
12    char string[] = "print characters of a string";
13
14    printf( "The string is:\n" );
15    printCharacters( string );
16    printf( "\n" );
17
18    return 0; /* indicates successful termination */
19
20 } /* end main */
21
```

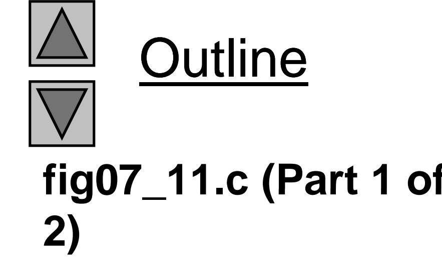


fig07_11.c (Part 1 of 2)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
22 /* sPtr cannot modify the character to which it points,
23    i.e., sPtr is a "read-only" pointer */
24 void printCharacters( const char *sPtr )
25 {
26    /* loop through entire string */
27    for ( ; *sPtr != '\0'; sPtr++ ) { /* no initialization */
28       printf( "%c", *sPtr );
29    } /* end for */
30
31 } /* end function printCharacters */
```

```
The string is:
print characters of a string
```

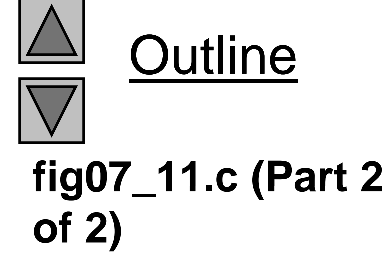


fig07_11.c (Part 2 of 2)

Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

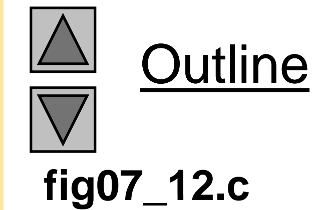
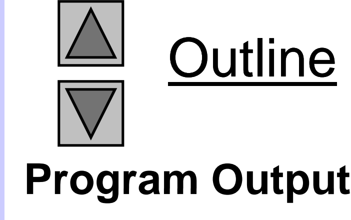


fig07_12.c

```
1  /* Fig. 7.12: fig07_12.c
2     Attempting to modify data through a
3     non-constant pointer to constant data. */
4  #include <stdio.h>
5
6  void f( const int *xPtr ); /* prototype */
7
8  int main()
9  {
10    int y;       /* define y */
11
12    f( &y );     /* f attempts illegal modification */
13
14    return 0;    /* indicates successful termination */
15
16 } /* end main */
17
18 /* xPtr cannot be used to modify the
19    value of the variable to which it points */
20 void f( const int *xPtr )
21 {
22    *xPtr = 100;  /* error: cannot modify a const object */
23 } /* end function f */
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
Compiling...
FIG07_12.c
d:\books\2003\chtp4\examples\ch07\fig07_12.c(22) : error C2166: l-value
   specifies const object
Error executing cl.exe.

FIG07_12.exe - 1 error(s), 0 warning(s)
```



Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

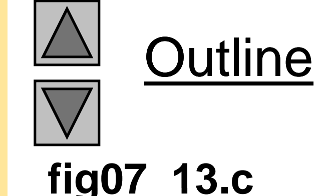


fig07_13.c

```
1  /* Fig. 7.13: fig07_13.c
2     Attempting to modify a constant pointer to non-constant data */
3  #include <stdio.h>
4
5  int main()
6  {
7     int x; /* define x */
8     int y; /* define y */
9
10    /* ptr is a constant pointer to an integer that can be modified
11       through ptr, but ptr always points to the same memory location */
12    int * const ptr = &x;
13
14    *ptr = 7; /* allowed: *ptr is not const */
15    ptr = &y; /* error: ptr is const; cannot assign new address */
16
17    return 0; /* indicates successful termination */
18
19 } /* end main */
```

Changing `*ptr` is allowed – x is not a constant.

Changing `ptr` is an error – `ptr` is a constant pointer.

Program Output

```
Compiling...
FIG07_13.c
D:\books\2003\chtp4\Examples\ch07\FIG07_13.c(15) : error C2166: l-value
   specifies- const object
Error executing cl.exe.

FIG07_13.exe - 1 error(s), 0 warning(s)
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
1  /* Fig. 7.14: fig07_14.c
2     Attempting to modify a constant pointer to constant data. */
3  #include <stdio.h>
4
5  int main()
6  {
7     int x = 5; /* initialize x */
8     int y;     /* define y */
9
10    /* ptr is a constant pointer to a constant integer. ptr always
11       points to the same location; the integer at that location
12       cannot be modified */
13    const int *const ptr = &x;
14
15    printf( "%d\n", *ptr );
16
17    *ptr = 7; /* error: *ptr is const; cannot assign new value */
18    ptr = &y; /* error: ptr is const; cannot assign new address */
19
20    return 0; /* indicates successful termination */
21
22 } /* end main */
```

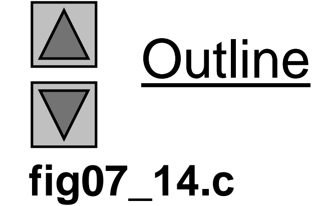
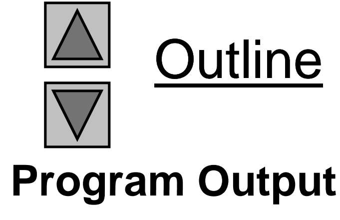


fig07_14.c

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
Compiling...
FIG07_14.c
D:\books\2003\chtp4\Examples\ch07\FIG07_14.c(17) : error C2166: l-value
   specifies- const object
D:\books\2003\chtp4\Examples\ch07\FIG07_14.c(18) : error C2166: l-value
   specifies- const object
Error executing cl.exe.

FIG07_12.exe - 2 error(s), 0 warning(s)
```



Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

7.6 Bubble Sort Using Call-by-reference

- Implement bubblesort using pointers
  - Swap two elements
  - `swap` function must receive address (using `&`) of array elements
    - Array elements have call-by-value default
  - Using pointers and the `*` operator, `swap` can switch array elements
- Psuedocode

    Initialize array

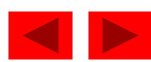
    print data in original order

    Call function bubblesort

    print sorted array

    Define bubblesort

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



7.6 Bubble Sort Using Call-by-reference

- `sizeof`
  - Returns size of operand in bytes
  - For arrays: size of 1 element * number of elements
  - if `sizeof( int )` equals 4 bytes, then

```
int myArray[ 10 ];
printf( "%d", sizeof( myArray ) );
```

    - will print 40
- `sizeof` can be used with
  - Variable names
  - Type name
  - Constant values

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

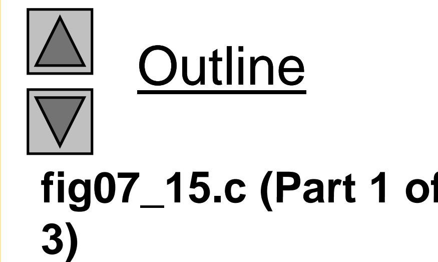
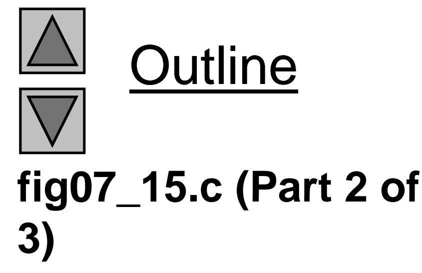


fig07_15.c (Part 1 of 3)

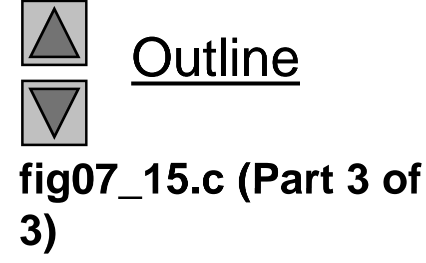
```
1  /* Fig. 7.15: fig07_15.c
2     This program puts values into an array, sorts the values into
3     ascending order, and prints the resulting array. */
4  #include <stdio.h>
5  #define SIZE 10
6
7  void bubbleSort( int *array, const int size ); /* prototype */
8
9  int main()
10 {
11    /* initialize array a */
12    int a[ SIZE ] = { 2, 6, 4, 8, 10, 12, 89, 68, 45, 37 };
13
14    int i; /* counter */
15
16    printf( "Data items in original order\n" );
17
18    /* loop through array a */
19    for ( i = 0; i < SIZE; i++ ) {
20       printf( "%4d", a[ i ] );
21    } /* end for */
22
23    bubbleSort( a, SIZE ); /* sort the array */
24
25    printf( "\nData items in ascending order\n" );
26
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
27    /* loop through array a */
28    for ( i = 0; i < SIZE; i++ ) {
29       printf( "%4d", a[ i ] );
30    } /* end for */
31
32    printf( "\n" );
33
34    return 0; /* indicates successful termination */
35
36 } /* end main */
37
38 /* sort an array of integers using bubble sort algorithm */
39 void bubbleSort( int *array, const int size )
40 {
41    void swap( int *element1Ptr, int *element2Ptr ); /* prototype */
42    int pass; /* pass counter */
43    int j;    /* comparison counter */
44
45    /* loop to control passes */
46    for ( pass = 0; pass < size - 1; pass++ ) {
47
48       /* loop to control comparisons during each pass */
49       for ( j = 0; j < size - 1; j++ ) {
50
```



© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Outline

fig07_15.c (Part 3 of 3)

```
51          /* swap adjacent elements if they are out of order */
52          if ( array[ j ] > array[ j + 1 ] ) {
53             swap( &array[ j ], &array[ j + 1 ] );
54          } /* end if */
55
56       } /* end inner for */
57
58    } /* end outer for */
59
60 } /* end function bubbleSort */
61
62 /* swap values at memory locations to which element1Ptr and
63    element2Ptr point */
64 void swap( int *element1Ptr, int *element2Ptr )
65 {
66    int hold = *element1Ptr;
67    *element1Ptr = *element2Ptr;
68    *element2Ptr = hold;
69 } /* end function swap */
```

Program Output

```
Data items in original order
   2   6   4   8  10  12  89  68  45  37
Data items in ascending order
   2   4   6   8  10  12  37  45  68  89
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
1  /* Fig. 7.16: fig07_16.c
2     Sizeof operator when used on an array name
3     returns the number of bytes in the array. */
4  #include <stdio.h>
5
6  size_t getSize( float *ptr ); /* prototype */
7
8  int main()
9  {
10    float array[ 20 ]; /* create array */
11
12    printf( "The number of bytes in the array is %d"
13            "\nThe number of bytes returned by getSize is %d\n",
14            sizeof( array ), getSize( array ) );
15
16    return 0; /* indicates successful termination */
17
18 } /* end main */
19
20 /* return size of ptr */
21 size_t getSize( float *ptr )
22 {
23    return sizeof( ptr );
24
25 } /* end function getSize */
```

```
The number of bytes in the array is 80
The number of bytes returned by getSize is 4
```

fig07_16.c

Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

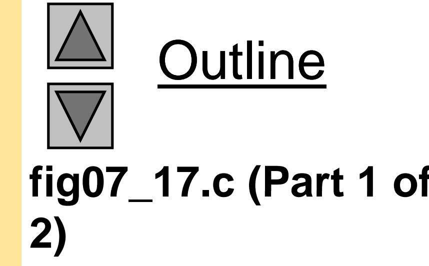


fig07_17.c (Part 1 of 2)

```
1  /* Fig. 7.17: fig07_17.c
2     Demonstrating the sizeof operator */
3  #include <stdio.h>
4
5  int main()
6  {
7     char c;           /* define c */
8     short s;          /* define s */
9     int i;            /* define i */
10    long l;           /* define l */
11    float f;          /* define f */
12    double d;         /* define d */
13    long double ld;   /* define ld */
14    int array[ 20 ];  /* initialize array */
15    int *ptr = array; /* create pointer to array */
16
17    printf( "     sizeof c = %d\tsizeof(char)  = %d"
18            "\n     sizeof s = %d\tsizeof(short) = %d"
19            "\n     sizeof i = %d\tsizeof(int) = %d"
20            "\n     sizeof l = %d\tsizeof(long) = %d"
21            "\n     sizeof f = %d\tsizeof(float) = %d"
22            "\n     sizeof d = %d\tsizeof(double) = %d"
23            "\n    sizeof ld = %d\tsizeof(long double) = %d"
24            "\n sizeof array = %d"
25            "\n   sizeof ptr = %d\n",
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
26           sizeof c, sizeof( char ), sizeof s,
27           sizeof( short ), sizeof i, sizeof( int ),
28           sizeof l, sizeof( long ), sizeof f,
29           sizeof( float ), sizeof d, sizeof( double ),
30           sizeof ld, sizeof( long double ),
31           sizeof array, sizeof ptr );
32
33    return 0; /* indicates successful termination */
34
35 } /* end main */
```

```
     sizeof c = 1       sizeof(char)  = 1
     sizeof s = 2       sizeof(short) = 2
     sizeof i = 4       sizeof(int) = 4
     sizeof l = 4       sizeof(long) = 4
     sizeof f = 4       sizeof(float) = 4
     sizeof d = 8       sizeof(double) = 8
    sizeof ld = 8       sizeof(long double) = 8
 sizeof array = 80
   sizeof ptr = 4
```

Outline

fig07_17.c (Part 2 of 2)

Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

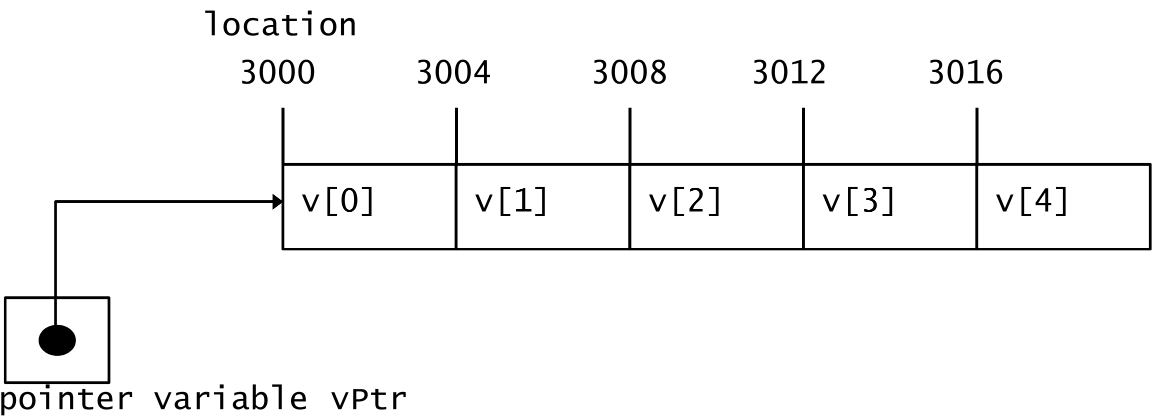
7.7 Pointer Expressions and Pointer Arithmetic

- Arithmetic operations can be performed on pointers
  - Increment/decrement pointer (++ or --)
  - Add an integer to a pointer(+ or += , - or -=)
  - Pointers may be subtracted from each other
  - Operations meaningless unless performed on an array

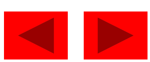
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

7.7 Pointer Expressions and Pointer Arithmetic

- 5 element `int` array on machine with 4 byte `ints`
  - `vPtr` points to first element `v[ 0 ]`
    - at location `3000 (vPtr = 3000)`
  - `vPtr += 2;` sets `vPtr` to `3008`
    - `vPtr` points to `v[ 2 ]` (incremented by 2), but the machine has 4 byte `ints`, so it points to address `3008`



© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



7.7 Pointer Expressions and Pointer Arithmetic

- Subtracting pointers
  - Returns number of elements from one to the other. If

```
vPtr2 = v[ 2 ];
vPtr = v[ 0 ];
```

  - `vPtr2 - vPtr` would produce 2
- Pointer comparison (<, == , >)
  - See which pointer points to the higher numbered array element
  - Also, see if a pointer points to `0`

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

7.7 Pointer Expressions and Pointer Arithmetic

- Pointers of the same type can be assigned to each other
  - If not the same type, a cast operator must be used
  - Exception: pointer to `void` (type `void *`)
    - Generic pointer, represents any type
    - No casting needed to convert a pointer to `void` pointer
    - `void` pointers cannot be dereferenced

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

7.8 The Relationship Between Pointers and Arrays

- Arrays and pointers closely related
  - Array name like a constant pointer
  - Pointers can do array subscripting operations
- Define an array `b[ 5 ]` and a pointer `bPtr`
  - To set them equal to one another use:

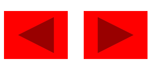
    ```
    bPtr = b;
    ```

    - The array name (`b`) is actually the address of first element of the array `b[ 5 ]`

    ```
    bPtr = &b[ 0 ]
    ```

    - Explicitly assigns `bPtr` to address of first element of `b`

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



7.8 The Relationship Between Pointers and Arrays

- Element `b[ 3 ]`
  - Can be accessed by `*( bPtr + 3 )`
    - Where `n` is the offset. Called pointer/offset notation
  - Can be accessed by `bptr[ 3 ]`
    - Called pointer/subscript notation
    - `bPtr[ 3 ]` same as `b[ 3 ]`
  - Can be accessed by performing pointer arithmetic on the array itself

```
*( b + 3 )
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
1  /* Fig. 7.20: fig07_20.cpp
2     Using subscripting and pointer notations with arrays */
3
4  #include <stdio.h>
5
6  int main()
7  {
8     int b[] = { 10, 20, 30, 40 }; /* initialize array b */
9     int *bPtr = b;                /* set bPtr to point to array b */
10    int i;                        /* counter */
11    int offset;                   /* counter */
12
13    /* output array b using array subscript notation */
14    printf( "Array b printed with:\nArray subscript notation\n" );
15
16    /* loop through array b */
17    for ( i = 0; i < 4; i++ ) {
18       printf( "b[ %d ] = %d\n", i, b[ i ] );
19    } /* end for */
20
21    /* output array b using array name and pointer/offset notation */
22    printf( "\nPointer/offset notation where\n"
23            "the pointer is the array name\n" );
24
```

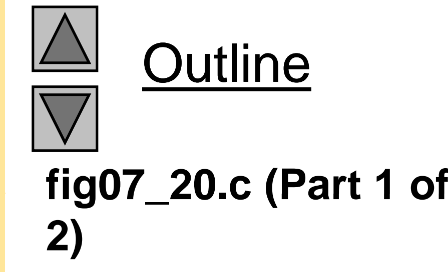


fig07_20.c (Part 1 of 2)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
25    /* loop through array b */
26    for ( offset = 0; offset < 4; offset++ ) {
27       printf( "*( b + %d ) = %d\n", offset, *( b + offset ) );
28    } /* end for */
29
30    /* output array b using bPtr and array subscript notation */
31    printf( "\nPointer subscript notation\n" );
32
33    /* loop through array b */
34    for ( i = 0; i < 4; i++ ) {
35       printf( "bPtr[ %d ] = %d\n", i, bPtr[ i ] );
36    } /* end for */
37
38    /* output array b using bPtr and pointer/offset notation */
39    printf( "\nPointer/offset notation\n" );
40
41    /* loop through array b */
42    for ( offset = 0; offset < 4; offset++ ) {
43       printf( "*( bPtr + %d ) = %d\n", offset, *( bPtr + offset ) );
44    } /* end for */
45
46    return 0; /* indicates successful termination */
47
48 } /* end main */
```

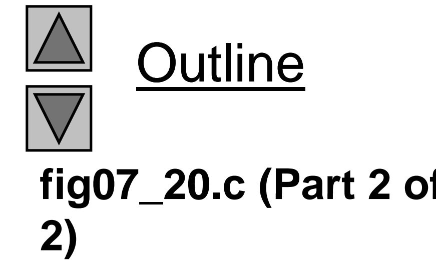
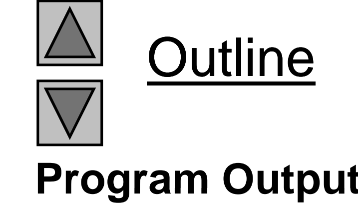


fig07_20.c (Part 2 of 2)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
Array b printed with:
Array subscript notation
b[ 0 ] = 10
b[ 1 ] = 20
b[ 2 ] = 30
b[ 3 ] = 40

Pointer/offset notation where
the pointer is the array name
*( b + 0 ) = 10
*( b + 1 ) = 20
*( b + 2 ) = 30
*( b + 3 ) = 40

Pointer subscript notation
bPtr[ 0 ] = 10
bPtr[ 1 ] = 20
bPtr[ 2 ] = 30
bPtr[ 3 ] = 40

Pointer/offset notation
*( bPtr + 0 ) = 10
*( bPtr + 1 ) = 20
*( bPtr + 2 ) = 30
*( bPtr + 3 ) = 40
```



Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

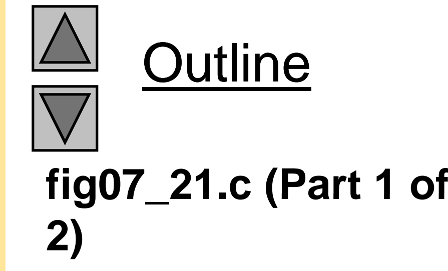


fig07_21.c (Part 1 of 2)

```
1  /* Fig. 7.21: fig07_21.c
2     Copying a string using array notation and pointer notation. */
3  #include <stdio.h>
4
5  void copy1( char *s1, const char *s2 ); /* prototype */
6  void copy2( char *s1, const char *s2 ); /* prototype */
7
8  int main()
9  {
10    char string1[ 10 ];          /* create array string1 */
11    char *string2 = "Hello";     /* create a pointer to a string */
12    char string3[ 10 ];          /* create array string3 */
13    char string4[] = "Good Bye"; /* create a pointer to a string */
14
15    copy1( string1, string2 );
16    printf( "string1 = %s\n", string1 );
17
18    copy2( string3, string4 );
19    printf( "string3 = %s\n", string3 );
20
21    return 0; /* indicates successful termination */
22
23 } /* end main */
24
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
25 /* copy s2 to s1 using array notation */
26 void copy1( char *s1, const char *s2 )
27 {
28    int i; /* counter */
29
30    /* loop through strings */
31    for ( i = 0; ( s1[ i ] = s2[ i ] ) != '\0'; i++ ) {
32       ; /* do nothing in body */
33    } /* end for */
34
35 } /* end function copy1 */
36
37 /* copy s2 to s1 using pointer notation */
38 void copy2( char *s1, const char *s2 )
39 {
40    /* loop through strings */
41    for ( ; ( *s1 = *s2 ) != '\0'; s1++, s2++ ) {
42       ; /* do nothing in body */
43    } /* end for */
44
45 } /* end function copy2 */
```

```
string1 = Hello
string3 = Good Bye
```

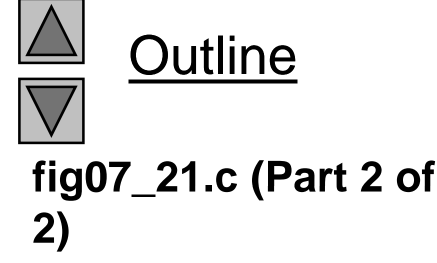


fig07_21.c (Part 2 of 2)

Program Output

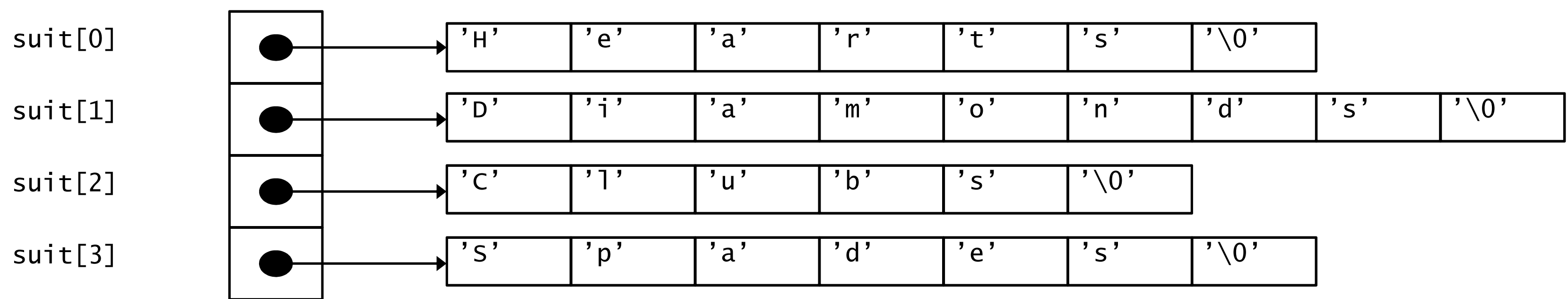
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

7.9 Arrays of Pointers

- Arrays can contain pointers
- For example: an array of strings

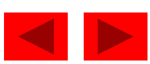
```
char *suit[ 4 ] = { "Hearts", "Diamonds",
  "Clubs", "Spades" };
```

  - Strings are pointers to the first character
  - `char *` – each element of `suit` is a pointer to a `char`
  - The strings are not actually stored in the array `suit`, only pointers to the strings are stored



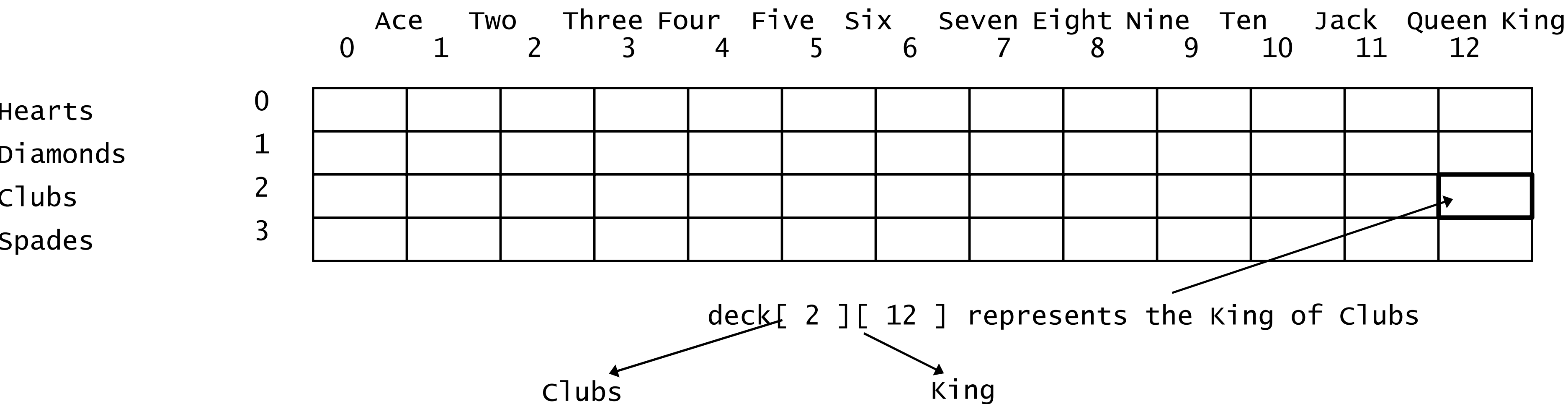
  - `suit` array has a fixed size, but strings can be of any size

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



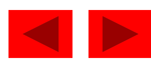
7.10 Case Study: A Card Shuffling and Dealing Simulation

- Card shuffling program
  - Use array of pointers to strings
  - Use double scripted array (suit, face)



  - The numbers 1-52 go into the array
    - Representing the order in which the cards are dealt

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



7.10 Case Study: A Card Shuffling and Dealing Simulation

- Pseudocode
  - Top level:

    Shuffle and deal 52 cards
  - First refinement:

    Initialize the suit array

    Initialize the face array

    Initialize the deck array

    Shuffle the deck

    Deal 52 cards

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

7.10 Case Study: A Card Shuffling and Dealing Simulation

- Second refinement
  - Convert *shuffle the deck* to

    For each of the 52 cards

    Place card number in randomly selected unoccupied slot of deck
  - Convert *deal 52 cards* to

    For each of the 52 cards

    Find card number in deck array and print face and suit of card

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

7.10 Case Study: A Card Shuffling and Dealing Simulation

- Third refinement
  - Convert *shuffle the deck* to

    Choose slot of deck randomly

    While chosen slot of deck has been previously chosen

    Choose slot of deck randomly

    Place card number in chosen slot of deck

  - Convert *deal 52 cards* to

    For each slot of the deck array

    If slot contains card number

    Print the face and suit of the card

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
1  /* Fig. 7.24: fig07_24.c
2     Card shuffling dealing program */
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include <time.h>
6
7  /* prototypes */
8  void shuffle( int wDeck[][ 13 ] );
9  void deal( const int wDeck[][ 13 ], const char *wFace[],
10            const char *wSuit[] );
11
12 int main()
13 {
14    /* initialize suit array */
15    const char *suit[ 4 ] = { "Hearts", "Diamonds", "Clubs", "Spades" };
16
17    /* initialize face array */
18    const char *face[ 13 ] =
19       { "Ace", "Deuce", "Three", "Four",
20         "Five", "Six", "Seven", "Eight",
21         "Nine", "Ten", "Jack", "Queen", "King" };
22
23    /* initialize deck array */
24    int deck[ 4 ][ 13 ] = { 0 };
25
```

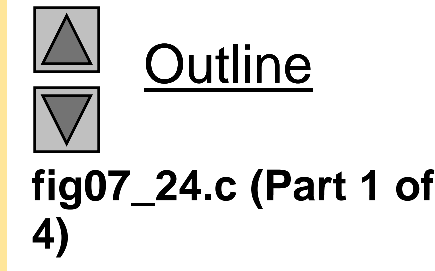


fig07_24.c (Part 1 of 4)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
26    srand( time( 0 ) ); /* seed random-number generator */
27
28    shuffle( deck );
29    deal( deck, face, suit );
30
31    return 0; /* indicates successful termination */
32
33 } /* end main */
34
35 /* shuffle cards in deck */
36 void shuffle( int wDeck[][ 13 ] )
37 {
38    int row;    /* row number */
39    int column; /* column number */
40    int card;   /* counter */
41
42    /* for each of the 52 cards, choose slot of deck randomly */
43    for ( card = 1; card <= 52; card++ ) {
44
45       /* choose new random location until unoccupied slot found */
46       do {
47          row = rand() % 4;
48          column = rand() % 13;
49       } while( wDeck[ row ][ column ] != 0 ); /* end do...while */
50
```

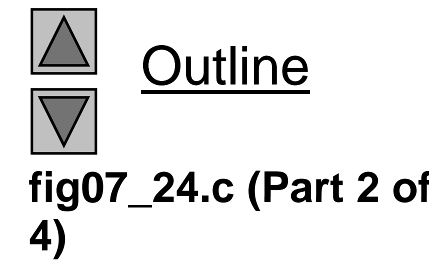


fig07_24.c (Part 2 of 4)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
51       /* place card number in chosen slot of deck */
52       wDeck[ row ][ column ] = card;
53    } /* end for */
54
55 } /* end function shuffle */
56
57 /* deal cards in deck */
58 void deal( const int wDeck[][ 13 ], const char *wFace[],
59            const char *wSuit[] )
60 {
61    int card;   /* card counter */
62    int row;    /* row counter */
63    int column; /* column counter */
64
65    /* deal each of the 52 cards */
66    for ( card = 1; card <= 52; card++ ) {
67
68       /* loop through rows of wDeck */
69       for ( row = 0; row <= 3; row++ ) {
70
71          /* loop through columns of wDeck for current row */
72          for ( column = 0; column <= 12; column++ ) {
73
74             /* if slot contains current card, display card */
75             if ( wDeck[ row ][ column ] == card ) {
```

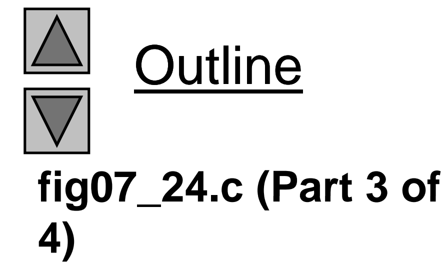


fig07_24.c (Part 3 of 4)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
76                 printf( "%5s of %-8s%c", wFace[ column ], wSuit[ row ],
77                         card % 2 == 0 ? '\n' : '\t' );
78              } /* end if */
79
80           } /* end for */
81
82        } /* end for */
83
84     } /* end for */
85
86  } /* end function deal */
```

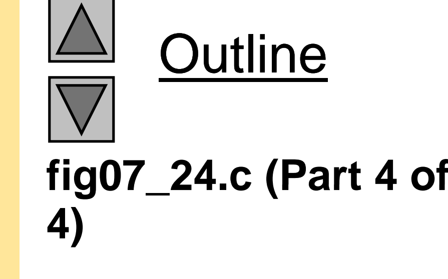
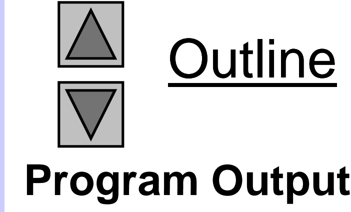


fig07_24.c (Part 4 of 4)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
Nine of Hearts          Five of Clubs
Queen of Spades         Three of Spades
Queen of Hearts           Ace of Clubs
 King of Hearts           Six of Spades
 Jack of Diamonds        Five of Spades
Seven of Hearts          King of Clubs
Three of Clubs          Eight of Hearts
Three of Diamonds        Four of Diamonds
Queen of Diamonds        Five of Diamonds
  Six of Diamonds        Five of Hearts
  Ace of Spades           Six of Hearts
 Nine of Diamonds       Queen of Clubs
Eight of Spades          Nine of Clubs
Deuce of Clubs            Six of Clubs
Deuce of Spades          Jack of Clubs
 Four of Clubs          Eight of Clubs
 Four of Spades         Seven of Spades
Seven of Diamonds       Seven of Clubs
 King of Spades           Ten of Diamonds
 Jack of Hearts           Ace of Hearts
 Jack of Spades           Ten of Clubs
Eight of Diamonds       Deuce of Diamonds
  Ace of Diamonds        Nine of Spades
 Four of Hearts         Deuce of Hearts
 King of Diamonds         Ten of Spades
Three of Hearts           Ten of Hearts
```



Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

7.11 Pointers to Functions

- Pointer to function
  - Contains address of function
  - Similar to how array name is address of first element
  - Function name is starting address of code that defines function
- Function pointers can be
  - Passed to functions
  - Stored in arrays
  - Assigned to other function pointers

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

7.11 Pointers to Functions

- Example: bubblesort
  - Function `bubble` takes a function pointer
    - `bubble` calls this helper function
    - this determines ascending or descending sorting
  - The argument in `bubblesort` for the function pointer:

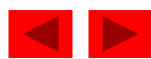
```
int ( *compare )( int a, int b )
```

  tells `bubblesort` to expect a pointer to a function that takes two `int`s and returns an `int`
  - If the parentheses were left out:

```
int *compare( int a, int b )
```

    - Defines a function that receives two integers and returns a pointer to a `int`

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



```
1  /* Fig. 7.26: fig07_26.c
2     Multipurpose sorting program using function pointers */
3  #include <stdio.h>
4  #define SIZE 10
5
6  /* prototypes */
7  void bubble( int work[], const int size, int (*compare)( int a, int b ) );
8  int ascending( int a, int b );
9  int descending( int a, int b );
10
11 int main()
12 {
13    int order;   /* 1 for ascending order or 2 for descending order */
14    int counter; /* counter */
15
16    /* initialize array a */
17    int a[ SIZE ] = { 2, 6, 4, 8, 10, 12, 89, 68, 45, 37 };
18
19    printf( "Enter 1 to sort in ascending order,\n"
20            "Enter 2 to sort in descending order: " );
21    scanf( "%d",  &order );
22
23    printf( "\nData items in original order\n" );
24
```

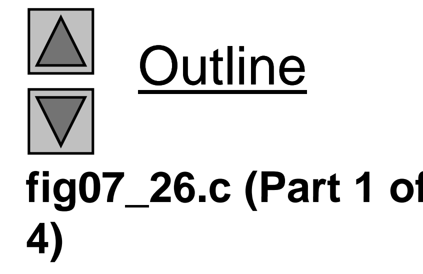


fig07_26.c (Part 1 of 4)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
25    /* output original array */
26    for ( counter = 0; counter < SIZE; counter++ ) {
27       printf( "%5d", a[ counter ] );
28    } /* end for */
29
30    /* sort array in ascending order; pass function ascending as an
31      argument to specify ascending sorting order */
32    if ( order == 1 ) {
33       bubble( a, SIZE, ascending );
34       printf( "\nData items in ascending order\n" );
35    } /* end if */
36    else { /* pass function descending */
37       bubble( a, SIZE, descending );
38       printf( "\nData items in descending order\n" );
39    } /* end else */
40
41    /* output sorted array */
42    for ( counter = 0; counter < SIZE; counter++ ) {
43       printf( "%5d", a[ counter ] );
44    } /* end for */
45
46    printf( "\n" );
47
48    return 0; /* indicates successful termination */
49
50 } /* end main */
51
```

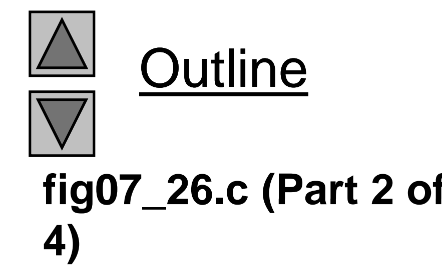


fig07_26.c (Part 2 of 4)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```c
52 /* multipurpose bubble sort; parameter compare is a pointer to
53    the comparison function that determines sorting order */
54 void bubble( int work[], const int size, int (*compare)( int a, int b ) )
55 {
56    int pass;  /* pass counter */
57    int count; /* comparison counter */
58
59    void swap( int *element1Ptr, int *element2ptr ); /* prototype */
60
61    /* loop to control passes */
62    for ( pass = 1; pass < size; pass++ ) {
63
64       /* loop to control number of comparisons per pass */
65       for ( count = 0; count < size - 1; count++ ) {
66
67          /* if adjacent elements are out of order, swap them */
68          if ( (*compare)( work[ count ], work[ count + 1 ] ) ) {
69             swap( &work[ count ], &work[ count + 1 ] );
70          } /* end if */
71
72       } /* end for */
73
74    } /* end for */
75
76 } /* end function bubble */
77
```

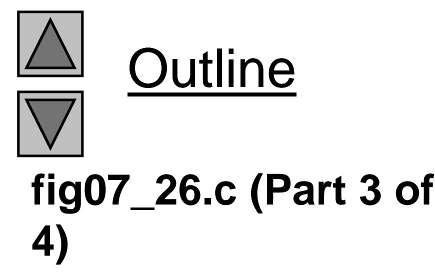


fig07_26.c (Part 3 of 4)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

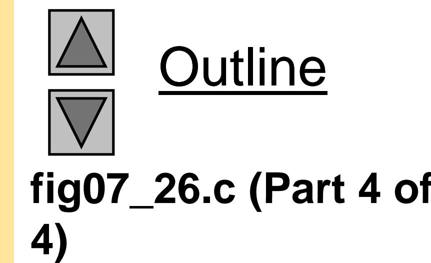


fig07_26.c (Part 4 of 4)

```
78 /* swap values at memory locations to which element1Ptr and
79    element2Ptr point */
80 void swap( int *element1Ptr, int *element2Ptr )
81 {
82    int hold; /* temporary holding variable */
83
84    hold = *element1Ptr;
85    *element1Ptr = *element2Ptr;
86    *element2Ptr = hold;
87 } /* end function swap */
88
89 /* determine whether elements are out of order for an ascending
90    order sort */
91 int ascending( int a, int b )
92 {
93    return b < a; /* swap if b is less than a */
94
95 } /* end function ascending */
96
97 /* determine whether elements are out of order for a descending
98    order sort */
99 int descending( int a, int b )
100 {
101    return b > a; /* swap if b is greater than a */
102
103 } /* end function descending */
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
Enter 1 to sort in ascending order,
Enter 2 to sort in descending order: 1

Data items in original order
    2    6    4    8   10   12   89   68   45   37
Data items in ascending order
    2    4    6    8   10   12   37   45   68   89
```

```
Enter 1 to sort in ascending order,
Enter 2 to sort in descending order: 2

Data items in original order
    2    6    4    8   10   12   89   68   45   37
Data items in descending order
   89   68   45   37   12   10    8    6    4    2
```

Outline

Program Output

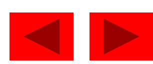
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Chapter 8 - Characters and Strings

Outline

8.1 Introduction
8.2 Fundamentals of Strings and Characters
8.3 Character Handling Library
8.4 String Conversion Functions
8.5 Standard Input/Output Library Functions
8.6 String Manipulation Functions of the String Handling Library
8.7 Comparison Functions of the String Handling Library
8.8 Search Functions of the String Handling Library
8.9 Memory Functions of the String Handling Library
8.10 Other Functions of the String Handling Library

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Objectives

- In this chapter, you will learn:
  - To be able to use the functions of the character handling library (`ctype`).
  - To be able to use the string and character input/output functions of the standard input/output library (`stdio`).
  - To be able to use the string conversion functions of the general utilities library (`stdlib`).
  - To be able to use the string processing functions of the string handling library (`string`).
  - To appreciate the power of function libraries as a means of achieving software reusability.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

8.1 Introduction

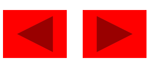
- Introduce some standard library functions
  - Easy string and character processing
  - Programs can process characters, strings, lines of text, and blocks of memory
- These techniques used to make
  - Word processors
  - Page layout software
  - Typesetting programs

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

8.2 Fundamentals of Strings and Characters

- Characters
  - Building blocks of programs
    - Every program is a sequence of meaningfully grouped characters
  - Character constant
    - An `int` value represented as a character in single quotes
    - `'z'` represents the integer value of `z`
- Strings
  - Series of characters treated as a single unit
    - Can include letters, digits and special characters (`*`, `/`, `$`)
  - String literal (string constant) - written in double quotes
    - `"Hello"`
  - Strings are arrays of characters
    - String a pointer to first character
    - Value of string is the address of first character

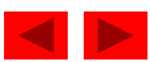
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



8.2 Fundamentals of Strings and Characters

- String definitions
  - Define as a character array or a variable of type `char *`
```
char color[] = "blue";
char *colorPtr = "blue";
```
  - Remember that strings represented as character arrays end with `'\0'`
    - `color` has `5 elements`
- Inputting strings
  - Use `scanf`
```
scanf("%s", word);
```
    - Copies input into `word[]`
    - Do not need & (because a string is a pointer)
  - Remember to leave room in the array for `'\0'`

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



8.3 Character Handling Library

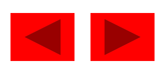
- Character handling library
  - Includes functions to perform useful tests and manipulations of character data
  - Each function receives a character (an `int`) or `EOF` as an argument
- The following slide contains a table of all the functions in `<ctype.h>`

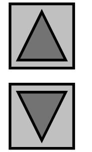
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

8.3 Character Handling Library

Prototype	Description
`int isdigit( int c );`	Returns `true` if `c` is a digit and `false` otherwise.
`int isalpha( int c );`	Returns `true` if `c` is a letter and `false` otherwise.
`int isalnum( int c );`	Returns `true` if `c` is a digit or a letter and `false` otherwise.
`int isxdigit( int c );`	Returns `true` if `c` is a hexadecimal digit character and `false` otherwise.
`int islower( int c );`	Returns `true` if `c` is a lowercase letter and `false` otherwise.
`int isupper( int c );`	Returns `true` if `c` is an uppercase letter; `false` otherwise.
`int tolower( int c );`	If `c` is an uppercase letter, `tolower` returns `c` as a lowercase letter. Otherwise, `tolower` returns the argument unchanged.
`int toupper( int c );`	If `c` is a lowercase letter, `toupper` returns `c` as an uppercase letter. Otherwise, `toupper` returns the argument unchanged.
`int isspace( int c );`	Returns `true` if `c` is a white-space character—newline (`'\n'`), space (`' '`), form feed (`'\f'`), carriage return (`'\r'`), horizontal tab (`'\t'`), or vertical tab (`'\v'`)—and `false` otherwise
`int iscntrl( int c );`	Returns `true` if `c` is a control character and `false` otherwise.
`int ispunct( int c );`	Returns `true` if `c` is a printing character other than a space, a digit, or a letter and `false` otherwise.
`int isprint( int c );`	Returns `true` value if `c` is a printing character including space (`' '`) and `false` otherwise.
`int isgraph( int c );`	Returns `true` if `c` is a printing character other than space (`' '`) and `false` otherwise.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.





Outline

fig08_02.c (Part 1 of 2)

```
1  /* Fig. 8.2: fig08_02.c
2     Using functions isdigit, isalpha, isalnum, and isxdigit */
3  #include <stdio.h>
4  #include <ctype.h>
5
6  int main()
7  {
8     printf( "%s\n%s%s\n%s%s\n\n", "According to isdigit: ",
9        isdigit( '8' ) ? "8 is a " : "8 is not a ", "digit",
10       isdigit( '#' ) ? "# is a " : "# is not a ", "digit" );
11
12    printf( "%s\n%s%s\n%s%s\n%s%s\n%s%s\n\n",
13       "According to isalpha:",
14       isalpha( 'A' ) ? "A is a " : "A is not a ", "letter",
15       isalpha( 'b' ) ? "b is a " : "b is not a ", "letter",
16       isalpha( '&' ) ? "& is a " : "& is not a ", "letter",
17       isalpha( '4' ) ? "4 is a " : "4 is not a ", "letter" );
18
19    printf( "%s\n%s%s\n%s%s\n%s%s\n\n",
20       "According to isalnum:",
21       isalnum( 'A' ) ? "A is a " : "A is not a ",
22       "digit or a letter",
23       isalnum( '8' ) ? "8 is a " : "8 is not a ",
24       "digit or a letter",
25       isalnum( '#' ) ? "# is a " : "# is not a ",
26       "digit or a letter" );
27
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
28    printf( "%s\n%s%s\n%s%s\n%s%s\n%s%s\n%s%s\n",
29        "According to isxdigit:",
30        isxdigit( 'F' ) ? "F is a " : "F is not a ",
31        "hexadecimal digit",
32        isxdigit( 'J' ) ? "J is a " : "J is not a ",
33        "hexadecimal digit",
34        isxdigit( '7' ) ? "7 is a " : "7 is not a ",
35        "hexadecimal digit",
36        isxdigit( '$' ) ? "$ is a " : "$ is not a ",
37        "hexadecimal digit",
38        isxdigit( 'f' ) ? "f is a " : "f is not a ",
39        "hexadecimal digit" );
40
41    return 0; /* indicates successful termination */
42
43 } /* end main */
```

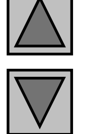
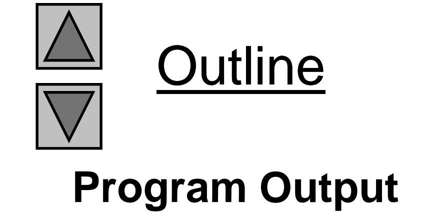


fig08_02.c (Part 2 of 2)

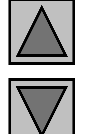
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Program Output

```
According to isdigit:
8 is a digit
# is not a digit

According to isalpha:
A is a letter
b is a letter
& is not a letter
4 is not a letter

According to isalnum:
A is a digit or a letter
8 is a digit or a letter
# is not a digit or a letter

According to isxdigit:
F is a hexadecimal digit
J is not a hexadecimal digit
7 is a hexadecimal digit
$ is not a hexadecimal digit
f is a hexadecimal digit
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Outline

fig08_03.c (Part 1 of 2)

```
1  /* Fig. 8.3: fig08_03.c
2     Using functions islower, isupper, tolower, toupper */
3  #include <stdio.h>
4  #include <ctype.h>
5
6  int main()
7  {
8     printf( "%s\n%s%s\n%s%s\n%s%s\n%s%s\n\n",
9             "According to islower:",
10            islower( 'p' ) ? "p is a " : "p is not a ",
11            "lowercase letter",
12            islower( 'P' ) ? "P is a " : "P is not a ",
13            "lowercase letter",
14            islower( '5' ) ? "5 is a " : "5 is not a ",
15            "lowercase letter",
16            islower( '!' ) ? "! is a " : "! is not a ",
17            "lowercase letter" );
18
19     printf( "%s\n%s%s\n%s%s\n%s%s\n%s%s\n\n",
20            "According to isupper:",
21            isupper( 'D' ) ? "D is an " : "D is not an ",
22            "uppercase letter",
23            isupper( 'd' ) ? "d is an " : "d is not an ",
24            "uppercase letter",
25            isupper( '8' ) ? "8 is an " : "8 is not an ",
26            "uppercase letter",
27            isupper( '$' ) ? "$ is an " : "$ is not an ",
28            "uppercase letter" );
29
```

Outline

fig08_03.c (Part 2 of 2)

```
30    printf( "%s%c\n%s%c\n%s%c\n%s%c\n",
31            "u converted to uppercase is ", toupper( 'u' ),
32            "7 converted to uppercase is ", toupper( '7' ),
33            "$ converted to uppercase is ", toupper( '$' ),
34            "L converted to lowercase is ", tolower( 'L' ) );
35
36    return 0; /* indicates successful termination */
37
38 } /* end main */
```

Program Output

```
According to islower:
p is a lowercase letter
P is not a lowercase letter
5 is not a lowercase letter
! is not a lowercase letter

According to isupper:
D is an uppercase letter
d is not an uppercase letter
8 is not an uppercase letter
$ is not an uppercase letter

u converted to uppercase is U
7 converted to uppercase is 7
$ converted to uppercase is $
L converted to lowercase is l
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

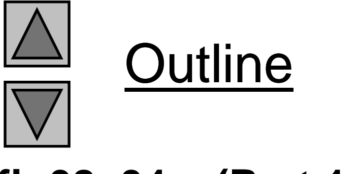


fig08_04.c (Part 1 of 2)

```
1  /* Fig. 8.4: fig08_04.c
2    Using functions isspace, iscntrl, ispunct, isprint, isgraph */
3  #include <stdio.h>
4  #include <ctype.h>
5
6  int main()
7  {
8     printf( "%s\n%s%s%s\n%s%s%s\n%s%s\n\n",
9         "According to isspace:",
10        "Newline", isspace( '\n' ) ? " is a " : " is not a ",
11        "whitespace character", "Horizontal tab",
12        isspace( '\t' ) ? " is a " : " is not a ",
13        "whitespace character",
14        isspace( '%' ) ? "% is a " : "% is not a ",
15        "whitespace character" );
16
17    printf( "%s\n%s%s%s\n%s%s\n\n", "According to iscntrl:",
18        "Newline", iscntrl( '\n' ) ? " is a " : " is not a ",
19        "control character", iscntrl( '$' ) ? "$ is a " :
20        "$ is not a ", "control character" );
21
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

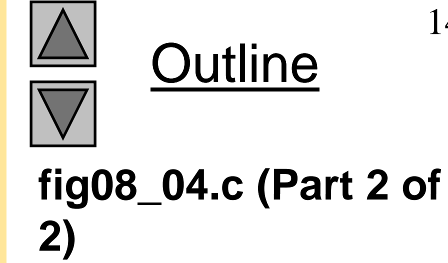
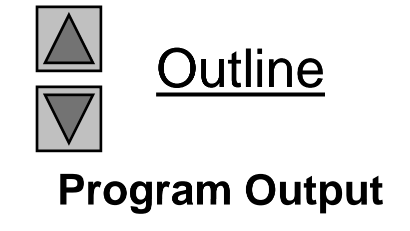


fig08_04.c (Part 2 of 2)

```
22    printf( "%s\n%s%s\n%s%s\n%s%s\n\n",
23       "According to ispunct:",
24       ispunct( ';' ) ? "; is a " : "; is not a ",
25       "punctuation character",
26       ispunct( 'Y' ) ? "Y is a " : "Y is not a ",
27       "punctuation character",
28       ispunct( '#' ) ? "# is a " : "# is not a ",
29       "punctuation character" );
30
31    printf( "%s\n%s%s\n%s%s%s\n\n", "According to isprint:",
32       isprint( '$' ) ? "$ is a " : "$ is not a ",
33       "printing character",
34       "Alert", isprint( '\a' ) ? " is a " : " is not a ",
35       "printing character" );
36
37    printf( "%s\n%s%s\n%s%s%s\n",  "According to isgraph:",
38       isgraph( 'Q' ) ? "Q is a " : "Q is not a ",
39       "printing character other than a space",
40       "Space", isgraph( ' ' ) ? " is a " : " is not a ",
41       "printing character other than a space" );
42
43    return 0; /* indicates successful termination */
44
45 } /* end main */
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Program Output

```
According to isspace:
Newline is a whitespace character
Horizontal tab is a whitespace character
% is not a whitespace character

According to iscntrl:
Newline is a control character
$ is not a control character

According to ispunct:
; is a punctuation character
Y is not a punctuation character
# is a punctuation character

According to isprint:
$ is a printing character
Alert is not a printing character

According to isgraph:
Q is a printing character other than a space
Space is not a printing character other than a space
```

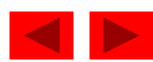
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

8.4 String Conversion Functions

- Conversion functions
  - In <stdlib.h> (general utilities library)
- Convert strings of digits to integer and floating-point values

Function prototype	Function description
`double atof( const char *nPtr );`	Converts the string `nPtr` to `double`.
`int atoi( const char *nPtr );`	Converts the string `nPtr` to `int`.
`long atol( const char *nPtr );`	Converts the string `nPtr` to `long int`.
`double strtod( const char *nPtr, char **endPtr );`	Converts the string `nPtr` to `double`.
`long strtol( const char *nPtr, char **endPtr, int base );`	Converts the string `nPtr` to `long`.
`unsigned long strtoul( const char *nPtr, char **endPtr, int base );`	Converts the string `nPtr` to `unsigned long`.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Outline

fig 08_06.c

```
1  /* Fig. 8.6: fig08_06.c
2     Using atof */
3  #include <stdio.h>
4  #include <stdlib.h>
5
6  int main()
7  {
8     double d; /* variable to hold converted string */
9
10    d = atof( "99.0" );
11
12    printf( "%s%.3f\n%s%.3f\n",
13            "The string \"99.0\" converted to double is ", d,
14            "The converted value divided by 2 is ",
15            d / 2.0 );
16
17    return 0; /* indicates successful termination */
18
19 } /* end main */
```

Program Output

```
The string "99.0" converted to double is 99.000
The converted value divided by 2 is 49.500
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

fig08_07.c

```
1  /* Fig. 8.7: fig08_07.c
2     Using atoi */
3  #include <stdio.h>
4  #include <stdlib.h>
5
6  int main()
7  {
8     int i; /* variable to hold converted string */
9
10    i = atoi( "2593" );
11
12    printf( "%s%d\n%s%d\n",
13            "The string \"2593\" converted to int is ", i,
14            "The converted value minus 593 is ", i - 593 );
15
16    return 0; /* indicates successful termination */
17
18 } /* end main */
```

Program Output

```
The string "2593" converted to int is 2593
The converted value minus 593 is 2000
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

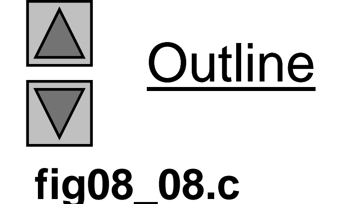


fig08_08.c

```
1  /* Fig. 8.8: fig08_08.c
2     Using atol */
3  #include <stdio.h>
4  #include <stdlib.h>
5
6  int main()
7  {
8     long l; /* variable to hold converted string */
9
10    l = atol( "1000000" );
11
12    printf( "%s%ld\n%s%ld\n",
13            "The string \"1000000\" converted to long int is ", l,
14            "The converted value divided by 2 is ", l / 2 );
15
16    return 0; /* indicates successful termination */
17
18 } /* end main */
```

Program Output

```
The string "1000000" converted to long int is 1000000
The converted value divided by 2 is 500000
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Outline

fig08_09.c

```
1  /* Fig. 8.9: fig08_09.c
2     Using strtod */
3  #include <stdio.h>
4  #include <stdlib.h>
5
6  int main()
7  {
8     /* initialize string pointer */
9     const char *string = "51.2% are admitted";
10
11     double d;        /* variable to hold converted sequence */
12     char *stringPtr; /* create char pointer */
13
14     d = strtod( string, &stringPtr );
15
16     printf( "The string \"%s\" is converted to the\n", string );
17     printf( "double value %.2f and the string \"%s\"\n", d, stringPtr );
18
19     return 0; /* indicates successful termination */
20
21  } /* end main */
```

Program Output

```
The string "51.2% are admitted" is converted to the
double value 51.20 and the string "% are admitted"
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

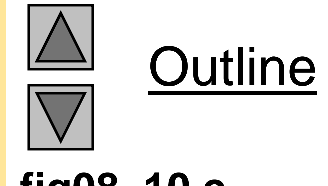


fig08_10.c

```
1  /* Fig. 8.10: fig08_10.c
2     Using strtol */
3  #include <stdio.h>
4  #include <stdlib.h>
5
6  int main()
7  {
8     const char *string = "-1234567abc"; /* initialize string pointer */
9
10    char *remainderPtr; /* create char pointer */
11    long x;             /* variable to hold converted sequence */
12
13    x = strtol( string, &remainderPtr, 0 );
14
15    printf( "%s\"%s\"\n%s%ld\n%s\"%s\"\n%s%ld\n",
16            "The original string is ", string,
17            "The converted value is ", x,
18            "The remainder of the original string is ",
19            remainderPtr,
20            "The converted value plus 567 is ", x + 567 );
21
22    return 0; /* indicates successful termination */
23
24 } /* end main */
```

Program Output

```
The original string is "-1234567abc"
The converted value is -1234567
The remainder of the original string is "abc"
The converted value plus 567 is -1234000
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

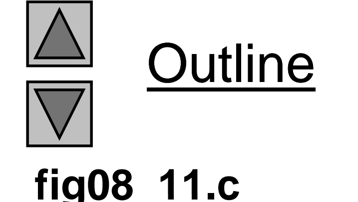


fig08_11.c

```
1  /* Fig. 8.11: fig08_11.c
2     Using strtoul */
3  #include <stdio.h>
4  #include <stdlib.h>
5
6  int main()
7  {
8     const char *string = "1234567abc"; /* initialize string pointer */
9     unsigned long x;    /* variable to hold converted sequence */
10    char *remainderPtr; /* create char pointer */
11
12    x = strtoul( string, &remainderPtr, 0 );
13
14    printf( "%s\"%s\"\n%s%lu\n%s\"%s\"\n%s%lu\n",
15            "The original string is ", string,
16            "The converted value is ", x,
17            "The remainder of the original string is ",
18            remainderPtr,
19            "The converted value minus 567 is ", x - 567 );
20
21    return 0; /* indicates successful termination */
22
23 } /* end main */
```

Program Output

```
The original string is "1234567abc"
The converted value is 1234567
The remainder of the original string is "abc"
The converted value minus 567 is 1234000
```

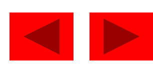
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

8.5 Standard Input/Output Library Functions

- Functions in <stdio.h>
- Used to manipulate character and string data

Function prototype	Function description
`int getchar( void );`	Inputs the next character from the standard input and returns it as an integer.
`char *gets( char *s );`	Inputs characters from the standard input into the array `s` until a newline or end-of-file character is encountered. A terminating null character is appended to the array.
`int putchar( int c );`	Prints the character stored in `c`.
`int puts( const char *s );`	Prints the string `s` followed by a newline character.
`int sprintf( char *s, const char *format, ... );`	Equivalent to `printf`, except the output is stored in the array `s` instead of printing it on the screen.
`int sscanf( char *s, const char *format, ... );`	Equivalent to `scanf`, except the input is read from the array `s` instead of reading it from the keyboard.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



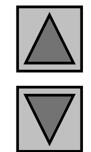


fig08_13.c (Part 1 of 2)

```
1  /* Fig. 8.13: fig08_13.c
2     Using gets and putchar */
3  #include <stdio.h>
4
5  int main()
6  {
7     char sentence[ 80 ]; /* create char array */
8
9     void reverse( const char * const sPtr ); /* prototype */
10
11     printf( "Enter a line of text:\n" );
12
13     /* use gets to read line of text */
14     gets( sentence );
15
16     printf( "\nThe line printed backwards is:\n" );
17     reverse( sentence );
18
19     return 0; /* indicates successful termination */
20
21  } /* end main */
22
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Outline

fig08_13.c (Part 1 of 2)

```
23 /* recursively outputs characters in string in reverse order */
24 void reverse( const char * const sPtr )
25 {
26    /* if end of the string */
27    if ( sPtr[ 0 ] == '\0' ) {
28       return;
29    } /* end if */
30    else { /* if not end of the string */
31       reverse( &sPtr[ 1 ] );
32
33       putchar( sPtr[ 0 ] ); /* use putchar to display character */
34    } /* end else */
35
36 } /* end function reverse */
```

Program Output

```
Enter a line of text:
Characters and Strings

The line printed backwards is:
sgnirtS dna sretcarahC

Enter a line of text:
able was I ere I saw elba

The line printed backwards is:
able was I ere I saw elba
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

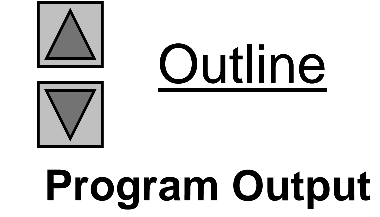
Outline

fig8_14.c

```
1  /* Fig. 8.14: fig08_14.c
2     Using getchar and puts */
3  #include <stdio.h>
4
5  int main()
6  {
7     char c;              /* variable to hold character input by user */
8     char sentence[ 80 ]; /* create char array */
9     int i = 0;           /* initialize counter i */
10
11    /* prompt user to enter line of text */
12    puts( "Enter a line of text:" );
13
14    /* use getchar to read each character */
15    while ( ( c = getchar() ) != '\n') {
16       sentence[ i++ ] = c;
17    } /* end while */
18
19    sentence[ i ] = '\0';
20
21    /* use puts to display sentence */
22    puts( "\nThe line entered was:" );
23    puts( sentence );
24
25    return 0; /* indicates successful termination */
26
27 } /* end main */
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
Enter a line of text:
This is a test.

The line entered was:
This is a test.
```



Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

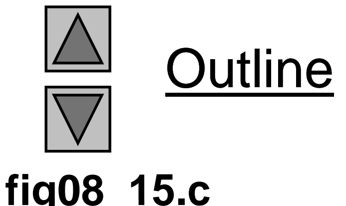


fig08_15.c

```
1  /* Fig. 8.15: fig08_15.c
2     Using sprintf */
3  #include <stdio.h>
4
5  int main()
6  {
7     char s[ 80 ]; /* create char array */
8     int x;        /* define x */
9     double y;     /* define y */
10
11    printf( "Enter an integer and a double:\n" );
12    scanf( "%d%lf", &x, &y );
13
14    sprintf( s, "integer:%6d\ndouble:%8.2f", x, y );
15
16    printf( "%s\n%s\n",
17            "The formatted output stored in array s is:", s );
18
19    return 0; /* indicates successful termination */
20
21 } /* end main */
```

Program Output

```
Enter an integer and a double:
298 87.375
The formatted output stored in array s is:
integer:   298
double:   87.38
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Outline

fig08_16.c

```
1  /* Fig. 8.16: fig08_16.c
2     Using sscanf */
3  #include <stdio.h>
4
5  int main()
6  {
7     char s[] = "31298 87.375"; /* initialize array s */
8     int x;                     /* define x */
9     double y;                  /* define y */
10
11     sscanf( s, "%d%lf", &x, &y );
12
13     printf( "%s\n%s%6d\n%s%8.3f\n",
14             "The values stored in character array s are:",
15             "integer:", x, "double:", y );
16
17     return 0; /* indicates successful termination */
18
19 } /* end main */
```

Program Output

```
The values stored in character array s are:
integer: 31298
double:  87.375
```

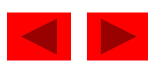
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

8.6 String Manipulation Functions of the String Handling Library

- String handling library has functions to
  - Manipulate string data
  - Search strings
  - Tokenize strings
  - Determine string length

Function prototype	Function description
`char *strcpy( char *s1, const char *s2 )`	Copies string `s2` into array `s1`. The value of `s1` is returned.
`char *strncpy( char *s1, const char *s2, size_t n )`	Copies at most `n` characters of string `s2` into array `s1`. The value of `s1` is returned.
`char *strcat( char *s1, const char *s2 )`	Appends string `s2` to array `s1`. The first character of `s2` overwrites the terminating null character of `s1`. The value of `s1` is returned.
`char *strncat( char *s1, const char *s2, size_t n )`	Appends at most `n` characters of string `s2` to array `s1`. The first character of `s2` overwrites the terminating null character of `s1`. The value of `s1` is returned.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



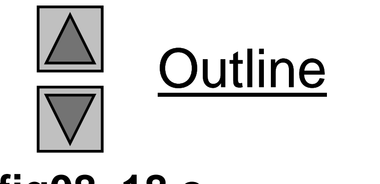


fig08_18.c

```
1  /* Fig. 8.18: fig08_18.c
2     Using strcpy and strncpy */
3  #include <stdio.h>
4  #include <string.h>
5
6  int main()
7  {
8     char x[] = "Happy Birthday to You"; /* initialize char array x */
9     char y[ 25 ];                       /* create char array y */
10     char z[ 15 ];                      /* create char array z */
11
12     /* copy contents of x into y */
13     printf( "%s%s\n%s%s\n",
14             "The string in array x is: ", x,
15             "The string in array y is: ", strcpy( y, x ) );
16
17     /* copy first 14 characters of x into z. Does not copy null
18        character */
19     strncpy( z, x, 14 );
20
21     z[ 14 ] = '\0'; /* append '\0' to z's contents */
22     printf( "The string in array z is: %s\n", z );
23
24     return 0; /* indicates successful termination */
25
```

```
The string in array x is: Happy Birthday to You
The string in array y is: Happy Birthday to You
The string in array z is: Happy Birthday
```

Program Output

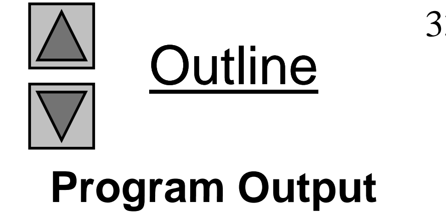
Outline

fig08_19.c

```
1  /* Fig. 8.19: fig08_19.c
2     Using strcat and strncat */
3  #include <stdio.h>
4  #include <string.h>
5
6  int main()
7  {
8     char s1[ 20 ] = "Happy "; /* initialize char array s1 */
9     char s2[] = "New Year ";  /* initialize char array s2 */
10    char s3[ 40 ] = "";       /* initialize char array s3 */
11
12    printf( "s1 = %s\ns2 = %s\n", s1, s2 );
13
14    /* concatenate s2 to s1 */
15    printf( "strcat( s1, s2 ) = %s\n", strcat( s1, s2 ) );
16
17    /* concatenate first 6 characters of s1 to s3. Place '\0'
18       after last character */
19    printf( "strncat( s3, s1, 6 ) = %s\n", strncat( s3, s1, 6 ) );
20
21    /* concatenate s1 to s3 */
22    printf( "strcat( s3, s1 ) = %s\n", strcat( s3, s1 ) );
23
24    return 0; /* indicates successful termination */
25
26 } /* end main */
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
s1 = Happy
s2 = New Year
strcat( s1, s2 ) = Happy New Year
strncat( s3, s1, 6 ) = Happy
strcat( s3, s1 ) = Happy Happy New Year
```



Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

8.7 Comparison Functions of the String Handling Library

- Comparing strings
  - Computer compares numeric ASCII codes of characters in string
  - Appendix D has a list of character codes

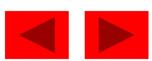
```
int strcmp( const char *s1, const char *s2 );
```

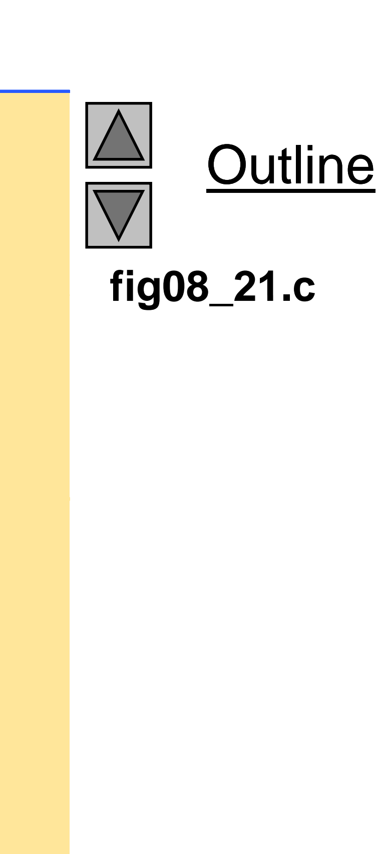
  - Compares string `s1` to `s2`
  - Returns a negative number if `s1 < s2`, zero if `s1 == s2` or a positive number if `s1 > s2`

```
int strncmp( const char *s1, const char *s2,
      size_t n );
```

  - Compares up to `n` characters of string `s1` to `s2`
  - Returns values as above

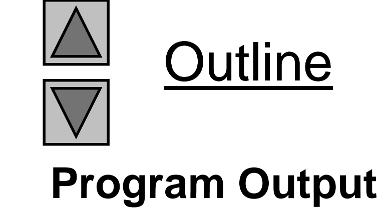
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.






```
1  /* Fig. 8.21: fig08_21.c
2     Using strcmp and strncmp */
3  #include <stdio.h>
4  #include <string.h>
5
6  int main()
7  {
8     const char *s1 = "Happy New Year"; /* initialize char pointer */
9     const char *s2 = "Happy New Year"; /* initialize char pointer */
10    const char *s3 = "Happy Holidays"; /* initialize char pointer */
11
12    printf("%s%s\n%s%s\n%s%s\n\n%s%2d\n%s%2d\n%s%2d\n\n",
13           "s1 = ", s1, "s2 = ", s2, "s3 = ", s3,
14           "strcmp(s1, s2) = ", strcmp( s1, s2 ),
15           "strcmp(s1, s3) = ", strcmp( s1, s3 ),
16           "strcmp(s3, s1) = ", strcmp( s3, s1 ) );
17
18    printf("%s%2d\n%s%2d\n%s%2d\n",
19           "strncmp(s1, s3, 6) = ", strncmp( s1, s3, 6 ),
20           "strncmp(s1, s3, 7) = ", strncmp( s1, s3, 7 ),
21           "strncmp(s3, s1, 7) = ", strncmp( s3, s1, 7 ) );
22
23    return 0; /* indicates successful termination */
24
25 } /* end main */
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Program Output

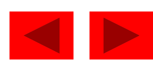
```
s1 = Happy New Year
s2 = Happy New Year
s3 = Happy Holidays

strcmp(s1, s2) =  0
strcmp(s1, s3) =  1
strcmp(s3, s1) = -1

strncmp(s1, s3, 6) =  0
strncmp(s1, s3, 7) =  1
strncmp(s3, s1, 7) = -1
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

8.8 Search Functions of the String Handling Library

Function prototype	Function description
`char *strchr( const char *s, int c );`	Locates the first occurrence of character `c` in string `s`. If `c` is found, a pointer to `c` in `s` is returned. Otherwise, a `NULL` pointer is returned.
`size_t strcspn( const char *s1, const char *s2 );`	Determines and returns the length of the initial segment of string `s1` consisting of characters not contained in string `s2`.
`size_t strspn( const char *s1, const char *s2 );`	Determines and returns the length of the initial segment of string `s1` consisting only of characters contained in string `s2`.
`char *strpbrk( const char *s1, const char *s2 );`	Locates the first occurrence in string `s1` of any character in string `s2`. If a character from string `s2` is found, a pointer to the character in string `s1` is returned. Otherwise, a `NULL` pointer is returned.
`char *strrchr( const char *s, int c );`	Locates the last occurrence of `c` in string `s`. If `c` is found, a pointer to `c` in string `s` is returned. Otherwise, a `NULL` pointer is returned.
`char *strstr( const char *s1, const char *s2 );`	Locates the first occurrence in string `s1` of string `s2`. If the string is found, a pointer to the string in `s1` is returned. Otherwise, a `NULL` pointer is returned.
`char *strtok( char *s1, const char *s2 );`	A sequence of calls to `strtok` breaks string `s1` into "tokens"—logical pieces such as words in a line of text—separated by characters contained in string `s2`. The first call contains `s1` as the first argument, and subsequent calls to continue tokenizing the same string contain `NULL` as the first argument. A pointer to the current token is returned by each call. If there are no more tokens when the function is called, `NULL` is returned.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



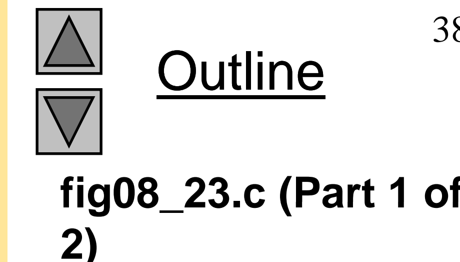


fig08_23.c (Part 1 of 2)

```
1  /* Fig. 8.23: fig08_23.c
2     Using strchr */
3  #include <stdio.h>
4  #include <string.h>
5
6  int main()
7  {
8     const char *string = "This is a test"; /* initialize char pointer */
9     char character1 = 'a';                 /* initialize character1 */
10    char character2 = 'z';                 /* initialize character2 */
11
12    /* if character1 was found in string */
13    if ( strchr( string, character1 ) != NULL ) {
14       printf( "\'%c\' was found in \"%s\".\n",
15               character1, string );
16    } /* end if */
17    else { /* if character1 was not found */
18       printf( "\'%c\' was not found in \"%s\".\n",
19               character1, string );
20    } /* end else */
21
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
22    /* if character2 was found in string */
23    if ( strchr( string, character2 ) != NULL ) {
24       printf( "\'%c\' was found in \"%s\".\n",
25               character2, string );
26    } /* end if */
27    else { /* if character2 was not found */
28       printf( "\'%c\' was not found in \"%s\".\n",
29              character2, string );
30    } /* end else */
31
32    return 0; /* indicates successful termination */
33
34 } /* end main */
```

```
'a' was found in "This is a test".
'z' was not found in "This is a test".
```


fig08_23.c (Part 2 of 2)

Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

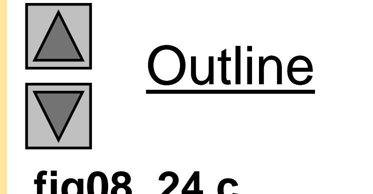


fig08_24.c

```
1  /* Fig. 8.24: fig08_24.c
2     Using strcspn */
3  #include <stdio.h>
4  #include <string.h>
5
6  int main()
7  {
8     /* initialize two char pointers */
9     const char *string1 = "The value is 3.14159";
10    const char *string2 = "1234567890";
11
12    printf( "%s%s\n%s%s\n\n%s\n%s%u",
13            "string1 = ", string1, "string2 = ", string2,
14            "The length of the initial segment of string1",
15            "containing no characters from string2 = ",
16            strcspn( string1, string2 ) );
17
18    return 0; /* indicates successful termination */
19
20 } /* end main */
```

Program Output

```
string1 = The value is 3.14159
string2 = 1234567890

The length of the initial segment of string1
containing no characters from string2 = 13
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

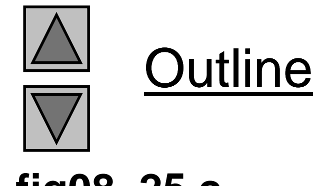


fig08_25.c

```
1  /* Fig. 8.25: fig08_25.c
2     Using strpbrk */
3  #include <stdio.h>
4  #include <string.h>
5
6  int main()
7  {
8     const char *string1 = "This is a test"; /* initialize char pointer */
9     const char *string2 = "beware";         /* initialize char pointer */
10
11     printf( "%s\"%s\"\n'%c'%s\n\"%s\"\n",
12             "Of the characters in ", string2,
13             *strpbrk( string1, string2 ),
14             " is the first character to appear in ", string1 );
15
16     return 0; /* indicates successful termination */
17
18 } /* end main */
```

Program Output

```
Of the characters in "beware"
'a' is the first character to appear in
"This is a test"
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

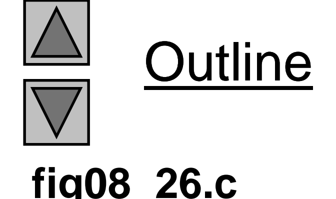


fig08_26.c

```
1  /* Fig. 8.26: fig08_26.c
2     Using strrchr */
3  #include <stdio.h>
4  #include <string.h>
5
6  int main()
7  {
8     /* initialize char pointer */
9     const char *string1 = "A zoo has many animals "
10                          "including zebras";
11    int c = 'z'; /* initialize c */
12
13    printf( "%s\n%s'%c'%s\"%s\"\n",
14            "The remainder of string1 beginning with the",
15            "last occurrence of character ", c,
16            " is: ", strrchr( string1, c ) );
17
18    return 0; /* indicates successful termination */
19
20 } /* end main */
```

Program Output

```
The remainder of string1 beginning with the
last occurrence of character 'z' is: "zebras"
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

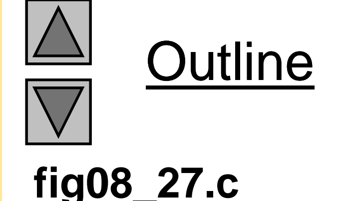


fig08_27.c

```
1  /* Fig. 8.27: fig08_27.c
2     Using strspn */
3  #include <stdio.h>
4  #include <string.h>
5
6  int main()
7  {
8     /* initialize two char pointers */
9     const char *string1 = "The value is 3.14159";
10    const char *string2 = "aehi lsTuv";
11
12    printf( "%s%s\n%s%s\n\n%s\n%s%u\n",
13            "string1 = ", string1, "string2 = ", string2,
14            "The length of the initial segment of string1",
15            "containing only characters from string2 = ",
16            strspn( string1, string2 ) );
17
18    return 0; /* indicates successful termination */
19
20 } /* end main */
```

Program Output

```
string1 = The value is 3.14159
string2 = aehi lsTuv

The length of the initial segment of string1
containing only characters from string2 = 13
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

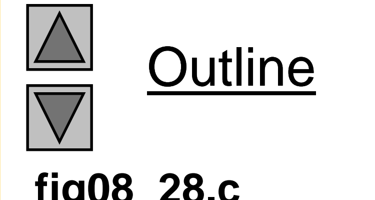


fig08_28.c

```
1  /* Fig. 8.28: fig08_28.c
2     Using strstr */
3  #include <stdio.h>
4  #include <string.h>
5
6  int main()
7  {
8     const char *string1 = "abcdefabcdef"; /* initialize char pointer */
9     const char *string2 = "def";          /* initialize char pointer */
10
11    printf( "%s%s\n%s%s\n\n%s\n%s%s\n",
12            "string1 = ", string1, "string2 = ", string2,
13            "The remainder of string1 beginning with the",
14            "first occurrence of string2 is: ",
15            strstr( string1, string2 ) );
16
17    return 0; /* indicates successful termination */
18
19 } /* end main */
```

Program Output

```
string1 = abcdefabcdef
string2 = def

The remainder of string1 beginning with the
first occurrence of string2 is: defabcdef
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Outline

fig08_29.c

```
1  /* Fig. 8.29: fig08_29.c
2     Using strtok */
3  #include <stdio.h>
4  #include <string.h>
5
6  int main()
7  {
8     /* initialize array string */
9     char string[] = "This is a sentence with 7 tokens";
10    char *tokenPtr; /* create char pointer */
11
12    printf( "%s\n%s\n\n%s\n",
13            "The string to be tokenized is:", string,
14            "The tokens are:" );
15
16    tokenPtr = strtok( string, " " ); /* begin tokenizing sentence */
17
18    /* continue tokenizing sentence until tokenPtr becomes NULL */
19    while ( tokenPtr != NULL ) {
20       printf( "%s\n", tokenPtr );
21       tokenPtr = strtok( NULL, " " ); /* get next token */
22    } /* end while */
23
24    return 0; /* indicates successful termination */
25
26 } /* end main */
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
The string to be tokenized is:
This is a sentence with 7 tokens

The tokens are:
This
is
a
sentence
with
7
tokens
```


Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

8.9 Memory Functions of the String-handling Library

- Memory Functions
  - In <stdlib.h>
  - Manipulate, compare, and search blocks of memory
  - Can manipulate any block of data
- Pointer parameters are `void *`
  - Any pointer can be assigned to `void *`, and vice versa
  - `void *` cannot be dereferenced
    - Each function receives a size argument specifying the number of bytes (characters) to process

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

8.9 Memory Functions of the String-handling Library

Function prototype	Function description
`void *memcpy( void *s1, const void *s2, size_t n );`	Copies `n` characters from the object pointed to by `s2` into the object pointed to by `s1`. A pointer to the resulting object is returned.
`void *memmove( void *s1, const void *s2, size_t n );`	Copies `n` characters from the object pointed to by `s2` into the object pointed to by `s1`. The copy is performed as if the characters were first copied from the object pointed to by `s2` into a temporary array and then from the temporary array into the object pointed to by `s1`. A pointer to the resulting object is returned.
`int memcmp( const void *s1, const void *s2, size_t n );`	Compares the first `n` characters of the objects pointed to by `s1` and `s2`. The function returns `0`, less than `0` or greater than `0` if `s1` is equal to, less than or greater than `s2`.
`void *memchr( const void *s, int c, size_t n );`	Locates the first occurrence of `c` (converted to `unsigned char`) in the first `n` characters of the object pointed to by `s`. If `c` is found, a pointer to `c` in the object is returned. Otherwise, `NULL` is returned.
`void *memset( void *s, int c, size_t n );`	Copies `c` (converted to `unsigned char`) into the first `n` characters of the object pointed to by `s`. A pointer to the result is returned.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

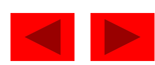


fig08_31.c

```
1  /* Fig. 8.31: fig08_31.c
2     Using memcpy */
3  #include <stdio.h>
4  #include <string.h>
5
6  int main()
7  {
8     char s1[ 17 ];                    /* create char array s1 */
9     char s2[]  = "Copy this string"; /* initialize char array s2 */
10
11     memcpy( s1, s2, 17 );
12     printf( "%s\n%s\"%s\"\n",
13             "After s2 is copied into s1 with memcpy,",
14             "s1 contains ", s1 );
15
16     return 0; /* indicates successful termination */
17
18 } /* end main */
```

Program Output

```
After s2 is copied into s1 with memcpy,
s1 contains "Copy this string"
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

fig08_32.c

```
1  /* Fig. 8.32: fig08_32.c
2     Using memmove */
3  #include <stdio.h>
4  #include <string.h>
5
6  int main()
7  {
8     char x[] = "Home Sweet Home"; /* initialize char array x */
9
10    printf( "%s%s\n", "The string in array x before memmove is: ", x );
11    printf( "%s%s\n", "The string in array x after memmove is: ",
12             memmove( x, &x[ 5 ], 10 ) );
13
14    return 0; /* indicates successful termination */
15
16 } /* end main */
```

Program Output

```
The string in array x before memmove is: Home Sweet Home
The string in array x after memmove is: Sweet Home Home
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

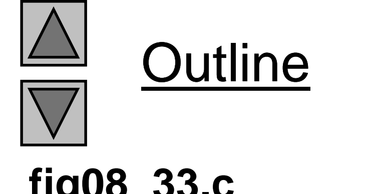


fig08_33.c

```
1  /* Fig. 8.33: fig08_33.c
2     Using memcmp */
3  #include <stdio.h>
4  #include <string.h>
5
6  int main()
7  {
8     char s1[] = "ABCDEFG"; /* initialize char array s1 */
9     char s2[] = "ABCDXYZ"; /* initialize char array s2 */
10
11    printf( "%s%s\n%s%s\n\n%s%2d\n%s%2d\n%s%2d\n",
12            "s1 = ", s1, "s2 = ", s2,
13            "memcmp( s1, s2, 4 ) = ", memcmp( s1, s2, 4 ),
14            "memcmp( s1, s2, 7 ) = ", memcmp( s1, s2, 7 ),
15            "memcmp( s2, s1, 7 ) = ", memcmp( s2, s1, 7 ) );
16
17    return 0; /* indicate successful termination */
18
19 } /* end main */
```

Program Output

```
s1 = ABCDEFG
s2 = ABCDXYZ

memcmp( s1, s2, 4 ) =  0
memcmp( s1, s2, 7 ) = -1
memcmp( s2, s1, 7 ) =  1
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

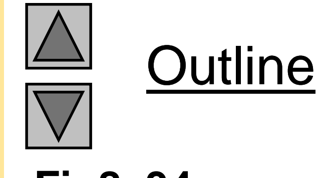


Fig8_34.c

```
1  /* Fig. 8.34: fig08_34.c
2     Using memchr */
3  #include <stdio.h>
4  #include <string.h>
5
6  int main()
7  {
8     const char *s = "This is a string"; /* initialize char pointer */
9
10    printf( "%s\'%c\'%s\"%s\"\n",
11            "The remainder of s after character ", 'r',
12            " is found is ", memchr( s, 'r', 16 ) );
13
14    return 0; /* indicates successful termination */
15
16 } /* end main */
```

Program Output

```
The remainder of s after character 'r' is found is "ring"
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

8.10 Other Functions of the String Handling Library

- `char *strerror( int errornum );`
  - Creates a system-dependent error message based on `errornum`
  - Returns a pointer to the string
- `size_t strlen( const char *s );`
  - Returns the number of characters (before `NULL`) in string `s`

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

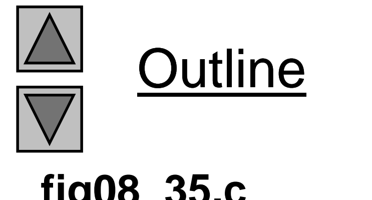


fig08_35.c

```
1  /* Fig. 8.35: fig08_35.c
2     Using memset */
3  #include <stdio.h>
4  #include <string.h>
5
6  int main()
7  {
8     char string1[ 15 ] = "BBBBBBBBBBBBBB"; /* initialize string1 */
9
10    printf( "string1 = %s\n", string1 );
11    printf( "string1 after memset = %s\n", memset( string1, 'b', 7 ) );
12
13    return 0; /* indicates successful termination */
14
15 } /* end main */
```

Program Output

```
string1 = BBBBBBBBBBBBBB
string1 after memset = bbbbbbbBBBBBBB
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

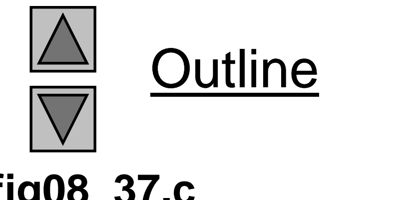


fig08_37.c

```
1  /* Fig. 8.37: fig08_37.c
2     Using strerror */
3  #include <stdio.h>
4  #include <string.h>
5
6  int main()
7  {
8     printf( "%s\n", strerror( 2 ) );
9
10     return 0; /* indicates successful termination */
11
12 } /* end main */
```

Program Output

```
No such file or directory
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

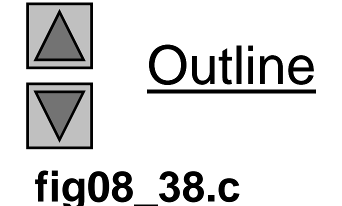


fig08_38.c

```
1  /* Fig. 8.38: fig08_38.c
2     Using strlen */
3  #include <stdio.h>
4  #include <string.h>
5
6  int main()
7  {
8     /* initialize 3 char pointers */
9     const char *string1 = "abcdefghijklmnopqrstuvwxyz";
10     const char *string2 = "four";
11     const char *string3 = "Boston";
12
13     printf("%s\"%s\"%s%lu\n%s\"%s\"%s%lu\n%s\"%s\"%s%lu\n",
14        "The length of ", string1, " is ",
15        ( unsigned long ) strlen( string1 ),
16        "The length of ", string2, " is ",
17        ( unsigned long ) strlen( string2 ),
18        "The length of ", string3, " is ",
19        ( unsigned long ) strlen( string3 ) );
20
21     return 0; /* indicates successful termination */
22
23 } /* end main */
```

Program Output

```
The length of "abcdefghijklmnopqrstuvwxyz" is 26
The length of "four" is 4
The length of "Boston" is 6
```

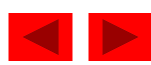
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Chapter 9 - Formatted Input/Output

Outline

9.1 Introduction
9.2 Streams
9.3 Formatting Output with `printf`
9.4 Printing Integers
9.5 Printing Floating-Point Numbers
9.6 Printing Strings and Characters
9.7 Other Conversion Specifiers
9.8 Printing with Field Widths and Precisions
9.9 Using Flags in the `printf` Format-Control String
9.10 Printing Literals and Escape Sequences
9.11 Formatting Input with `scanf`

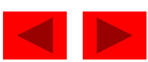
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Objectives

- In this chapter, you will learn:
  - To understand input and output streams.
  - To be able to use all print formatting capabilities.
  - To be able to use all input formatting capabilities.

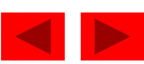
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



9.1 Introduction

- In this chapter
  - Presentation of results
  - `scanf` and `printf`
  - Streams (input and output)
    - `gets`, `puts`, `getchar`, `putchar` (in `<stdio.h>`)

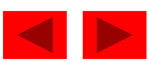
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



9.2 Streams

- Streams
  - Sequences of characters organized into lines
    - Each line consists of zero or more characters and ends with newline character
    - ANSI C must support lines of at least 254 characters
  - Performs all input and output
  - Can often be redirected
    - Standard input – keyboard
    - Standard output – screen
    - Standard error – screen
    - More in Chapter 11

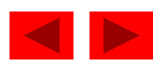
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



9.3 Formatting Output with `printf`

- `printf`
  - Precise output formatting
    - Conversion specifications: flags, field widths, precisions, etc.
  - Can perform rounding, aligning columns, right/left justification, inserting literal characters, exponential format, hexadecimal format, and fixed width and precision
- Format
  - `printf(` *format-control-string, other-arguments* `);`
  - Format control string: describes output format
  - Other-arguments: correspond to each conversion specification in format-control-string
    - Each specification begins with a percent sign(%), ends with conversion specifier

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

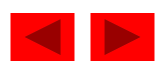


9.4 Printing Integers

Conversion Specifier	Description
d	Display a signed decimal integer.
i	Display a signed decimal integer. (*Note:* The i and d specifiers are different when used with scanf.)
o	Display an unsigned octal integer.
u	Display an unsigned decimal integer.
x or X	Display an unsigned hexadecimal integer. X causes the digits 0-9 and the letters A-F to be displayed and x causes the digits 0-9 and a-f to be displayed.
h or l (letter l)	Place before any integer conversion specifier to indicate that a short or long integer is displayed respectively. Letters h and l are more precisely called *length modifiers*.

Fig. 9.1 Integer conversion specifiers.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



9.4 Printing Integers

- Integer
  - Whole number (no decimal point): `25`, `0`, `-9`
  - Positive, negative, or zero
  - Only minus sign prints by default (later we shall change this)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

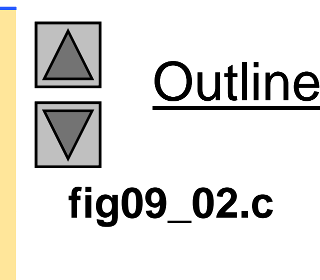


fig09_02.c

```
1  /* Fig 9.2: fig09_02.c */
2  /* Using the integer conversion specifiers */
3  #include <stdio.h>
4
5  int main()
6  {
7     printf( "%d\n", 455 );
8     printf( "%i\n", 455 );  /* i same as d in printf */
9     printf( "%d\n", +455 );
10    printf( "%d\n", -455 );
11    printf( "%hd\n", 32000 );
12    printf( "%ld\n", 2000000000 );
13    printf( "%o\n", 455 );
14    printf( "%u\n", 455 );
15    printf( "%u\n", -455 );
16    printf( "%x\n", 455 );
17    printf( "%X\n", 455 );
18
19    return 0; /* indicates successful termination */
20
21 } /* end main */
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
455
455
455
-455
32000
2000000000
707
455
4294966841
1c7
1C7
```

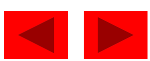
Program Ouptut

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

9.5 Printing Floating-Point Numbers

- Floating Point Numbers
  - Have a decimal point (`33.5`)
  - Exponential notation (computer's version of scientific notation)
    - `150.3` is `1.503 x 10²` in scientific
    - `150.3` is `1.503E+02` in exponential (`E` stands for exponent)
    - use `e` or `E`
  - `f` – print floating point with at least one digit to left of decimal
  - `g` (or `G`) - prints in `f` or `e` with no trailing zeros (`1.2300` becomes `1.23`)
    - Use exponential if exponent less than `-4`, or greater than or equal to precision (6 digits by default)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



9.5 Printing Floating-Point Numbers

Conversion specifier	Description
`e` or `E`	Display a floating-point value in exponential notation.
`f`	Display floating-point values.
`g` or `G`	Display a floating-point value in either the floating-point form `f` or the exponential form `e` (or `E`).
`L`	Place before any floating-point conversion specifier to indicate that a `long double` floating-point value is displayed.

Fig. 9.3 Floating-point conversion specifiers.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

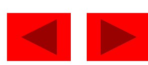


fig09_04.c

```
1  /* Fig 9.4: fig09_04.c */
2  /* Printing floating-point numbers with
3     floating-point conversion specifiers */
4
5  #include <stdio.h>
6
7  int main()
8  {
9     printf( "%e\n", 1234567.89 );
10    printf( "%e\n", +1234567.89 );
11    printf( "%e\n", -1234567.89 );
12    printf( "%E\n", 1234567.89 );
13    printf( "%f\n", 1234567.89 );
14    printf( "%g\n", 1234567.89 );
15    printf( "%G\n", 1234567.89 );
16
17    return 0; /* indicates successful termination */
18
19 } /* end main */
```

Program Output

```
1.234568e+006
1.234568e+006
-1.234568e+006
1.234568E+006
1234567.890000
1.23457e+006
1.23457E+006
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

9.6 Printing Strings and Characters

- `c`
  - Prints `char` argument
  - Cannot be used to print the first character of a string
- `s`
  - Requires a pointer to `char` as an argument
  - Prints characters until `NULL` (`'\0'`) encountered
  - Cannot print a `char` argument
- Remember
  - Single quotes for character constants (`'z'`)
  - Double quotes for strings `"z"` (which actually contains two characters, `'z'` and `'\0'`)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Outline

fig09_05.c

```
1  /* Fig 9.5: fig09_05c */
2  /* Printing strings and characters */
3  #include <stdio.h>
4
5  int main()
6  {
7     char character = 'A'; /* initialize char */
8     char string[] = "This is a string"; /* initialize char array */
9     const char *stringPtr = "This is also a string"; /* char pointer */
10
11     printf( "%c\n", character );
12     printf( "%s\n", "This is a string" );
13     printf( "%s\n", string );
14     printf( "%s\n", stringPtr );
15
16     return 0; /* indicates successful termination */
17
18  } /* end main */
```

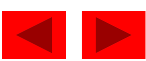
```
A
This is a string
This is a string
This is also a string
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

9.7 Other Conversion Specifiers

- `p`
  - Displays pointer value (address)
- `n`
  - Stores number of characters already output by current `printf` statement
  - Takes a pointer to an integer as an argument
  - Nothing printed by a `%n` specification
  - Every `printf` call returns a value
    - Number of characters output
    - Negative number if error occurs
- `%`
  - Prints a percent sign
  - `%%`

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

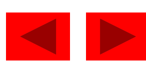


9.7 Other Conversion Specifiers

Conversion specifier	Description
p	Display a pointer value in an implementation-defined manner.
n	Store the number of characters already output in the current `printf` statement. A pointer to an integer is supplied as the corresponding argument. Nothing is displayed.
%	Display the percent character.

Fig. 9.6 Other conversion specifiers.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

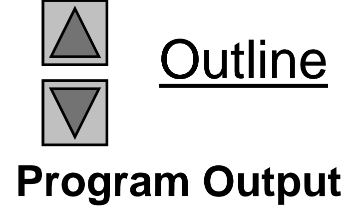


Outline

fig09_07.c (1 of 2)

```
1  /* Fig 9.7: fig09_07.c */
2  /* Using the p, n, and % conversion specifiers */
3  #include <stdio.h>
4
5  int main()
6  {
7     int *ptr;      /* define pointer to int */
8     int x = 12345; /* initialize int x */
9     int y;         /* define int y */
10
11    ptr = &x;      /* assign address of x to ptr */
12    printf( "The value of ptr is %p\n", ptr );
13    printf( "The address of x is %p\n\n", &x );
14
15    printf( "Total characters printed on this line:%n", &y );
16    printf( " %d\n\n", y );
17
18    y = printf( "This line has 28 characters\n" );
19    printf( "%d characters were printed\n\n", y );
20
21    printf( "Printing a %% in a format control string\n" );
22
23    return 0; /* indicates successful termination */
24
25 } /* end main */
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Program Output

```
The value of ptr is 0012FF78
The address of x is 0012FF78

Total characters printed on this line: 38

This line has 28 characters
28 characters were printed

Printing a % in a format control string
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

9.8 Printing with Field Widths and Precisions

- Field width
  - Size of field in which data is printed
  - If width larger than data, default right justified
    - If field width too small, increases to fit data
    - Minus sign uses one character position in field
  - Integer width inserted between % and conversion specifier
  - %4d – field width of 4

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

9.8 Printing with Field Widths and Precisions

- Precision
  - Meaning varies depending on data type
  - Integers (default `1`)
    - Minimum number of digits to print
      - If data too small, prefixed with zeros
  - Floating point
    - Number of digits to appear after decimal (`e` and `f`)
      - For `g` – maximum number of significant digits
  - Strings
    - Maximum number of characters to be written from string
  - Format
    - Use a dot (.) then precision number after %

```
%.3f
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

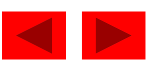
9.8 Printing with Field Widths and Precisions

- Field width and precision
  - Can both be specified
    - `%width.precision`

      `%5.3f`
  - Negative field width – left justified
  - Positive field width – right justified
  - Precision must be positive
  - Can use integer expressions to determine field width and precision values
    - Place an asterisk (*) in place of the field width or precision
      - Matched to an `int` argument in argument list
    - Example:

```
printf( "%*.*f", 7, 2, 98.736 );
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



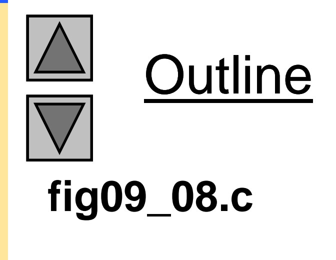


fig09_08.c

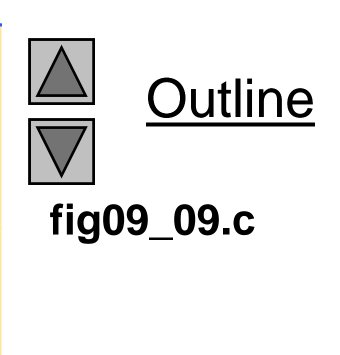
```
1  /* Fig 9.8: fig09_08.c */
2  /* Printing integers right-justified */
3  #include <stdio.h>
4
5  int main()
6  {
7     printf( "%4d\n", 1 );
8     printf( "%4d\n", 12 );
9     printf( "%4d\n", 123 );
10    printf( "%4d\n", 1234 );
11    printf( "%4d\n\n", 12345 );
12
13    printf( "%4d\n", -1 );
14    printf( "%4d\n", -12 );
15    printf( "%4d\n", -123 );
16    printf( "%4d\n", -1234 );
17    printf( "%4d\n", -12345 );
18
19    return 0; /* indicates successful termination */
20
21 } /* end main */
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
 1
  12
 123
1234
12345

  -1
 -12
-123
-1234
-12345
```

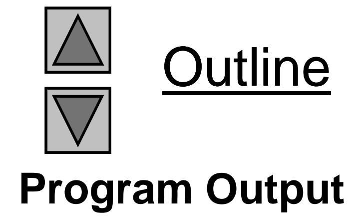

Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.




```
1  /* Fig 9.9: fig09_09.c */
2  /* Using precision while printing integers,
3     floating-point numbers, and strings */
4  #include <stdio.h>
5
6  int main()
7  {
8     int i = 873;                 /* initialize int i */
9     double f = 123.94536;        /* initialize double f */
10    char s[] = "Happy Birthday"; /* initialize char array s */
11
12    printf( "Using precision for integers\n" );
13    printf( "\t%.4d\n\t%.9d\n\n", i, i );
14
15    printf( "Using precision for floating-point numbers\n" );
16    printf( "\t%.3f\n\t%.3e\n\t%.3g\n\n", f, f, f );
17
18    printf( "Using precision for strings\n" );
19    printf( "\t%.11s\n", s );
20
21    return 0; /* indicates successful termination */
22
23 } /* end main */
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Program Output

```
 Using precision for integers
        0873
        000000873

Using precision for floating-point numbers
        123.945
        1.239e+002
        124

Using precision for strings
        Happy Birth
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

9.9 Using Flags in the `printf` Format-Control String

- Flags
  - Supplement formatting capabilities
  - Place flag immediately to the right of percent sign
  - Several flags may be combined

Flag	Description
`-` (minus sign)	Left-justify the output within the specified field.
`+` (plus sign)	Display a plus sign preceding positive values and a minus sign preceding negative values.
space	Print a space before a positive value not printed with the `+` flag.
`#`	Prefix `0` to the output value when used with the octal conversion specifier `o`.
	Prefix `0x` or `0X` to the output value when used with the hexadecimal conversion specifiers `x` or `X`.
	Force a decimal point for a floating-point number printed with `e`, `E`, `f`, `g` or `G` that does not contain a fractional part. (Normally the decimal point is only printed if a digit follows it.) For `g` and `G` specifiers, trailing zeros are not eliminated.
`0` (zero)	Pad a field with leading zeros.

Fig. 9.10 Format control string flags.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Outline

fig09_11.c

```
1  /* Fig 9.11: fig09_11.c */
2  /* Right justifying and left justifying values */
3  #include <stdio.h>
4
5  int main()
6  {
7     printf( "%10s%10d%10c%10f\n\n", "hello", 7, 'a', 1.23 );
8     printf( "%-10s%-10d%-10c%-10f\n", "hello", 7, 'a', 1.23 );
9
10     return 0; /* indicates successful termination */
11
12 } /* end main */
```

Program Output

```
     hello         7         a  1.230000

hello     7         a         1.230000
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Outline

fig09_12.c

```
1  /* Fig 9.12: fig09_12.c */
2  /* Printing numbers with and without the + flag */
3  #include <stdio.h>
4
5  int main()
6  {
7     printf( "%d\n%d\n", 786, -786 );
8     printf( "%+d\n%+d\n", 786, -786 );
9
10     return 0; /* indicates successful termination */
11
12 } /* end main */
```

Program Output

```
786
-786
+786
-786
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Outline

fig09_13.c

```
1  /* Fig 9.13: fig09_13.c */
2  /* Printing a space before signed values
3     not preceded by + or - */
4  #include <stdio.h>
5
6  int main()
7  {
8     printf( "% d\n% d\n", 547, -547 );
9
10     return 0; /* indicates successful termination */
11
12 } /* end main */
```

Program Output

```
 547
-547
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

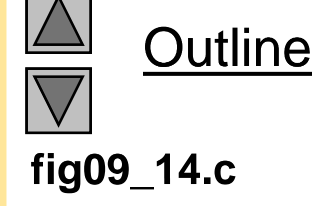


fig09_14.c

```
1  /* Fig 9.14: fig09_14.c */
2  /* Using the # flag with conversion specifiers
3     o, x, X and any floating-point specifier */
4  #include <stdio.h>
5
6  int main()
7  {
8     int c = 1427;      /* initialize c */
9     double p = 1427.0; /* initialize p */
10
11     printf( "%#o\n", c );
12     printf( "%#x\n", c );
13     printf( "%#X\n", c );
14     printf( "\n%g\n", p );
15     printf( "%#g\n", p );
16
17     return 0; /* indicates successful termination */
18
19  } /* end main */
```

Program Output

```
02623
0x593
0X593

1427
1427.00
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Outline

fig09_15.c

```
1  /* Fig 9.15: fig09_15.c */
2  /* Printing with the 0( zero ) flag fills in leading zeros */
3  #include <stdio.h>
4
5  int main()
6  {
7     printf( "%+09d\n", 452 );
8     printf( "%09d\n", 452 );
9
10     return 0; /* indicates successful termination */
11
12 } /* end main */
```

Program Output

```
+00000452
000000452
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

9.10 Printing Literals and Escape Sequences

- Printing Literals
  - Most characters can be printed
  - Certain "problem" characters, such as the quotation mark "
  - Must be represented by escape sequences
    - Represented by a backslash \ followed by an escape character

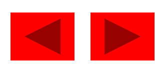
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

9.10 Printing Literals and Escape Sequences

Escape sequence	Description
`\'`	Output the single quote (`'`) character.
`\"`	Output the double quote (`"`) character.
`\?`	Output the question mark (`?`) character.
`\\`	Output the backslash (`\`) character.
`\a`	Cause an audible (bell) or visual alert.
`\b`	Move the cursor back one position on the current line.
`\f`	Move the cursor to the start of the next logical page.
`\n`	Move the cursor to the beginning of the next line.
`\r`	Move the cursor to the beginning of the current line.
`\t`	Move the cursor to the next horizontal tab position.
`\v`	Move the cursor to the next vertical tab position.

Fig. 9.16 Escape sequences.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

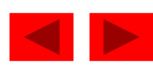


9.11 Formatting Input with scanf

Conversion specifier	Description
Integers	
d	Read an optionally signed decimal integer. The corresponding argument is a pointer to integer.
i	Read an optionally signed decimal, octal, or hexadecimal integer. The corresponding argument is a pointer to integer.
o	Read an \octal integer. The corresponding argument is a pointer to unsigned integer.
u	Read an unsigned decimal integer. The corresponding argument is a pointer to unsigned integer.
x or X	Read a hexadecimal integer. The corresponding argument is a pointer to unsigned integer.
h or l	Place before any of the integer conversion specifiers to indicate that a short or l ong integer is to be input.

Fig. 9.17 Conversion specifiers for scanf.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

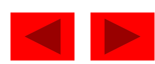


9.11 Formatting Input with scanf

Conversion specifier	Description
Floating-point numbers	
e, E, f, g or G	Read a floating-point value. The corresponding argument is a pointer to a floating-point variable.
l or L	Place before any of the floating-point conversion specifiers to indicate that a double or long double value is to be input.
Characters and strings	
C	Read a character. The corresponding argument is a pointer to char, no null ('\0') is added.
S	Read a string. The corresponding argument is a pointer to an array of type char that is large enough to hold the string and a terminating null ('\0') character—which is automatically added.
Scan set	
[scan characters	Scan a string for a set of characters that are stored in an array.
Miscellaneous	
P	Read an address of the same form produced when an address is output with %p in a printf statement.
N	Store the number of characters input so far in this scanf. The corresponding argument is a pointer to integer
%	Skip a percent sign (%) in the input.

Fig. 9.17 Conversion specifiers for scanf.

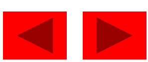
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



9.11 Formatting Input with `scanf`

- `scanf`
  - Input formatting
  - Capabilities
    - Input all types of data
    - Input specific characters
    - Skip specific characters
- Format
  - `scanf`(*format-control-string*, *other-arguments*);
  - Format-control-string
    - Describes formats of inputs
  - Other-arguments
    - Pointers to variables where input will be stored
  - Can include field widths to read a specific number of characters from the stream

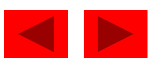
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



9.11 Formatting Input with `scanf`

- Scan sets
  - Set of characters enclosed in square brackets `[]`
    - Preceded by `%` sign
  - Scans input stream, looking only for characters in scan set
    - Whenever a match occurs, stores character in specified array
    - Stops scanning once a character not in the scan set is found
  - Inverted scan sets
    - Use a caret `^`: `[^aeiou]`
    - Causes characters not in the scan set to be stored
- Skipping characters
  - Include character to skip in format control
  - Or, use `*` (assignment suppression character)
    - Skips any type of character without storing it

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



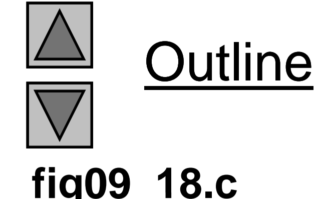


fig09_18.c

```
1  /* Fig 9.18: fig09_18.c */
2  /* Reading integers */
3  #include <stdio.h>
4
5  int main()
6  {
7     int a; /* define a */
8     int b; /* define b */
9     int c; /* define c */
10    int d; /* define d */
11    int e; /* define e */
12    int f; /* define f */
13    int g; /* define g */
14
15    printf( "Enter seven integers: " );
16    scanf( "%d%i%i%i%o%u%x", &a, &b, &c, &d, &e, &f, &g );
17
18    printf( "The input displayed as decimal integers is:\n" );
19    printf( "%d %d %d %d %d %d %d\n", a, b, c, d, e, f, g );
20
21    return 0; /* indicates successful termination */
22
23 } /* end main */
```

Program Output

```
Enter seven integers: -70 -70 070 0x70 70 70 70
The input displayed as decimal integers is:
-70 -70 56 112 56 70 112
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

fig09_19.c

```
1  /* Fig 9.19: fig09_19.c */
2  /* Reading floating-point numbers */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main()
7  {
8     double a; /* define a */
9     double b; /* define b */
10    double c; /* define c */
11
12    printf( "Enter three floating-point numbers: \n" );
13    scanf( "%le%lf%lg", &a, &b, &c );
14
15    printf( "Here are the numbers entered in plain\n" );
16    printf( "floating-point notation:\n" );
17    printf( "%f\n%f\n%f\n", a, b, c );
18
19    return 0; /* indicates successful termination */
20
21 } /* end main */
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

fig09_20.c

```
1  /* Fig 9.20: fig09_20.c */
2  /* Reading characters and strings */
3  #include <stdio.h>
4
5  int main()
6  {
7     char x;      /* define x */
8     char y[ 9 ]; /* define array y */
9
10    printf( "Enter a string: " );
11    scanf( "%c%s", &x, y );
12
13    printf( "The input was:\n" );
14    printf( "the character \"%c\" ", x );
15    printf( "and the string \"%s\"\n", y );
16
17    return 0; /* indicates successful termination */
18
19 } /* end main */
```

Program Output

```
Enter a string: Sunday
The input was:
the character "S" and the string "unday"
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

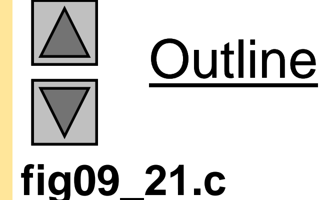


fig09_21.c

```
1  /* Fig 9.21: fig09_21.c */
2  /* Using a scan set */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main()
7  {
8     char z[ 9 ]; /* define array z */
9
10    printf( "Enter string: " );
11    scanf( "%[aeiou]", z ); /* search for set of characters */
12
13    printf( "The input was \"%s\"\n", z );
14
15    return 0; /* indicates successful termination */
16
17 } /* end main */
```

Program Output

```
Enter string: ooeeooahah
The input was "ooeeooa"
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Outline

fig09_22.c

```
1  /* Fig 9.22: fig09_22.c */
2  /* Using an inverted scan set */
3  #include <stdio.h>
4
5  int main()
6  {
7     char z[ 9 ] = { '\0' }; /* initialize array z */
8
9     printf( "Enter a string: " );
10    scanf( "%[^aeiou]", z ); /* inverted scan set */
11
12    printf( "The input was \"%s\"\n", z );
13
14    return 0; /* indicates successful termination */
15
16 } /* end main */
```

Program Output

```
Enter a string: String
The input was "Str"
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

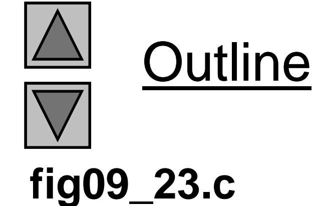


fig09_23.c

```
1  /* Fig 9.23: fig09_23.c */
2  /* inputting data with a field width */
3  #include <stdio.h>
4
5  int main()
6  {
7     int x; /* define x */
8     int y; /* define y */
9
10    printf( "Enter a six digit integer: " );
11    scanf( "%2d%d", &x, &y );
12
13    printf( "The integers input were %d and %d\n", x, y );
14
15    return 0; /* indicates successful termination */
16
17 } /* end main */
```

Program Output

```
Enter a six digit integer: 123456
The integers input were 12 and 3456
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

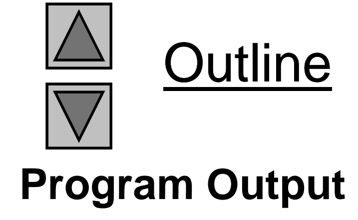
Outline

fig09_24.c

```
1  /* Fig 9.24: fig09_24.c */
2  /* Reading and discarding characters from the input stream */
3  #include <stdio.h>
4
5  int main()
6  {
7     int month1; /* define month1 */
8     int day1;   /* define day1 */
9     int year1;  /* define year1 */
10    int month2; /* define month2 */
11    int day2;   /* define day2 */
12    int year2;  /* define year2 */
13
14    printf( "Enter a date in the form mm-dd-yyyy: " );
15    scanf( "%d%*c%d%*c%d", &month1, &day1, &year1 );
16
17    printf( "month = %d  day = %d  year = %d\n\n", month1, day1, year1 );
18
19    printf( "Enter a date in the form mm/dd/yyyy: " );
20    scanf( "%d%*c%d%*c%d", &month2, &day2, &year2 );
21
22    printf( "month = %d  day = %d  year = %d\n", month2, day2, year2 );
23
24    return 0; /* indicates successful termination */
25
26 } /* end main */
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
Enter a date in the form mm-dd-yyyy: 11-18-2003
month = 11  day = 18  year = 2003

Enter a date in the form mm/dd/yyyy: 11/18/2003
month = 11  day = 18  year = 2003
```



Program Output

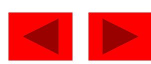
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Chapter 10 - C Structures, Unions, Bit Manipulations, and Enumerations

Outline

10.1 Introduction
10.2 Structure Definitions
10.3 Initializing Structures
10.4 Accessing Members of Structures
10.5 Using Structures with Functions
10.6 `typedef`
10.7 Example: High-Performance Card Shuffling and Dealing Simulation
10.8 Unions
10.9 Bitwise Operators
10.10 Bit Fields
10.11 Enumeration Constants

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Objectives

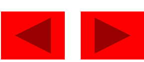
- In this tutorial, you will learn:
  - To be able to create and use structures, unions and enumerations.
  - To be able to pass structures to functions call by value and call by reference.
  - To be able to manipulate data with the bitwise operators.
  - To be able to create bit fields for storing data compactly.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

10.1 Introduction

- Structures
  - Collections of related variables (aggregates) under one name
    - Can contain variables of different data types
  - Commonly used to define records to be stored in files
  - Combined with pointers, can create linked lists, stacks, queues, and trees

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



10.2 Structure Definitions

- Example

```
struct card {
   char *face;
   char *suit;
};
```

  - `struct` introduces the definition for structure `card`
  - `card` is the structure name and is used to declare variables of the structure type
  - `card` contains two members of type `char *`
    - These members are `face` and `suit`

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

10.2 Structure Definitions

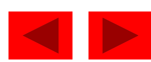
- `struct` information
  - A `struct` cannot contain an instance of itself
  - Can contain a member that is a pointer to the same structure type
  - A structure definition does not reserve space in memory
    - Instead creates a new data type used to define structure variables
- Definitions
  - Defined like other variables:

```
card oneCard, deck[ 52 ], *cPtr;
```

  - Can use a comma separated list:

```
struct card {
   char *face;
   char *suit;
} oneCard, deck[ 52 ], *cPtr;
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



10.2 Structure Definitions

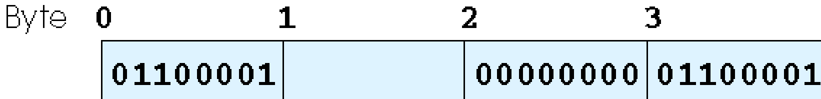


Fig. 10.1 A possible storage alignment for a variable of type `struct example` showing an undefined area in memory.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

10.2 Structure Definitions

- Valid Operations
  - Assigning a structure to a structure of the same type
  - Taking the address (&) of a structure
  - Accessing the members of a structure
  - Using the `sizeof` operator to determine the size of a structure

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

10.3 Initializing Structures

- Initializer lists
  - Example:

```
card oneCard = { "Three", "Hearts" };
```

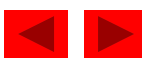
- Assignment statements
  - Example:

```
card threeHearts = oneCard;
```

  - Could also define and initialize `threeHearts` as follows:

```
card threeHearts;
threeHearts.face = "Three";
threeHearts.suit = "Hearts";
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



10.4 Accessing Members of Structures

- Accessing structure members
  - Dot operator (`.`) used with structure variables

```
card myCard;
printf( "%s", myCard.suit );
```

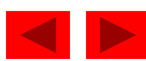
  - Arrow operator (`->`) used with pointers to structure variables

```
card *myCardPtr = &myCard;
printf( "%s", myCardPtr->suit );
```

  - `myCardPtr->suit` is equivalent to

```
( *myCardPtr ).suit
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



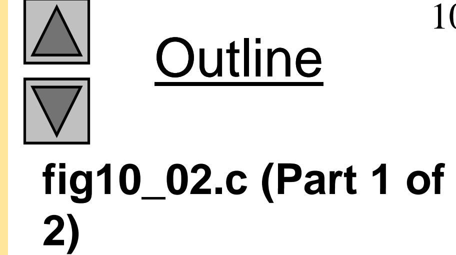


fig10_02.c (Part 1 of 2)

```
1  /* Fig. 10.2: fig10_02.c
2     Using the structure member and
3     structure pointer operators */
4  #include <stdio.h>
5
6  /* card structure definition */
7  struct card {
8     char *face; /* define pointer face */
9     char *suit; /* define pointer suit */
10 }; /* end structure card */
11
12 int main()
13 {
14    struct card a;     /* define struct a */
15    struct card *aPtr; /* define a pointer to card */
16
17    /* place strings into card structures */
18    a.face = "Ace";
19    a.suit = "Spades";
20
21    aPtr = &a; /* assign address of a to aPtr */
22
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
23    printf( "%s%s%s\n%s%s%s\n%s%s%s\n", a.face, " of ", a.suit,
24            aPtr->face, " of ", aPtr->suit,
25            ( *aPtr ).face, " of ", ( *aPtr ).suit );
26
27    return 0; /* indicates successful termination */
28
29 } /* end main */
```

```
Ace of Spades
Ace of Spades
Ace of Spades
```


fig10_02.c (Part 2 of 2)

Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

10.5 Using Structures With Functions

- Passing structures to functions
  - Pass entire structure
    - Or, pass individual members
  - Both pass call by value
- To pass structures call-by-reference
  - Pass its address
  - Pass reference to it
- To pass arrays call-by-value
  - Create a structure with the array as a member
  - Pass the structure

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

10.6 typedef

- `typedef`
  - Creates synonyms (aliases) for previously defined data types
  - Use `typedef` to create shorter type names
  - Example:

```
typedef struct Card *CardPtr;
```

  - Defines a new type name `CardPtr` as a synonym for type `struct Card *`
  - `typedef` does not create a new data type
    - Only creates an alias

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

10.7 Example: High-Performance Card-shuffling and Dealing Simulation

- Pseudocode:
  - Create an array of card structures
  - Put cards in the deck
  - Shuffle the deck
  - Deal the cards

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

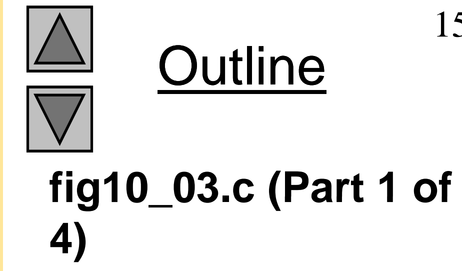


fig10_03.c (Part 1 of 4)

```
1  /* Fig. 10.3: fig10_03.c
2     The card shuffling and dealing program using structures */
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include <time.h>
6
7  /* card structure definition */
8  struct card {
9     const char *face; /* define pointer face */
10    const char *suit; /* define pointer suit */
11 }; /* end structure card */
12
13 typedef struct card Card;
14
15 /* prototypes */
16 void fillDeck( Card * const wDeck, const char * wFace[],
17    const char * wSuit[] );
18 void shuffle( Card * const wDeck );
19 void deal( const Card * const wDeck );
20
21 int main()
22 {
23    Card deck[ 52 ]; /* define array of Cards */
24
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

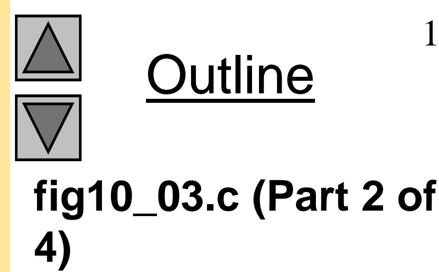
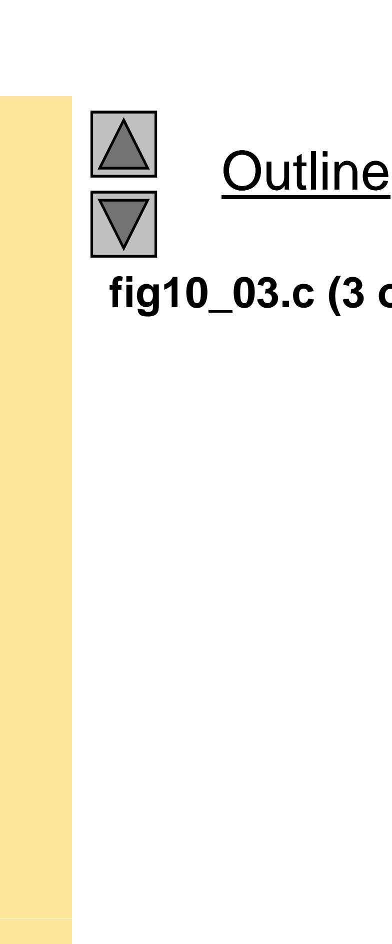


fig10_03.c (Part 2 of 4)

```
25    /* initialize array of pointers */
26    const char *face[] = { "Ace", "Deuce", "Three", "Four", "Five",
27       "Six", "Seven", "Eight", "Nine", "Ten",
28       "Jack", "Queen", "King"};
29
30    /* initialize array of pointers */
31    const char *suit[] = { "Hearts", "Diamonds", "Clubs", "Spades"};
32
33    srand( time( NULL ) ); /* randomize */
34
35    fillDeck( deck, face, suit ); /* load the deck with Cards */
36    shuffle( deck ); /* put Cards in random order */
37    deal( deck ); /* deal all 52 Cards */
38
39    return 0; /* indicates successful termination */
40
41 } /* end main */
42
43 /* place strings into Card structures */
44 void fillDeck( Card * const wDeck, const char * wFace[],
45    const char * wSuit[] )
46 {
47    int i; /* counter */
48
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.




```
49    /* loop through wDeck */
50    for ( i = 0; i <= 51; i++ ) {
51       wDeck[ i ].face = wFace[ i % 13 ];
52       wDeck[ i ].suit = wSuit[ i / 13 ];
53    } /* end for */
54
55 } /* end function fillDeck */
56
57 /* shuffle cards */
58 void shuffle( Card * const wDeck )
59 {
60    int i;     /* counter */
61    int j;     /* variable to hold random value between 0 - 51 */
62    Card temp; /* define temporary structure for swapping Cards */
63
64    /* loop through wDeck randomly swapping Cards */
65    for ( i = 0; i <= 51; i++ ) {
66       j = rand() % 52;
67       temp = wDeck[ i ];
68       wDeck[ i ] = wDeck[ j ];
69       wDeck[ j ] = temp;
70    } /* end for */
71
72 } /* end function shuffle */
73
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

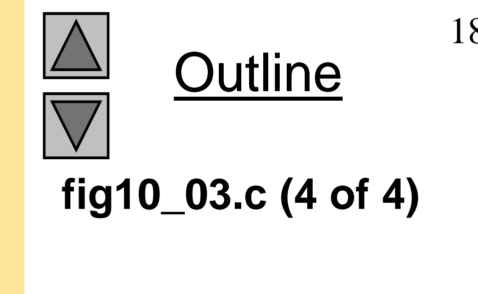
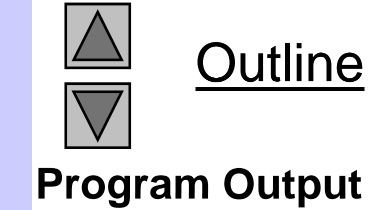


fig10_03.c (4 of 4)

```
74 /* deal cards */
75 void deal( const Card * const wDeck )
76 {
77    int i; /* counter */
78
79    /* loop through wDeck */
80    for ( i = 0; i <= 51; i++ ) {
81       printf( "%5s of %-8s%c", wDeck[ i ].face, wDeck[ i ].suit,
82          ( i + 1 ) % 2 ? '\t' : '\n' );
83    } /* end for */
84
85 } /* end function deal */
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Program Output

```
 Four of Clubs         Three of Hearts
Three of Diamonds      Three of Spades
 Four of Diamonds        Ace of Diamonds
 Nine of Hearts          Ten of Clubs
Three of Clubs          Four of Hearts
Eight of Clubs          Nine of Diamonds
Deuce of Clubs         Queen of Clubs
Seven of Clubs          Jack of Spades
  Ace of Clubs          Five of Diamonds
  Ace of Spades         Five of Clubs
Seven of Diamonds        Six of Spades
Eight of Spades        Queen of Hearts
 Five of Spades        Deuce of Diamonds
Queen of Spades          Six of Hearts
Queen of Diamonds      Seven of Hearts
 Jack of Diamonds       Nine of Spades
Eight of Hearts         Five of Hearts
 King of Spades          Six of Clubs
Eight of Diamonds        Ten of Spades
  Ace of Hearts         King of Hearts
 Four of Spades         Jack of Hearts
Deuce of Hearts         Jack of Clubs
Deuce of Spades          Ten of Diamonds
Seven of Spades         Nine of Clubs
 King of Clubs           Six of Diamonds
  Ten of Hearts         King of Diamonds
```

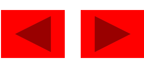
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

10.8 Unions

- `union`
  - Memory that contains a variety of objects over time
  - Only contains one data member at a time
  - Members of a `union` share space
  - Conserves storage
  - Only the last data member defined can be accessed
- `union` definitions
  - Same as struct

```
union Number {
  int x;
  float y;
};
union Number value;
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



10.8 Unions

- Valid `union` operations
  - Assignment to `union` of same type: `=`
  - Taking address: `&`
  - Accessing union members: `.`
  - Accessing members using pointers: `->`

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

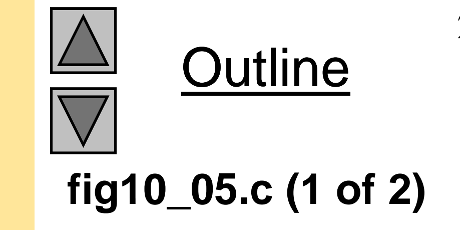


fig10_05.c (1 of 2)

```
1  /* Fig. 10.5: fig10_05.c
2     An example of a union */
3  #include <stdio.h>
4
5  /* number union definition */
6  union number {
7     int x;    /* define int x */
8     double y; /* define double y */
9  }; /* end union number */
10
11 int main()
12 {
13    union number value; /* define union value */
14
15    value.x = 100; /* put an integer into the union */
16    printf( "%s\n%s\n%s%d\n%s%f\n\n",
17           "Put a value in the integer member",
18           "and print both members.",
19           "int:   ", value.x,
20           "double:\n", value.y );
21
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Outline

fig10_05.c (2 of 2)

```
22    value.y = 100.0; /* put a double into the same union */
23    printf( "%s\n%s\n%s%d\n%s%f\n",
24            "Put a value in the floating member",
25            "and print both members.",
26            "int:   ", value.x,
27            "double:\n", value.y );
28
29    return 0; /* indicates successful termination */
30
31 } /* end main */
```

```
Put a value in the integer member
and print both members.
int:   100
double:
-92559592117433136000000000000000000000000000000000000000000000.000000

Put a value in the floating member
and print both members.
int:   0
double:
100.000000
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

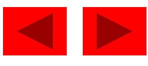
10.9 Bitwise Operators

- All data represented internally as sequences of bits
  - Each bit can be either 0 or 1
  - Sequence of 8 bits forms a byte

Operator		Description
&	bitwise AND	The bits in the result are set to 1 if the corresponding bits in the two operands are both 1.
|	bitwise inclusive OR	The bits in the result are set to 1 if at least one of the corresponding bits in the two operands is 1.
^	bitwise exclusive OR	The bits in the result are set to 1 if exactly one of the corresponding bits in the two operands is 1.
<<	left shift	Shifts the bits of the first operand left by the number of bits specified by the second operand; fill from the right with 0 bits.
>>	right shift	Shifts the bits of the first operand right by the number of bits specified by the second operand; the method of filling from the left is machine dependent.
~	one's complement	All 0 bits are set to 1 and all 1 bits are set to 0.

Fig. 10.6 The bitwise operators.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



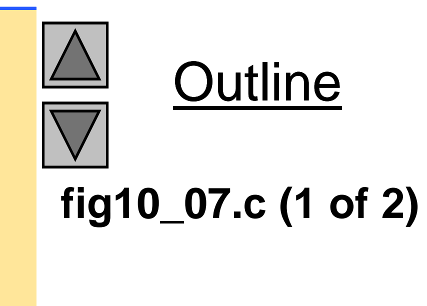


fig10_07.c (1 of 2)

```
1  /* Fig. 10.7: fig10_07.c
2     Printing an unsigned integer in bits */
3  #include <stdio.h>
4
5  void displayBits( unsigned value ); /* prototype */
6
7  int main()
8  {
9     unsigned x; /* variable to hold user input */
10
11    printf( "Enter an unsigned integer: " );
12    scanf( "%u", &x );
13
14    displayBits( x );
15
16    return 0; /* indicates successful termination */
17
18 } /* end main */
19
20 /* display bits of an unsigned integer value */
21 void displayBits( unsigned value )
22 {
23    unsigned c; /* counter */
24
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
25    /* define displayMask and left shift 31 bits */
26    unsigned displayMask = 1 << 31;
27
28    printf( "%7u = ", value );
29
30    /* loop through bits */
31    for ( c = 1; c <= 32; c++ ) {
32       putchar( value & displayMask ? '1' : '0' );
33       value <<= 1; /* shift value left by 1 */
34
35       if ( c % 8 == 0 ) { /* output space after 8 bits */
36          putchar( ' ' );
37       } /* end if */
38
39    } /* end for */
40
41    putchar( '\n' );
42 } /* end function displayBits */
```

```
Enter an unsigned integer: 65000
  65000 = 00000000 00000000 11111101 11101000
```

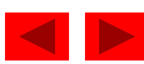

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

10.9 Bitwise Operators

Bit 1	Bit 2	Bit 1 & Bit 2
0	0	0
1	0	0
0	1	0
1	1	1

Fig. 10.8 Results of combining two bits with the bitwise AND operator &.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Outline

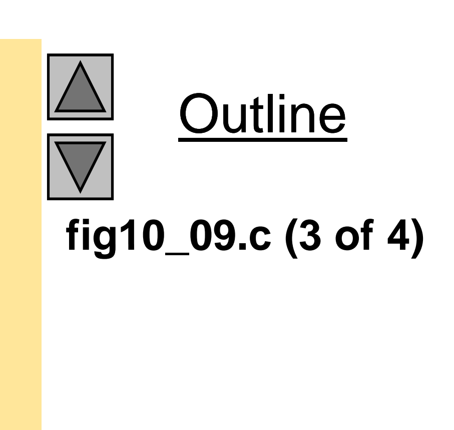
fig10_09.c (1 of 4)

```
1  /* Fig. 10.9: fig10_09.c
2     Using the bitwise AND, bitwise inclusive OR, bitwise
3     exclusive OR and bitwise complement operators */
4  #include <stdio.h>
5
6  void displayBits( unsigned value ); /* prototype */
7
8  int main()
9  {
10    unsigned number1; /* define number1 */
11    unsigned number2; /* define number2 */
12    unsigned mask;    /* define mask */
13    unsigned setBits; /* define setBits */
14
15    /* demonstrate bitwise & */
16    number1 = 65535;
17    mask = 1;
18    printf( "The result of combining the following\n" );
19    displayBits( number1 );
20    displayBits( mask );
21    printf( "using the bitwise AND operator & is\n" );
22    displayBits( number1 & mask );
23
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
24    /* demonstrate bitwise | */
25    number1 = 15;
26    setBits = 241;
27    printf( "\nThe result of combining the following\n" );
28    displayBits( number1 );
29    displayBits( setBits );
30    printf( "using the bitwise inclusive OR operator | is\n" );
31    displayBits( number1 | setBits );
32
33    /* demonstrate bitwise exclusive OR */
34    number1 = 139;
35    number2 = 199;
36    printf( "\nThe result of combining the following\n" );
37    displayBits( number1 );
38    displayBits( number2 );
39    printf( "using the bitwise exclusive OR operator ^ is\n" );
40    displayBits( number1 ^ number2 );
41
42    /* demonstrate bitwise complement */
43    number1 = 21845;
44    printf( "\nThe one's complement of\n" );
45    displayBits( number1 );
46    printf( "is\n" );
47    displayBits( ~number1 );
48
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.




```
49    return 0; /* indicates successful termination */
50
51 } /* end main */
52
53 /* display bits of an unsigned integer value */
54 void displayBits( unsigned value )
55 {
56    unsigned c; /* counter */
57
58    /* declare displayMask and left shift 31 bits */
59    unsigned displayMask = 1 << 31;
60
61    printf( "%10u = ", value );
62
63    /* loop through bits */
64    for ( c = 1; c <= 32; c++ ) {
65       putchar( value & displayMask ? '1' : '0' );
66       value <<= 1; /* shift value left by 1 */
67
68       if ( c % 8 == 0 ) { /* output a space after 8 bits */
69          putchar( ' ' );
70       } /* end if */
71
72    } /* end for */
73
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
74    putchar( '\n' );
75 } /* end function displayBits */
```

```
The result of combining the following
     65535 = 00000000 00000000 11111111 11111111
         1 = 00000000 00000000 00000000 00000001
using the bitwise AND operator & is
         1 = 00000000 00000000 00000000 00000001

The result of combining the following
        15 = 00000000 00000000 00000000 00001111
       241 = 00000000 00000000 00000000 11110001
using the bitwise inclusive OR operator | is
       255 = 00000000 00000000 00000000 11111111

The result of combining the following
       139 = 00000000 00000000 00000000 10001011
       199 = 00000000 00000000 00000000 11000111
using the bitwise exclusive OR operator ^ is
        76 = 00000000 00000000 00000000 01001100

The one's complement of
     21845 = 00000000 00000000 01010101 01010101
is
4294945450 = 11111111 11111111 10101010 10101010
```


fig10_09.c (4 of 4)

Program Output

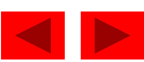
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

10.9 Bitwise Operators

Bit 1	Bit 2	Bit 1 | Bit 2
0	0	0
1	0	1
0	1	1
1	1	1

Fig. 10.11 Results of combining two bits with the bitwise inclusive OR operator |.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

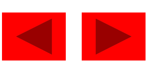


10.9 Bitwise Operators

Bit 1	Bit 2	Bit 1 ^ Bit 2
0	0	0
1	0	1
0	1	1
1	1	0

Fig. 10.12 Results of combining two bits with the bitwise exclusive OR operator ^.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



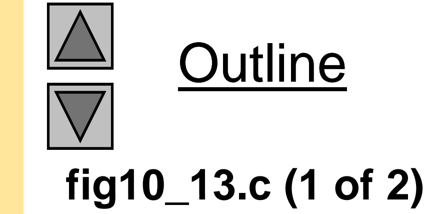


fig10_13.c (1 of 2)

```
1  /* Fig. 10.13: fig10_13.c
2     Using the bitwise shift operators */
3  #include <stdio.h>
4
5  void displayBits( unsigned value ); /* prototype */
6
7  int main()
8  {
9     unsigned number1 = 960; /* initialize number1 */
10
11    /* demonstrate bitwise left shift */
12    printf( "\nThe result of left shifting\n" );
13    displayBits( number1 );
14    printf( "8 bit positions using the " );
15    printf( "left shift operator << is\n" );
16    displayBits( number1 << 8 );
17
18    /* demonstrate bitwise right shift */
19    printf( "\nThe result of right shifting\n" );
20    displayBits( number1 );
21    printf( "8 bit positions using the " );
22    printf( "right shift operator >> is\n" );
23    displayBits( number1 >> 8 );
24
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

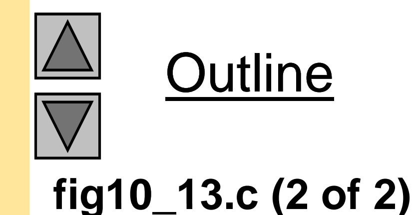
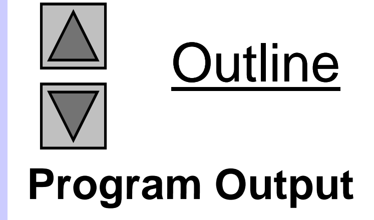


fig10_13.c (2 of 2)

```
25    return 0; /* indicates successful termination */
26
27 } /* end main */
28
29 /* display bits of an unsigned integer value */
30 void displayBits( unsigned value )
31 {
32    unsigned c; /* counter */
33
34    /* declare displayMask and left shift 31 bits */
35    unsigned displayMask = 1 << 31;
36
37    printf( "%7u = ", value );
38
39    /* loop through bits */
40    for ( c = 1; c <= 32; c++ ) {
41       putchar( value & displayMask ? '1' : '0' );
42       value <<= 1; /* shift value left by 1 */
43
44       if ( c % 8 == 0 ) { /* output a space after 8 bits */
45          putchar( ' ' );
46       } /* end if */
47
48    } /* end for */
49
50    putchar( '\n' );
51 } /* end function displayBits */
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.




```
The result of left shifting
    960 = 00000000 00000000 00000011 11000000
8 bit positions using the left shift operator << is
 245760 = 00000000 00000011 11000000 00000000

The result of right shifting
    960 = 00000000 00000000 00000011 11000000
8 bit positions using the right shift operator >> is
      3 = 00000000 00000000 00000000 00000011
```

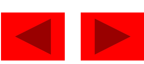
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

10.9 Bitwise Operators

<table>
<tr><th>Bitwise assignment operators</th><th></th></tr>
<tr><td>&=</td><td>Bitwise AND assignment operator.</td></tr>
<tr><td>|=</td><td>Bitwise inclusive OR assignment operator.</td></tr>
<tr><td>^=</td><td>Bitwise exclusive OR assignment operator.</td></tr>
<tr><td><<=</td><td>Left-shift assignment operator.</td></tr>
<tr><td>>>=</td><td>Right-shift assignment operator.</td></tr>
</table>

Fig. 10.14 The bitwise assignment operators.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

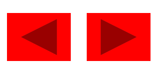


10.9 Bitwise Operators

<table>
<tr><th>Operator</th><th>Associativity</th><th>Type</th></tr>
<tr><td>() [] . -></td><td>left to right</td><td>Highest</td></tr>
<tr><td>+ - ++ -- ! & * ~ sizeof (type)</td><td>right to left</td><td>Unary</td></tr>
<tr><td>* / %</td><td>left to right</td><td>multiplicative</td></tr>
<tr><td>+ -</td><td>left to right</td><td>additive</td></tr>
<tr><td><< >></td><td>left to right</td><td>shifting</td></tr>
<tr><td>< <= > >=</td><td>left to right</td><td>relational</td></tr>
<tr><td>== !=</td><td>left to right</td><td>equality</td></tr>
<tr><td>&</td><td>left to right</td><td>bitwise AND</td></tr>
<tr><td>^</td><td>left to right</td><td>bitwise OR</td></tr>
<tr><td>|</td><td>left to right</td><td>bitwise OR</td></tr>
<tr><td>&&</td><td>left to right</td><td>logical AND</td></tr>
<tr><td>||</td><td>left to right</td><td>logical OR</td></tr>
<tr><td>?:</td><td>right to left</td><td>conditional</td></tr>
<tr><td>= += -= *= /= &= |= ^= <<= >>= %=</td><td>right to left</td><td>assignment</td></tr>
<tr><td>,</td><td>left to right</td><td>comma</td></tr>
</table>

Fig. 10.15 Operator precedence and associativity.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

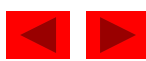


10.10 Bit Fields

- Bit field
  - Member of a structure whose size (in bits) has been specified
  - Enable better memory utilization
  - Must be defined as `int` or `unsigned`
  - Cannot access individual bits
- Defining bit fields
  - Follow `unsigned` or `int` member with a colon (`:`) and an integer constant representing the width of the field
  - Example:

```
struct BitCard {
   unsigned face : 4;
   unsigned suit : 2;
   unsigned color : 1;
};
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



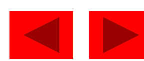
10.10 Bit Fields

- Unnamed bit field
  - Field used as padding in the structure
  - Nothing may be stored in the bits

```
struct Example {
    unsigned a : 13;
    unsigned   : 3;
    unsigned b : 4;
}
```

  - Unnamed bit field with zero width aligns next bit field to a new storage unit boundary

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Outline

fig10_16.c (1 of 3)

```
1  /* Fig. 10.16: fig10_16.c
2     Representing cards with bit fields in a struct */
3
4  #include <stdio.h>
5
6  /* bitCard structure definition with bit fields */
7  struct bitCard {
8     unsigned face : 4;  /* 4 bits; 0-15 */
9     unsigned suit : 2;  /* 2 bits; 0-3 */
10    unsigned color : 1; /* 1 bit; 0-1 */
11 }; /* end struct bitCard */
12
13 typedef struct bitCard Card;
14
15 void fillDeck( Card * const wDeck );   /* prototype */
16 void deal( const Card * const wDeck ); /* prototype */
17
18 int main()
19 {
20    Card deck[ 52 ]; /* create array of Cards */
21
22    fillDeck( deck );
23    deal( deck );
24
25    return 0; /* indicates successful termination */
26
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Outline

fig10_16.c (2 of 3)

```
27 } /* end main */
28
29 /* initialize Cards */
30 void fillDeck( Card * const wDeck )
31 {
32    int i; /* counter */
33
34    /* loop through wDeck */
35    for ( i = 0; i <= 51; i++ ) {
36       wDeck[ i ].face = i % 13;
37       wDeck[ i ].suit = i / 13;
38       wDeck[ i ].color = i / 26;
39    } /* end for */
40
41 } /* end function fillDeck */
42
43 /* output cards in two column format; cards 0-25 subscripted with
44    k1 (column 1); cards 26-51 subscripted k2 (column 2) */
45 void deal( const Card * const wDeck )
46 {
47    int k1; /* subscripts 0-25 */
48    int k2; /* subscripts 26-51 */
49
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

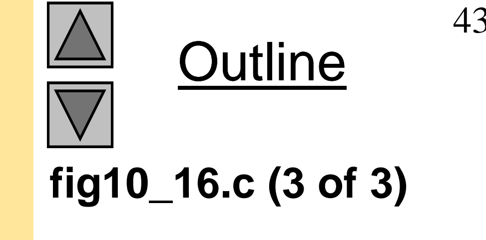
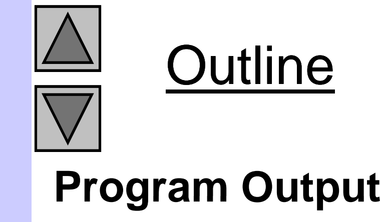


fig10_16.c (3 of 3)

```
50    /* loop through wDeck */
51    for ( k1 = 0, k2 = k1 + 26; k1 <= 25; k1++, k2++ ) {
52       printf( "Card:%3d  Suit:%2d  Color:%2d   ",
53          wDeck[ k1 ].face, wDeck[ k1 ].suit, wDeck[ k1 ].color );
54       printf( "Card:%3d  Suit:%2d  Color:%2d\n",
55          wDeck[ k2 ].face, wDeck[ k2 ].suit, wDeck[ k2 ].color );
56    } /* end for */
57
58 } /* end function deal */
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Program Output

```
Card:  0  Suit: 0  Color: 0   Card:  0  Suit: 2  Color: 1
Card:  1  Suit: 0  Color: 0   Card:  1  Suit: 2  Color: 1
Card:  2  Suit: 0  Color: 0   Card:  2  Suit: 2  Color: 1
Card:  3  Suit: 0  Color: 0   Card:  3  Suit: 2  Color: 1
Card:  4  Suit: 0  Color: 0   Card:  4  Suit: 2  Color: 1
Card:  5  Suit: 0  Color: 0   Card:  5  Suit: 2  Color: 1
Card:  6  Suit: 0  Color: 0   Card:  6  Suit: 2  Color: 1
Card:  7  Suit: 0  Color: 0   Card:  7  Suit: 2  Color: 1
Card:  8  Suit: 0  Color: 0   Card:  8  Suit: 2  Color: 1
Card:  9  Suit: 0  Color: 0   Card:  9  Suit: 2  Color: 1
Card: 10  Suit: 0  Color: 0   Card: 10  Suit: 2  Color: 1
Card: 11  Suit: 0  Color: 0   Card: 11  Suit: 2  Color: 1
Card: 12  Suit: 0  Color: 0   Card: 12  Suit: 2  Color: 1
Card:  0  Suit: 1  Color: 0   Card:  0  Suit: 3  Color: 1
Card:  1  Suit: 1  Color: 0   Card:  1  Suit: 3  Color: 1
Card:  2  Suit: 1  Color: 0   Card:  2  Suit: 3  Color: 1
Card:  3  Suit: 1  Color: 0   Card:  3  Suit: 3  Color: 1
Card:  4  Suit: 1  Color: 0   Card:  4  Suit: 3  Color: 1
Card:  5  Suit: 1  Color: 0   Card:  5  Suit: 3  Color: 1
Card:  6  Suit: 1  Color: 0   Card:  6  Suit: 3  Color: 1
Card:  7  Suit: 1  Color: 0   Card:  7  Suit: 3  Color: 1
Card:  8  Suit: 1  Color: 0   Card:  8  Suit: 3  Color: 1
Card:  9  Suit: 1  Color: 0   Card:  9  Suit: 3  Color: 1
Card: 10  Suit: 1  Color: 0   Card: 10  Suit: 3  Color: 1
Card: 11  Suit: 1  Color: 0   Card: 11  Suit: 3  Color: 1
Card: 12  Suit: 1  Color: 0   Card: 12  Suit: 3  Color: 1
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

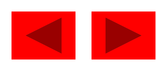
10.11 Enumeration Constants

- Enumeration
  - Set of integer constants represented by identifiers
  - Enumeration constants are like symbolic constants whose values are automatically set
    - Values start at 0 and are incremented by 1
    - Values can be set explicitly with =
    - Need unique constant names
  - Example:

```
enum Months { JAN = 1, FEB, MAR, APR, MAY, JUN, JUL,
  AUG, SEP, OCT, NOV, DEC};
```

    - Creates a new type enum Months in which the identifiers are set to the integers 1 to 12
  - Enumeration variables can only assume their enumeration constant values (not the integer representations)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Outline

fig10_18.c

```
1  /* Fig. 10.18: fig10_18.c
2     Using an enumeration type */
3  #include <stdio.h>
4
5  /* enumeration constants represent months of the year */
6  enum months { JAN = 1, FEB, MAR, APR, MAY, JUN,
7                JUL, AUG, SEP, OCT, NOV, DEC };
8
9  int main()
10 {
11    enum months month; /* can contain any of the 12 months */
12
13    /* initialize array of pointers */
14    const char *monthName[] = { "", "January", "February", "March",
15       "April", "May", "June", "July", "August", "September", "October",
16       "November", "December" };
17
18    /* loop through months */
19    for ( month = JAN; month <= DEC; month++ ) {
20       printf( "%2d%11s\n", month, monthName[ month ] );
21    } /* end for */
22
23    return 0; /* indicates successful termination */
24 } /* end main */
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
 1    January
 2   February
 3      March
 4      April
 5        May
 6       June
 7       July
 8     August
 9  September
10    October
11   November
12   December
```

Outline

Program Output

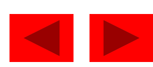
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Chapter 11 – File Processing

Outline

11.1 Introduction
11.2 The Data Hierarchy
11.3 Files and Streams
11.4 Creating a Sequential Access File
11.5 Reading Data from a Sequential Access File
11.6 Random Access Files
11.7 Creating a Randomly Accessed File
11.8 Writing Data Randomly to a Randomly Accessed File
11.9 Reading Data Randomly from a Randomly Accessed File
11.10 Case Study: A Transaction-Processing Program

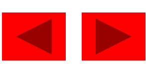
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Objectives

- In this chapter, you will learn:
  - To be able to create, read, write and update files.
  - To become familiar with sequential access file processing.
  - To become familiar with random-access file processing.

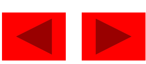
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



11.1 Introduction

- Data files
  - Can be created, updated, and processed by C programs
  - Are used for permanent storage of large amounts of data
    - Storage of data in variables and arrays is only temporary

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



11.2 The Data Hierarchy

- Data Hierarchy:
  - Bit – smallest data item
    - Value of `0` or `1`
  - Byte – 8 bits
    - Used to store a character
      - Decimal digits, letters, and special symbols
  - Field – group of characters conveying meaning
    - Example: your name
  - Record – group of related fields
    - Represented by a `struct` or a `class`
    - Example: In a payroll system, a record for a particular employee that contained his/her identification number, name, address, etc.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

11.2 The Data Hierarchy

- Data Hierarchy (continued):
  - File – group of related records
    - Example: payroll file
  - Database – group of related files

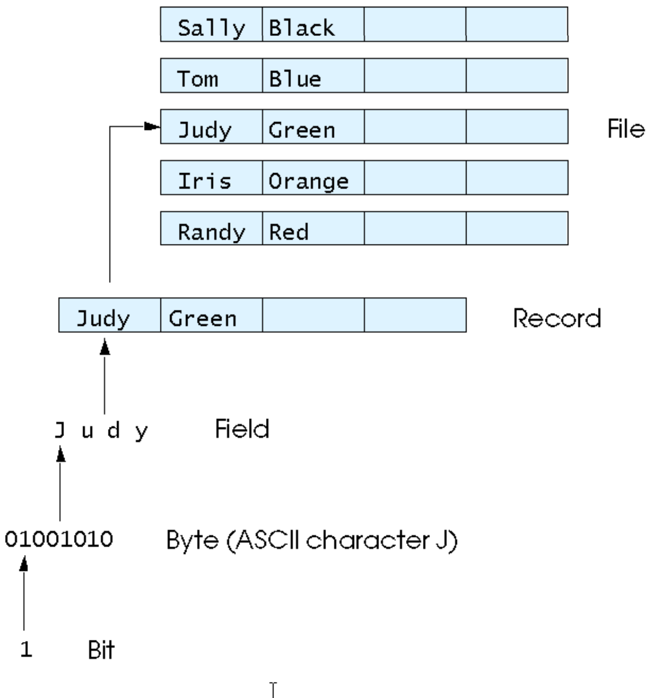
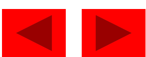


Fig. 11.1 The data hierarchy.

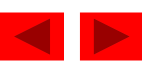
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



11.2 The Data Hierarchy

- Data files
  - Record key
    - Identifies a record to facilitate the retrieval of specific records from a file
  - Sequential file
    - Records typically sorted by key

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



11.3 Files and Streams

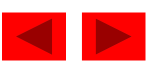
- C views each file as a sequence of bytes
  - File ends with the *end-of-file marker*
    - Or, file ends at a specified byte
- Stream created when a file is opened
  - Provide communication channel between files and programs
  - Opening a file returns a pointer to a `FILE` structure
    - Example file pointers:
    - `stdin` - standard input (keyboard)
    - `stdout` - standard output (screen)
    - `stderr` - standard error (screen)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

11.3 Files and Streams

- `FILE` structure
  - File descriptor
    - Index into operating system array called the open file table
  - File Control Block (FCB)
    - Found in every array element, system uses it to administer the file

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



11.3 Files and Streams

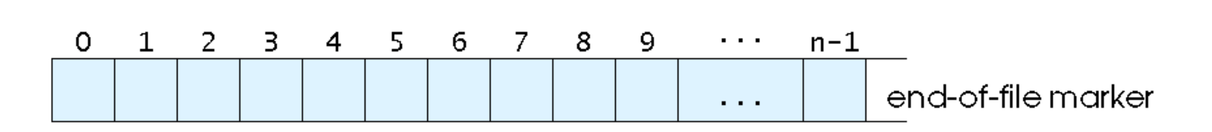


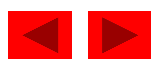
Fig. 11.2 C's view of a file of *n* bytes.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

11.3 Files and Streams

- Read/Write functions in standard library
  - `fgetc`
    - Reads one character from a file
    - Takes a `FILE` pointer as an argument
    - `fgetc( stdin )` equivalent to `getchar()`
  - `fputc`
    - Writes one character to a file
    - Takes a `FILE` pointer and a character to write as an argument
    - `fputc( 'a', stdout )` equivalent to `putchar( 'a' )`
  - `fgets`
    - Reads a line from a file
  - `fputs`
    - Writes a line to a file
  - `fscanf` / `fprintf`
    - File processing equivalents of `scanf` and `printf`

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



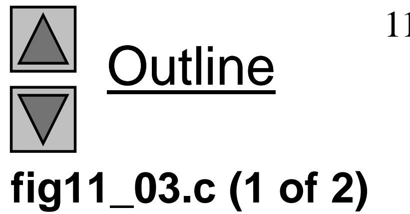


fig11_03.c (1 of 2)

```
1  /* Fig. 11.3: fig11_03.c
2     Create a sequential file */
3  #include <stdio.h>
4
5  int main()
6  {
7     int account;     /* account number */
8     char name[ 30 ]; /* account name */
9     double balance;  /* account balance */
10
11    FILE *cfPtr;      /* cfPtr = clients.dat file pointer */
12
13    /* fopen opens file. Exit program if unable to create file  */
14    if ( ( cfPtr = fopen( "clients.dat", "w" ) ) == NULL ) {
15       printf( "File could not be opened\n" );
16    } /* end if */
17    else {
18       printf( "Enter the account, name, and balance.\n" );
19       printf( "Enter EOF to end input.\n" );
20       printf( "? " );
21       scanf( "%d%s%lf", &account, name, &balance );
22
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Outline

fig11_03.c (2 of 2)

```
23       /* write account, name and balance into file with fprintf */
24       while ( !feof( stdin ) ) {
25          fprintf( cfPtr, "%d %s %.2f\n", account, name, balance );
26          printf( "? " );
27          scanf( "%d%s%lf", &account, name, &balance );
28       } /* end while */
29
30       fclose( cfPtr ); /* fclose closes file */
31    } /* end else */
32
33    return 0; /* indicates successful termination */
34
35 } /* end main */
```

Program Output

```
Enter the account, name, and balance.
Enter EOF to end input.
? 100 Jones 24.98
? 200 Doe 345.67
? 300 White 0.00
? 400 Stone -42.16
? 500 Rich 224.62
? ^Z
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

11.4 Creating a Sequential Access File

- C imposes no file structure
  - No notion of records in a file
  - Programmer must provide file structure
- Creating a File
  - `FILE *cfPtr;`
    - Creates a `FILE` pointer called `cfPtr`
  - `cfPtr = fopen("clients.dat", "w");`
    - Function `fopen` returns a `FILE` pointer to file specified
    - Takes two arguments – file to open and file open mode
    - If open fails, `NULL` returned

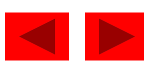
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

11.4 Creating a Sequential Access File

Computer system	Key combination
UNIX systems	*<return> <ctrl> d*
IBM PC and compatibles	*<ctrl> z*
Macintosh	*<ctrl> d*

Fig. 11.4 End-of-file key combinations for various popular computer systems.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

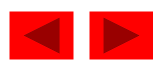


11.4 Creating a Sequential Access File

- `fprintf`
  - Used to print to a file
  - Like `printf`, except first argument is a `FILE` pointer (pointer to the file you want to print in)
- `feof(` *`FILE`* *pointer* `)`
  - Returns true if end-of-file indicator (no more data to process) is set for the specified file
- `fclose(` *`FILE`* *pointer* `)`
  - Closes specified file
  - Performed automatically when program ends
  - Good practice to close files explicitly

- Details
  - Programs may process no files, one file, or many files
  - Each file must have a unique name and should have its own pointer

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

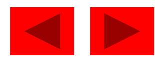


11.4 Creating a Sequential Access File

Mode	Description
r	Open a file for reading.
w	Create a file for writing. If the file already exists, discard the current contents.
a	Append; open or create a file for writing at end of file.
r+	Open a file for update (reading and writing).
w+	Create a file for update. If the file already exists, discard the current contents.
a+	Append; open or create a file for update; writing is done at the end of the file.
rb	Open a file for reading in binary mode.
wb	Create a file for writing in binary mode. If the file already exists, discard the current contents.
ab	Append; open or create a file for writing at end of file in binary mode.
rb+	Open a file for update (reading and writing) in binary mode.
wb+	Create a file for update in binary mode. If the file already exists, discard the current contents.
ab+	Append; open or create a file for update in binary mode; writing is done at the end of the file.

Fig. 11.6 File open modes.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



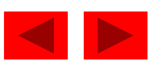
11.5 Reading Data from a Sequential Access File

- Reading a sequential access file
  - Create a `FILE` pointer, link it to the file to read

    ```
    cfPtr = fopen( "clients.dat", "r" );
    ```
  - Use `fscanf` to read from the file
    - Like `scanf`, except first argument is a `FILE` pointer

    ```
    fscanf( cfPtr, "%d%s%f", &accounnt, name, &balance );
    ```
  - Data read from beginning to end
  - File position pointer
    - Indicates number of next byte to be read / written
    - Not really a pointer, but an integer value (specifies byte location)
    - Also called byte offset
  - `rewind( cfPtr )`
    - Repositions file position pointer to beginning of file (byte `0`)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



```
1  /* Fig. 11.7: fig11_07.c
2     Reading and printing a sequential file */
3  #include <stdio.h>
4
5  int main()
6  {
7     int account;     /* account number */
8     char name[ 30 ]; /* account name */
9     double balance;  /* account balance */
10
11    FILE *cfPtr;     /* cfPtr = clients.dat file pointer */
12
13    /* fopen opens file; exits program if file cannot be opened */
14    if ( ( cfPtr = fopen( "clients.dat", "r" ) ) == NULL ) {
15       printf( "File could not be opened\n" );
16    } /* end if */
17    else { /* read account, name and balance from file */
18       printf( "%-10s%-13s%s\n", "Account", "Name", "Balance" );
19       fscanf( cfPtr, "%d%s%lf", &account, name, &balance );
20
21       /* while not end of file */
22       while ( !feof( cfPtr ) ) {
23          printf( "%-10d%-13s%7.2f\n", account, name, balance );
24          fscanf( cfPtr, "%d%s%lf", &account, name, &balance );
25       } /* end while */
26
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

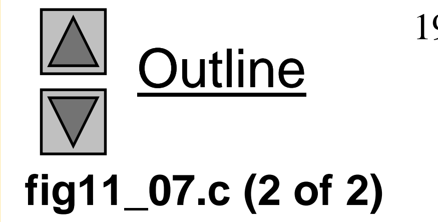
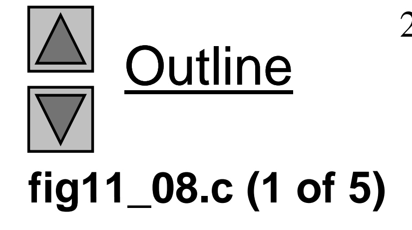


fig11_07.c (2 of 2)

```
27       fclose( cfPtr ); /* fclose closes the file */
28    } /* end else */
29
30    return 0; /* indicates successful termination */
31
32 } /* end main */
```

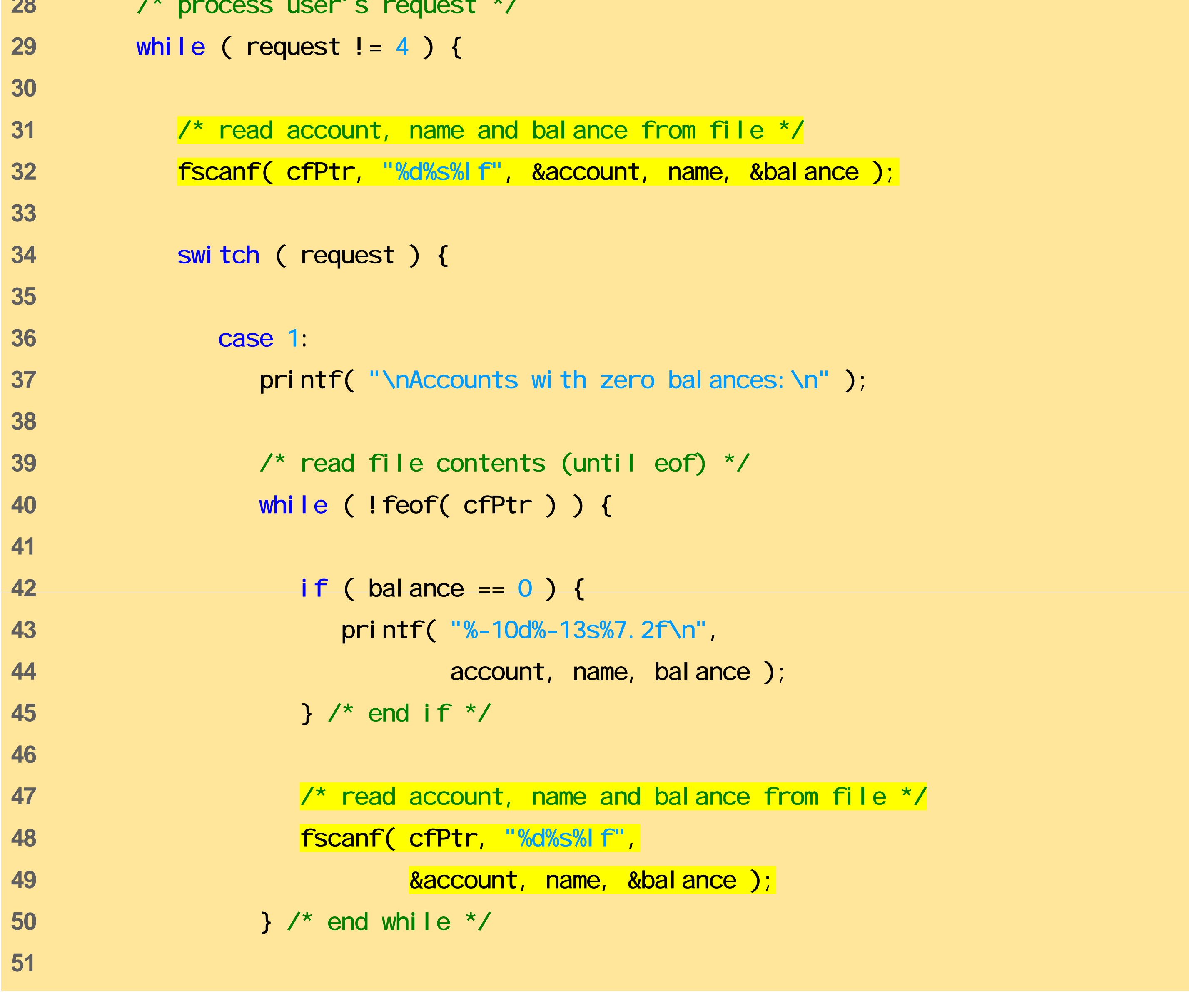
```
Account    Name          Balance
100        Jones           24.98
200        Doe            345.67
300        White            0.00
400        Stone          -42.16
500        Rich           224.62
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.




```
1  /* Fig. 11.8: fig11_08.c
2     Credit inquiry program */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main()
7  {
8     int request;      /* request number */
9     int account;      /* account number */
10    double balance;  /* account balance */
11    char name[ 30 ]; /* account name */
12    FILE *cfPtr;     /* clients.dat file pointer */
13
14    /* fopen opens the file; exits program if file cannot be opened */
15    if ( ( cfPtr = fopen( "clients.dat", "r" ) ) == NULL ) {
16       printf( "File could not be opened\n" );
17    } /* end if */
18    else {
19
20       /* display request options */
21       printf( "Enter request\n"
22              " 1 - List accounts with zero balances\n"
23              " 2 - List accounts with credit balances\n"
24              " 3 - List accounts with debit balances\n"
25              " 4 - End of run\n? " );
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



```
26      scanf( "%d", &request );
27
28      /* process user's request */
29      while ( request != 4 ) {
30
31         /* read account, name and balance from file */
32         fscanf( cfPtr, "%d%s%lf", &account, name, &balance );
33
34         switch ( request ) {
35
36            case 1:
37               printf( "\nAccounts with zero balances:\n" );
38
39               /* read file contents (until eof) */
40               while ( !feof( cfPtr ) ) {
41
42                  if ( balance == 0 ) {
43                     printf( "%-10d%-13s%7.2f\n",
44                             account, name, balance );
45                  } /* end if */
46
47                  /* read account, name and balance from file */
48                  fscanf( cfPtr, "%d%s%lf",
49                          &account, name, &balance );
50               } /* end while */
51
```


© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

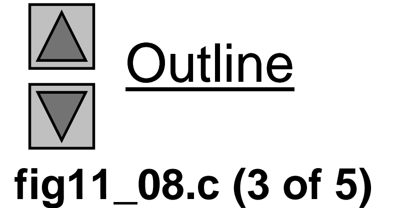


fig11_08.c (3 of 5)

```
52              break;
53
54           case 2:
55              printf( "\nAccounts with credit balances:\n" );
56
57              /* read file contents (until eof) */
58              while ( !feof( cfPtr ) ) {
59
60                 if ( balance < 0 ) {
61                    printf( "%-10d%-13s%7.2f\n",
62                            account, name, balance );
63                 } /* end if */
64
65                 /* read account, name and balance from file */
66                 fscanf( cfPtr, "%d%s%lf",
67                         &account, name, &balance );
68              } /* end while */
69
70              break;
71
72           case 3:
73              printf( "\nAccounts with debit balances:\n" );
74
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

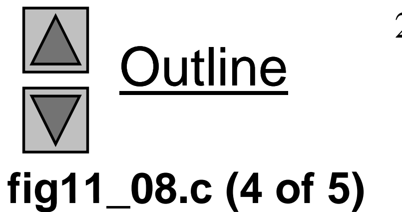


fig11_08.c (4 of 5)

```
75                /* read file contents (until eof) */
76                while ( !feof( cfPtr ) ) {
77
78                   if ( balance > 0 ) {
79                      printf( "%-10d%-13s%7.2f\n",
80                              account, name, balance );
81                   } /* end if */
82
83                   /* read account, name and balance from file */
84                   fscanf( cfPtr, "%d%s%lf",
85                           &account, name, &balance );
86                } /* end while */
87
88                break;
89
90          } /* end switch */
91
92          rewind( cfPtr ); /* return cfPtr to beginning of file */
93
94          printf( "\n? " );
95          scanf( "%d", &request );
96       } /* end while */
97
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
98        printf( "End of run.\n" );
99        fclose( cfPtr ); /* fclose closes the file */
100    } /* end else */
101
102    return 0; /* indicates successful termination */
103
104 } /* end main */
```

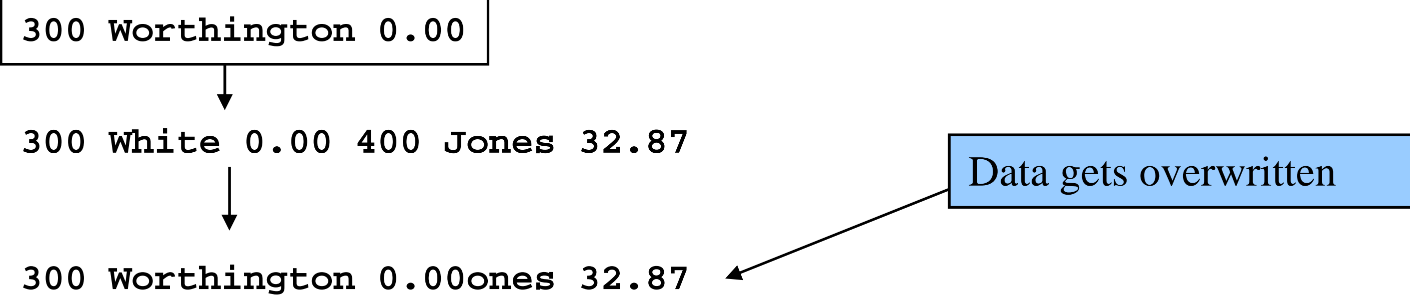
```
Enter request
 1 - List accounts with zero balances
 2 - List accounts with credit balances
 3 - List accounts with debit balances
 4 - End of run
? 1

Accounts with zero balances:
300       White            0.00

? 2

Accounts with credit balances:
400       Stone          -42.16

? 3

Accounts with debit balances:
100       Jones           24.98
200       Doe            345.67
500       Rich           224.62

? 4
End of run.
```


Program Output

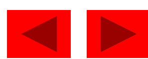
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

11.5 Reading Data from a Sequential Access File

- Sequential access file
  - Cannot be modified without the risk of destroying other data
  - Fields can vary in size
    - Different representation in files and screen than internal representation
    - `1`, `34`, `-890` are all `int`s, but have different sizes on disk

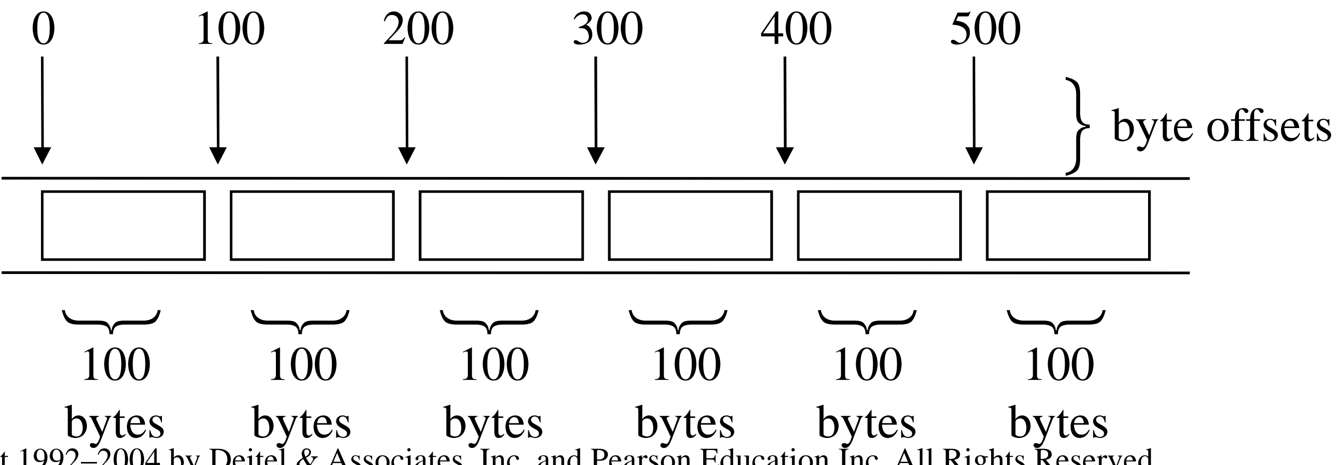


© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

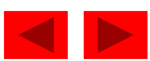


11.6 Random-Access Files

- Random access files
  - Access individual records without searching through other records
  - Instant access to records in a file
  - Data can be inserted without destroying other data
  - Data previously stored can be updated or deleted without overwriting
- Implemented using fixed length records
  - Sequential files do not have fixed length records



© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



11.7 Creating a Randomly Accessed File

- Data in random access files
  - Unformatted (stored as "raw bytes")
    - All data of the same type (**`int`**s, for example) uses the same amount of memory
    - All records of the same type have a fixed length
    - Data not human readable

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

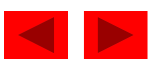
11.7 Creating a Randomly Accessed File

- Unformatted I/O functions
  - `fwrite`
    - Transfer bytes from a location in memory to a file
  - `fread`
    - Transfer bytes from a file to a location in memory
  - Example:

```
fwrite( &number, sizeof( int ), 1, myPtr );
```

    - `&number` – Location to transfer bytes from
    - `sizeof( int )` – Number of bytes to transfer
    - `1` – For arrays, number of elements to transfer
      - In this case, "one element" of an array is being transferred
    - `myPtr` – File to transfer to or from

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



11.7 Creating a Randomly Accessed File

- Writing `structs`

```
fwrite( &myObject, sizeof (struct myStruct), 1,
  myPtr );
```

  - `sizeof` – returns size in bytes of object in parentheses
- To write several array elements
  - Pointer to array as first argument
  - Number of elements to write as third argument

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Outline

fig11_11.c (1 of 2)

```
1  /* Fig. 11.11: fig11_11.c
2     Creating a randomly accessed file sequentially */
3  #include <stdio.h>
4
5  /* clientData structure definition */
6  struct clientData {
7     int acctNum;          /* account number */
8     char lastName[ 15 ];  /* account last name */
9     char firstName[ 10 ]; /* account first name */
10    double balance;       /* account balance */
11 }; /* end structure clientData */
12
13 int main()
14 {
15    int i; /* counter */
16
17    /* create clientData with no information */
18    struct clientData blankClient = { 0, "", "", 0.0 };
19
20    FILE *cfPtr; /* credit.dat file pointer */
21
22    /* fopen opens the file; exits if file cannot be opened */
23    if ( ( cfPtr = fopen( "credit.dat", "wb" ) ) == NULL ) {
24       printf( "File could not be opened.\n" );
25    } /* end if */
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

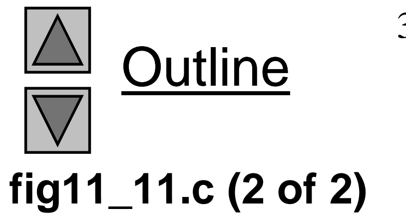


fig11_11.c (2 of 2)

```
26    else {
27
28       /* output 100 blank records to file */
29       for ( i = 1; i <= 100; i++ ) {
30          fwrite( &blankClient, sizeof( struct clientData ), 1, cfPtr );
31       } /* end for */
32
33       fclose ( cfPtr ); /* fclose closes the file */
34    } /* end else */
35
36    return 0; /* indicates successful termination */
37
38 } /* end main */
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

11.8 Writing Data Randomly to a Randomly Accessed File

- `fseek`
  - Sets file position pointer to a specific position
  - `fseek(` *pointer*, *offset*, *symbolic_constant* `);`
    - *pointer* – pointer to file
    - *offset* – file position pointer (0 is first location)
    - *symbolic_constant* – specifies where in file we are reading from
    - `SEEK_SET` – seek starts at beginning of file
    - `SEEK_CUR` – seek starts at current location in file
    - `SEEK_END` – seek starts at end of file

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

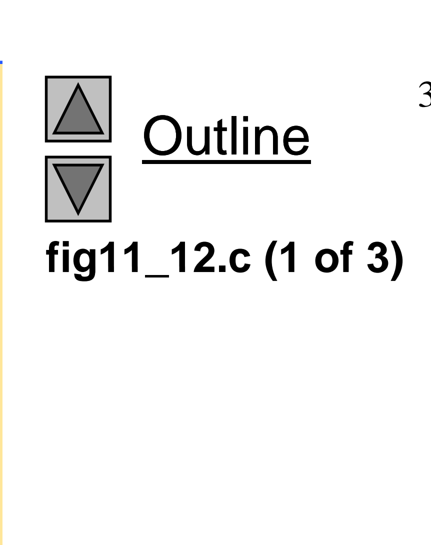


fig11_12.c (1 of 3)

```
1  /* Fig. 11.12: fig11_12.c
2     Writing to a random access file */
3  #include <stdio.h>
4
5  /* clientData structure definition */
6  struct clientData {
7     int acctNum;           /* account number */
8     char lastName[ 15 ];   /* account last name */
9     char firstName[ 10 ]; /* account first name */
10     double balance;        /* account balance */
11 }; /* end structure clientData */
12
13 int main()
14 {
15    FILE *cfPtr; /* credit.dat file pointer */
16
17    /* create clientData with no information */
18    struct clientData client = { 0, "", "", 0.0 };
19
20    /* fopen opens the file; exits if file cannot be opened */
21    if ( ( cfPtr = fopen( "credit.dat", "rb+" ) ) == NULL ) {
22       printf( "File could not be opened.\n" );
23    } /* end if */
24    else {
25
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Outline

fig11_12.c (2 of 3)

```
26      /* require user to specify account number */
27      printf( "Enter account number"
28              " ( 1 to 100, 0 to end input )\n? " );
29      scanf( "%d", &client.acctNum );
30
31      /* user enters information, which is copied into file */
32      while ( client.acctNum != 0 ) {
33
34         /* user enters last name, first name and balance */
35         printf( "Enter lastname, firstname, balance\n? " );
36
37         /* set record lastName, firstName and balance value */
38         fscanf( stdin, "%s%s%lf", client.lastName,
39                 client.firstName, &client.balance );
40
41         /* seek position in file of user-specified record */
42         fseek( cfPtr, ( client.acctNum - 1 ) *
43                sizeof( struct clientData ), SEEK_SET );
44
45         /* write user-specified information in file */
46         fwrite( &client, sizeof( struct clientData ), 1, cfPtr );
47
48         /* enable user to specify another account number */
49         printf( "Enter account number\n? " );
50         scanf( "%d", &client.acctNum );
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

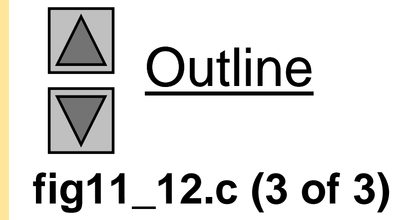


fig11_12.c (3 of 3)

```
51       } /* end while */
52
53       fclose( cfPtr ); /* fclose closes the file */
54    } /* end else */
55
56    return 0; /* indicates successful termination */
57
58 } /* end main */
```

Program Output

```
Enter account number ( 1 to 100, 0 to end input )
? 37
Enter lastname, firstname, balance
? Barker Doug 0.00
Enter account number
? 29
Enter lastname, firstname, balance
? Brown Nancy -24.54
Enter account number
? 96
Enter lastname, firstname, balance
? Stone Sam 34.98
Enter account number
? 88
Enter lastname, firstname, balance
? Smith Dave 258.34
Enter account number
? 33
Enter lastname, firstname, balance
? Dunn Stacey 314.33
Enter account number
? 0
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

11.8 Writing Data Randomly to a Randomly Accessed File

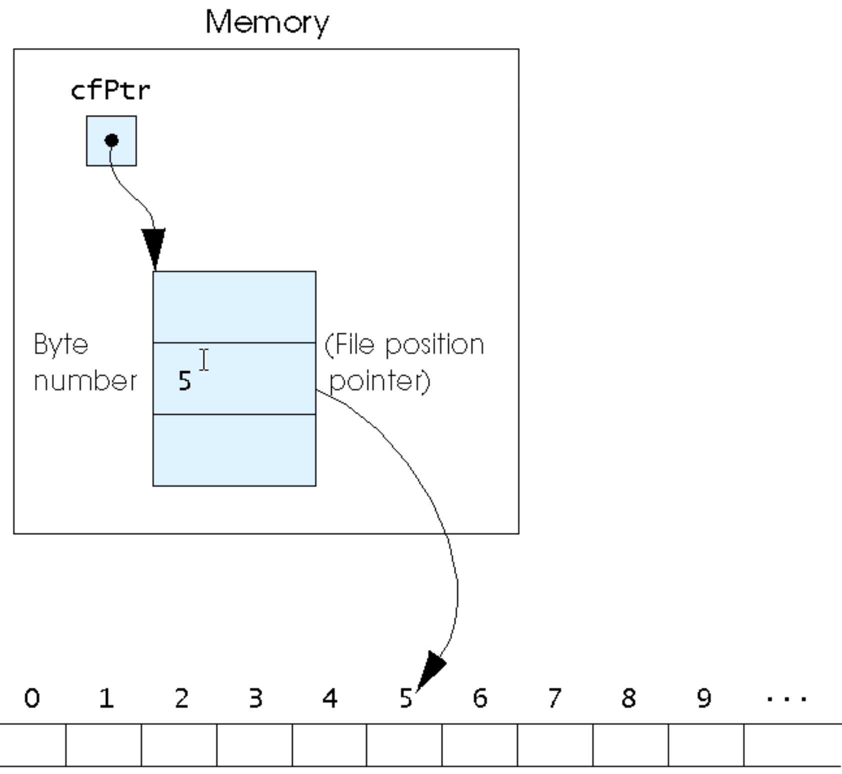


Fig. 11.14 The file position pointer indicating an offset of 5 bytes from the beginning of the file.

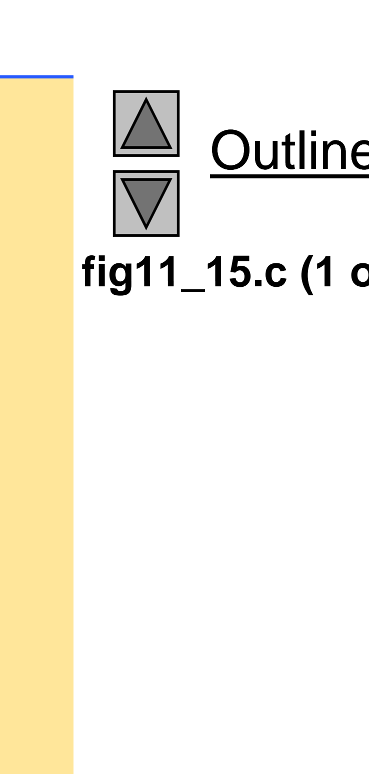
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

11.9 Reading Data Randomly from a Randomly Accessed File

- `fread`
  - Reads a specified number of bytes from a file into memory

    ```
    fread( &client, sizeof (struct clientData), 1,
      myPtr );
    ```
  - Can read several fixed-size array elements
    - Provide pointer to array
    - Indicate number of elements to read
  - To read multiple elements, specify in third argument

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.




```
1  /* Fig. 11.15: fig11_15.c
2     Reading a random access file sequentially */
3  #include <stdio.h>
4
5  /* clientData structure definition */
6  struct clientData {
7     int acctNum;          /* account number */
8     char lastName[ 15 ];  /* account last name */
9     char firstName[ 10 ]; /* account first name */
10    double balance;       /* account balance */
11 }; /* end structure clientData */
12
13 int main()
14 {
15    FILE *cfPtr; /* credit.dat file pointer */
16
17    /* create clientData with no information */
18    struct clientData client = { 0, "", "", 0.0 };
19
20    /* fopen opens the file; exits if file cannot be opened */
21    if ( ( cfPtr = fopen( "credit.dat", "rb" ) ) == NULL ) {
22       printf( "File could not be opened.\n" );
23    } /* end if */
```

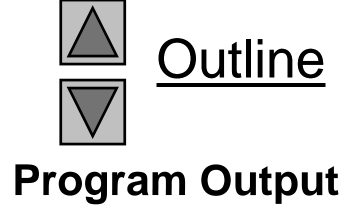
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Outline

fig11_15.c (2 of 2)

```
24     else {
25        printf( "%-6s%-16s%-11s%10s\n", "Acct", "Last Name",
26                "First Name", "Balance" );
27
28        /* read all records from file (until eof) */
29        while ( !feof( cfPtr ) ) {
30           fread( &client, sizeof( struct clientData ), 1, cfPtr );
31
32           /* display record */
33           if ( client.acctNum != 0 ) {
34              printf( "%-6d%-16s%-11s%10.2f\n",
35                      client.acctNum, client.lastName,
36                      client.firstName, client.balance );
37           } /* end if */
38
39        } /* end while */
40
41        fclose( cfPtr ); /* fclose closes the file */
42     } /* end else */
43
44     return 0; /* indicates successful termination */
45
46  } /* end main */
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Program Output

```
Acct  Last Name       First Name    Balance
29    Brown           Nancy          -24.54
33    Dunn            Stacey         314.33
37    Barker          Doug             0.00
88    Smith           Dave           258.34
96    Stone           Sam             34.98
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

11.10 Case Study: A Transaction Processing Program

- This program
  - Demonstrates using random access files to achieve instant access processing of a bank's account information
- We will
  - Update existing accounts
  - Add new accounts
  - Delete accounts
  - Store a formatted listing of all accounts in a text file

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

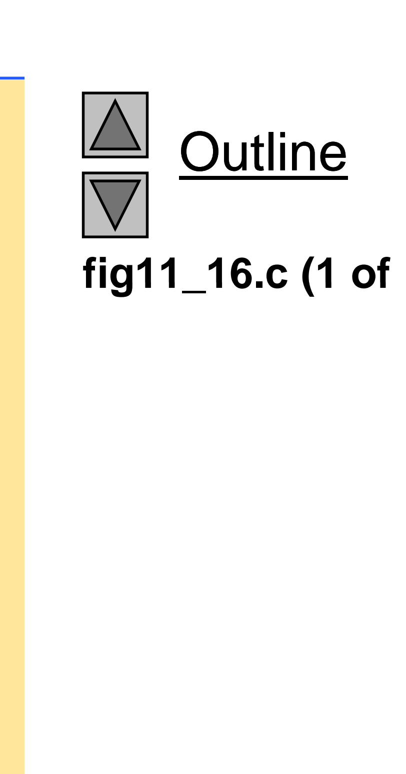


fig11_16.c (1 of 11)

```c
1  /* Fig. 11.16: fig11_16.c
2     This program reads a random access file sequentially, updates data
3     already written to the file, creates new data to be placed in the
4     file, and deletes data previously in the file. */
5  #include <stdio.h>
6
7  /* clientData structure definition */
8  struct clientData {
9     int acctNum;           /* account number */
10     char lastName[ 15 ];  /* account last name */
11     char firstName[ 10 ]; /* account first name */
12     double balance;       /* account balance */
13  }; /* end structure clientData */
14
15  /* prototypes */
16  int enterChoice( void );
17  void textFile( FILE *readPtr );
18  void updateRecord( FILE *fPtr );
19  void newRecord( FILE *fPtr );
20  void deleteRecord( FILE *fPtr );
21
22  int main()
23  {
24     FILE *cfPtr; /* credit.dat file pointer */
25     int choice;  /* user's choice */
26
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

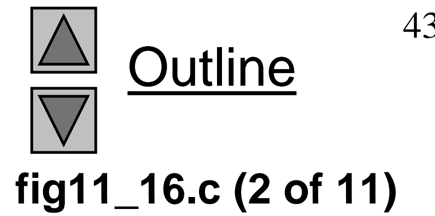


fig11_16.c (2 of 11)

```
27    /* fopen opens the file; exits if file cannot be opened */
28    if ( ( cfPtr = fopen( "credit.dat", "rb+" ) ) == NULL ) {
29       printf( "File could not be opened.\n" );
30    } /* end if */
31    else {
32
33       /* enable user to specify action */
34       while ( ( choice = enterChoice() ) != 5 ) {
35
36          switch ( choice ) {
37
38             /* create text file from record file */
39             case 1:
40                textFile( cfPtr );
41                break;
42
43             /* update record */
44             case 2:
45                updateRecord( cfPtr );
46                break;
47
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Outline

fig11_16.c (3 of 11)

```
48             /* create record */
49             case 3:
50                newRecord( cfPtr );
51                break;
52 
53             /* delete existing record */
54             case 4:
55                deleteRecord( cfPtr );
56                break;
57 
58             /* display message if user does not select valid choice */
59             default:
60                printf( "Incorrect choice\n" );
61                break;
62 
63          } /* end switch */
64 
65       } /* end while */
66 
67       fclose( cfPtr ); /* fclose closes the file */
68    } /* end else */
69 
70    return 0; /* indicates successful termination */
71 
72 } /* end main */
73 
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

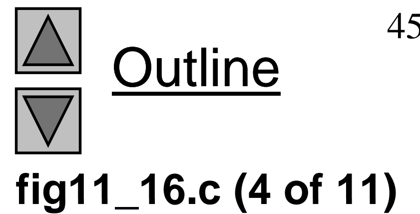


fig11_16.c (4 of 11)

```
74 /* create formatted text file for printing */
75 void textFile( FILE *readPtr )
76 {
77    FILE *writePtr; /* accounts.txt file pointer */
78
79    /* create clientData with no information */
80    struct clientData client = { 0, "", "", 0.0 };
81
82    /* fopen opens the file; exits if file cannot be opened */
83    if ( ( writePtr = fopen( "accounts.txt", "w" ) ) == NULL ) {
84       printf( "File could not be opened.\n" );
85    } /* end if */
86    else {
87       rewind( readPtr ); /* sets pointer to beginning of record file */
88       fprintf( writePtr, "%-6s%-16s%-11s%10s\n",
89                "Acct", "Last Name", "First Name","Balance" );
90
91       /* copy all records from record file into text file */
92       while ( !feof( readPtr ) ) {
93          fread( &client, sizeof( struct clientData ), 1, readPtr );
94
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
95          /* write single record to text file */
96          if ( client.acctNum != 0 ) {
97             fprintf( writePtr, "%-6d%-16s%-11s%10.2f\n",
98                      client.acctNum, client.lastName,
99                      client.firstName, client.balance );
100          } /* end if */
101
102       } /* end while */
103
104       fclose( writePtr ); /* fclose closes the file */
105    } /* end else */
106
107 } /* end function textFile */
108
109 /* update balance in record */
110 void updateRecord( FILE *fPtr )
111 {
112    int account;        /* account number */
113    double transaction; /* account transaction */
114
115    /* create clientData with no information */
116    struct clientData client = { 0, "", "", 0.0 };
117
```


© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

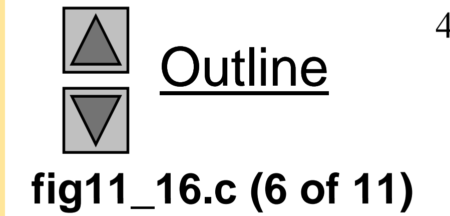


fig11_16.c (6 of 11)

```
118    /* obtain number of account to update */
119    printf( "Enter account to update ( 1 - 100 ): " );
120    scanf( "%d", &account );
121
122    /* move file pointer to correct record in file */
123    fseek( fPtr, ( account - 1 ) * sizeof( struct clientData ),
124           SEEK_SET );
125
126    /* read record from file */
127    fread( &client, sizeof( struct clientData ), 1, fPtr );
128
129    /* display error if account does not exist */
130    if ( client.acctNum == 0 ) {
131       printf( "Acount #%d has no information.\n", account );
132    } /* end if */
133    else { /* update record */
134       printf( "%-6d%-16s%-11s%10.2f\n\n",
135               client.acctNum, client.lastName,
136               client.firstName, client.balance );
137
138       /* request user to specify transaction */
139       printf( "Enter charge ( + ) or payment ( - ): " );
140       scanf( "%lf", &transaction );
141       client.balance += transaction; /* update record balance */
142
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

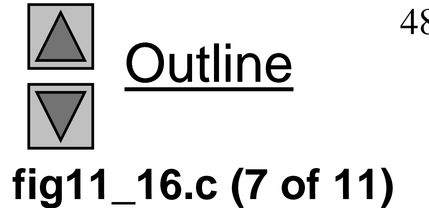


fig11_16.c (7 of 11)

```
143       printf( "%-6d%-16s%-11s%10.2f\n",
144               client.acctNum, client.lastName,
145               client.firstName, client.balance );
146
147       /* move file pointer to correct record in file */
148       fseek( fPtr, ( account - 1 ) * sizeof( struct clientData ),
149              SEEK_SET );
150
151       /* write updated record over old record in file */
152       fwrite( &client, sizeof( struct clientData ), 1, fPtr );
153    } /* end else */
154
155 } /* end function updateRecord */
156
157 /* delete an existing record */
158 void deleteRecord( FILE *fPtr )
159 {
160    /* create two clientDatas and initialize blankClient */
161    struct clientData client;
162    struct clientData blankClient = { 0, "", "", 0 };
163
164    int accountNum; /* account number */
165
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

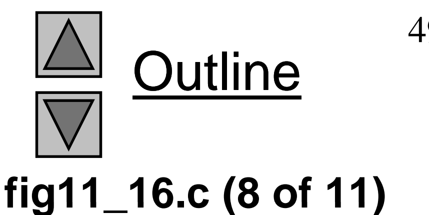


fig11_16.c (8 of 11)

```
166    /* obtain number of account to delete */
167    printf( "Enter account number to delete ( 1 - 100 ): " );
168    scanf( "%d", &accountNum );
169
170    /* move file pointer to correct record in file */
171    fseek( fPtr, ( accountNum - 1 ) * sizeof( struct clientData ),
172          SEEK_SET );
173
174    /* read record from file */
175    fread( &client, sizeof( struct clientData ), 1, fPtr );
176
177    /* display error if record does not exist */
178    if ( client.acctNum == 0 ) {
179       printf( "Account %d does not exist.\n", accountNum );
180    } /* end if */
181    else { /* delete record */
182
183       /* move file pointer to correct record in file */
184       fseek( fPtr, ( accountNum - 1 ) * sizeof( struct clientData ),
185          SEEK_SET );
186
187       /* replace existing record with blank record */
188       fwrite( &blankClient,
189               sizeof( struct clientData ), 1, fPtr );
190    } /* end else */
191
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
192 } /* end function deleteRecord */
193
194 /* create and insert record */
195 void newRecord( FILE *fPtr )
196 {
197    /* create clientData with no information */
198    struct clientData client = { 0, "", "", 0.0 };
199
200    int accountNum; /* account number */
201
202    /* obtain number of account to create */
203    printf( "Enter new account number ( 1 - 100 ): " );
204    scanf( "%d", &accountNum );
205
206    /* move file pointer to correct record in file */
207    fseek( fPtr, ( accountNum - 1 ) * sizeof( struct clientData ),
208           SEEK_SET );
209
210    /* read record from file */
211    fread( &client, sizeof( struct clientData ), 1, fPtr );
212
```


© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Outline

fig11_16.c (10 of 11)

```
213    /* display error if account previously exists */
214    if ( client.acctNum != 0 ) {
215       printf( "Account #%d already contains information.\n",
216               client.acctNum );
217    } /* end if */
218    else { /* create record */
219
220       /* user enters last name, first name and balance */
221       printf( "Enter lastname, firstname, balance\n? " );
222       scanf( "%s%s%lf", &client.lastName, &client.firstName,
223              &client.balance );
224
225       client.acctNum = accountNum;
226
227       /* move file pointer to correct record in file */
228       fseek( fPtr, ( client.acctNum - 1 ) *
229              sizeof( struct clientData ), SEEK_SET );
230
231       /* insert record in file */
232       fwrite( &client,
233               sizeof( struct clientData ), 1, fPtr );
234    } /* end else */
235
236 } /* end function newRecord */
237
```

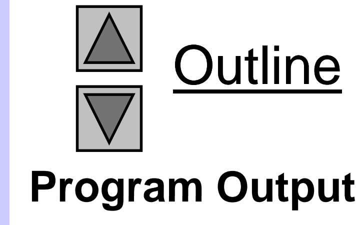
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Outline

fig11_16.c (11 of 11)

```
238 /* enable user to input menu choice */
239 int enterChoice( void )
240 {
241    int menuChoice; /* variable to store user's choice */
242
243    /* display available options */
244    printf( "\nEnter your choice\n"
245           "1 - store a formatted text file of acounts called\n"
246           "    \"accounts.txt\" for printing\n"
247           "2 - update an account\n"
248           "3 - add a new account\n"
249           "4 - delete an account\n"
250           "5 - end program\n? " );
251
252    scanf( "%d", &menuChoice ); /* receive choice from user */
253
254    return menuChoice;
255
256 } /* end function enterChoice */
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Program Output

```
After choosing option 1 accounts.txt contains:

Acct  Last Name        First Name     Balance
29    Brown            Nancy           -24.54
33    Dunn             Stacey          314.33
37    Barker           Doug              0.00
88    Smith            Dave            258.34
96    Stone            Sam              34.98
```

```
After choosing option 2 accounts.txt contains:

Enter account to update ( 1 - 100 ): 37
37    Barker           Doug              0.00

Enter charge ( + ) or payment ( - ): +87.99
37    Barker           Doug             87.99
```

```
After choosing option 3 accounts.txt contains:

Enter new account number ( 1 - 100 ): 22
Enter lastname, firstname, balance
? Johnston Sarah 247.45
```

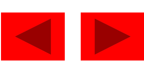
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Chapter 12 – Data Structures

Outline

12.1 Introduction
12.2 Self-Referential Structures
12.3 Dynamic Memory Allocation
12.4 Linked Lists
12.5 Stacks
12.6 Queues
12.7 Trees

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Objectives

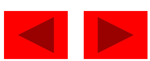
- In this chapter, you will learn:
  - To be able to allocate and free memory dynamically for data objects.
  - To be able to form linked data structures using pointers, self-referential structures and recursion.
  - To be able to create and manipulate linked lists, queues, stacks and binary trees.
  - To understand various important applications of linked data structures.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

12.1 Introduction

- Dynamic data structures
  - Data structures that grow and shrink during execution
- Linked lists
  - Allow insertions and removals anywhere
- Stacks
  - Allow insertions and removals only at top of stack
- Queues
  - Allow insertions at the back and removals from the front
- Binary trees
  - High-speed searching and sorting of data and efficient elimination of duplicate data items

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



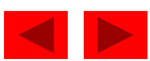
12.2 Self-Referential Structures

- Self-referential structures
  - Structure that contains a pointer to a structure of the same type
  - Can be linked together to form useful data structures such as lists, queues, stacks and trees
  - Terminated with a `NULL` pointer (**`0`**)

```
struct node {
   int data;
   struct node *nextPtr;
}
```

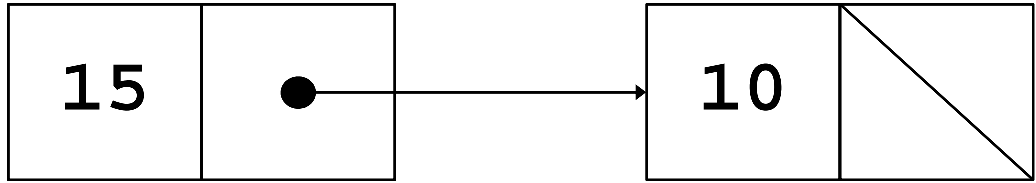
- `nextPtr`
  - Points to an object of type node
  - Referred to as a link
    - Ties one node to another **`node`**

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



12.3 Dynamic Memory Allocation

Figure 12.1 Two self-referential structures linked together



© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

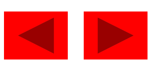
12.3 Dynamic Memory Allocation

- Dynamic memory allocation
  - Obtain and release memory during execution
- `malloc`
  - Takes number of bytes to allocate
    - Use `sizeof` to determine the size of an object
  - Returns pointer of type `void *`
    - A `void *` pointer may be assigned to any pointer
    - If no memory available, returns `NULL`
  - Example

```
newPtr = malloc( sizeof( struct node ) );
```

- `free`
  - Deallocates memory allocated by `malloc`
  - Takes a pointer as an argument
  - `free ( newPtr );`

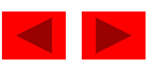
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



12.4 Linked Lists

- Linked list
  - Linear collection of self-referential class objects, called nodes
  - Connected by pointer links
  - Accessed via a pointer to the first node of the list
  - Subsequent nodes are accessed via the link-pointer member of the current node
  - Link pointer in the last node is set to `NULL` to mark the list's end
- Use a linked list instead of an array when
  - You have an unpredictable number of data elements
  - Your list needs to be sorted quickly

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



12.4 Linked Lists

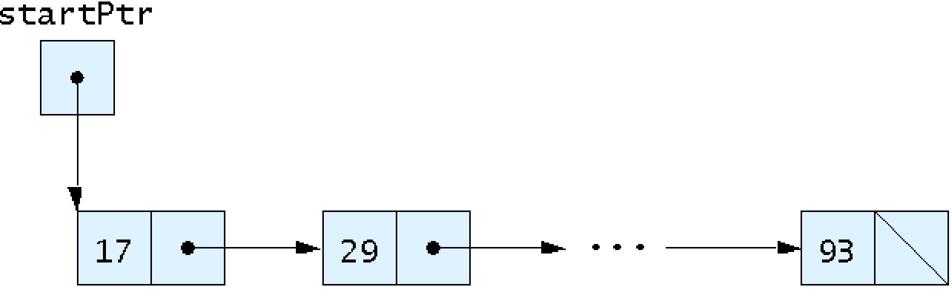
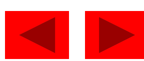
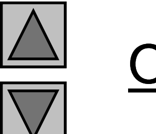


Fig. 12.2 A graphical representation of a linked list.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

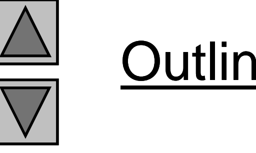


```c
1  /* Fig. 12.3: fig12_03.c
2     Operating and maintaining a list */
3  #include <stdio.h>
4  #include <stdlib.h>
5
6  /* self-referential structure */
7  struct listNode {
8     char data;               /* define data as char */
9     struct listNode *nextPtr; /* listNode pointer */
10 }; /* end structure listNode */
11
12 typedef struct listNode ListNode;
13 typedef ListNode *ListNodePtr;
14
15 /* prototypes */
16 void insert( ListNodePtr *sPtr, char value );
17 char delete( ListNodePtr *sPtr, char value );
18 int isEmpty( ListNodePtr sPtr );
19 void printList( ListNodePtr currentPtr );
20 void instructions( void );
21
22 int main()
23 {
```



© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
24    ListNodePtr startPtr = NULL; /* initialize startPtr */
25    int choice;                  /* user's choice */
26    char item;                   /* char entered by user */
27
28    instructions(); /* display the menu */
29    printf( "? " );
30    scanf( "%d", &choice );
31
32    /* loop while user does not choose 3 */
33    while ( choice != 3 ) {
34
35       switch ( choice ) {
36
37          case 1:
38             printf( "Enter a character: " );
39             scanf( "\n%c", &item );
40             insert( &startPtr, item );
41             printList( startPtr );
42             break;
43
44          case 2:
45
46             /* if list is not empty */
47             if ( !isEmpty( startPtr ) ) {
48                printf( "Enter character to be deleted: " );
49                scanf( "\n%c", &item );
50
```



Outline

fig12_03.c (Part 2 of 8)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

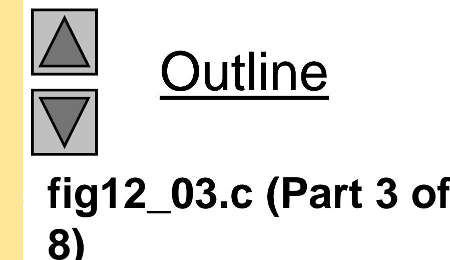
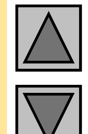


fig12_03.c (Part 3 of 8)

```
51                /* if character is found */
52                if ( delete( &startPtr, item ) ) {
53                   printf( "%c deleted.\n", item );
54                   printList( startPtr );
55                } /* end if */
56                else {
57                   printf( "%c not found.\n\n", item );
58                } /* end else */
59
60             } /* end if */
61             else {
62                printf( "List is empty.\n\n" );
63             } /* end else */
64
65             break;
66
67          default:
68             printf( "Invalid choice.\n\n" );
69             instructions();
70             break;
71
72       } /* end switch */
73
```

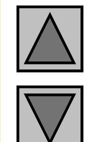
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
74       printf( "? " );
75       scanf( "%d", &choice );
76    } /* end while */
77
78    printf( "End of run.\n" );
79
80    return 0; /* indicates successful termination */
81
82 } /* end main */
83
84 /* display program instructions to user */
85 void instructions( void )
86 {
87    printf( "Enter your choice:\n"
88           "   1 to insert an element into the list.\n"
89           "   2 to delete an element from the list.\n"
90           "   3 to end.\n" );
91 } /* end function instructions */
92
93 /* Insert a new value into the list in sorted order */
94 void insert( ListNodePtr *sPtr, char value )
95 {
96    ListNodePtr newPtr;      /* pointer to new node */
97    ListNodePtr previousPtr; /* pointer to previous node in list */
98    ListNodePtr currentPtr;  /* pointer to current node in list */
99
```



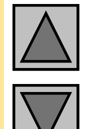
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
100    newPtr = malloc( sizeof( ListNode ) );
101
102    if ( newPtr != NULL ) {      /* is space available */
103       newPtr->data = value;
104       newPtr->nextPtr = NULL;
105
106       previousPtr = NULL;
107       currentPtr = *sPtr;
108
109       /* loop to find the correct location in the list */
110       while ( currentPtr != NULL && value > currentPtr->data ) {
111          previousPtr = currentPtr;          /* walk to ...   */
112          currentPtr = currentPtr->nextPtr;  /* ... next node */
113       } /* end while */
114
115       /* insert newPtr at beginning of list */
116       if ( previousPtr == NULL ) {
117          newPtr->nextPtr = *sPtr;
118          *sPtr = newPtr;
119       } /* end if */
120       else { /* insert newPtr between previousPtr and currentPtr */
121          previousPtr->nextPtr = newPtr;
122          newPtr->nextPtr = currentPtr;
123       } /* end else */
124
```



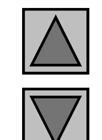
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
125    } /* end if */
126    else {
127       printf( "%c not inserted. No memory available.\n", value );
128    } /* end else */
129
130 } /* end function insert */
131
132 /* Delete a list element */
133 char delete( ListNodePtr *sPtr, char value )
134 {
135    ListNodePtr previousPtr; /* pointer to previous node in list */
136    ListNodePtr currentPtr;  /* pointer to current node in list */
137    ListNodePtr tempPtr;     /* temporary node pointer */
138
139    /* delete first node */
140    if ( value == ( *sPtr )->data ) {
141       tempPtr = *sPtr;
142       *sPtr = ( *sPtr )->nextPtr;  /* de-thread the node */
143       free( tempPtr );             /* free the de-threaded node */
144       return value;
145    } /* end if */
146    else {
147       previousPtr = *sPtr;
148       currentPtr = ( *sPtr )->nextPtr;
149
```



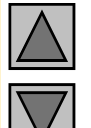
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
150       /* loop to find the correct location in the list */
151       while ( currentPtr != NULL && currentPtr->data != value ) {
152          previousPtr = currentPtr;          /* walk to ...    */
153          currentPtr = currentPtr->nextPtr;  /* ... next node */
154       } /* end while */
155
156       /* delete node at currentPtr */
157       if ( currentPtr != NULL ) {
158          tempPtr = currentPtr;
159          previousPtr->nextPtr = currentPtr->nextPtr;
160          free( tempPtr );
161          return value;
162       } /* end if */
163
164    } /* end else */
165
166    return '\0';
167
168 } /* end function delete */
169
170 /* Return 1 if the list is empty, 0 otherwise */
171 int isEmpty( ListNodePtr sPtr )
172 {
173    return sPtr == NULL;
174
175 } /* end function isEmpty */
176
```



© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
177 /* Print the list */
178 void printList( ListNodePtr currentPtr )
179 {
180
181    /* if list is empty */
182    if ( currentPtr == NULL ) {
183       printf( "List is empty.\n\n" );
184    } /* end if */
185    else {
186       printf( "The list is:\n" );
187
188       /* while not the end of the list */
189       while ( currentPtr != NULL ) {
190          printf( "%c --> ", currentPtr->data );
191          currentPtr = currentPtr->nextPtr;
192       } /* end while */
193
194       printf( "NULL\n\n" );
195    } /* end else */
196
197 } /* end function printList */
```

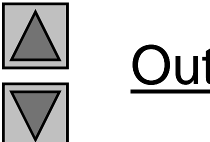


Outline

fig12_03.c (Part 8 of 8)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

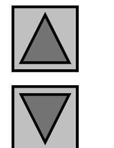
```
Enter your choice:
   1 to insert an element into the list.
   2 to delete an element from the list.
   3 to end.
? 1
Enter a character: B
The list is:
B --> NULL

? 1
Enter a character: A
The list is:
A --> B --> NULL

? 1
Enter a character: C
The list is:
A --> B --> C --> NULL

? 2
Enter character to be deleted: D
D not found.

? 2
Enter character to be deleted: B
B deleted.
The list is:
A --> C --> NULL
```



Program Output (Part 1 of 3)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

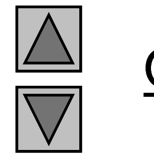
```
? 2
Enter character to be deleted: C
C deleted.
The list is:
A --> NULL

? 2
Enter character to be deleted: A
A deleted.
List is empty.

? 4
Invalid choice.

Enter your choice:
   1 to insert an element into the list.
   2 to delete an element from the list.
   3 to end.
? 3
End of run.

? 2
Enter character to be deleted: C
C deleted.
The list is:
A --> NULL

? 2
Enter character to be deleted: A
A deleted.
List is empty.
```



Program Output (Part 2 of 3)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
? 4
Invalid choice.

Enter your choice:
   1 to insert an element into the list.
   2 to delete an element from the list.
   3 to end.
? 3
End of run.
```



Program Output (Part 3 of 3)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

12.4 Linked Lists

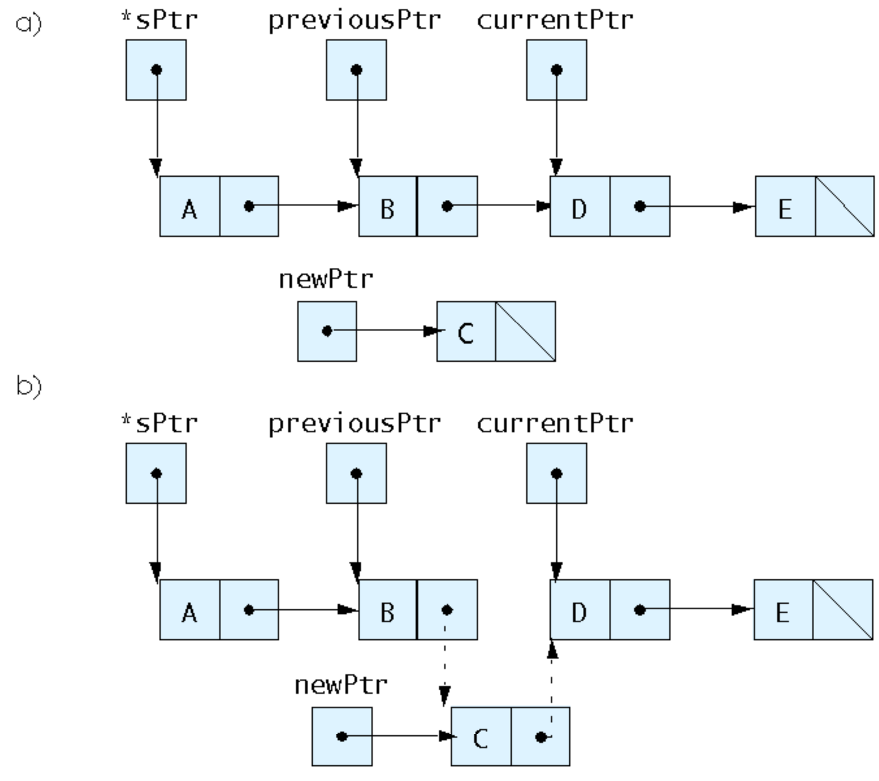


Fig. 12.5 Inserting a node in order in a list.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

12.5 Stacks

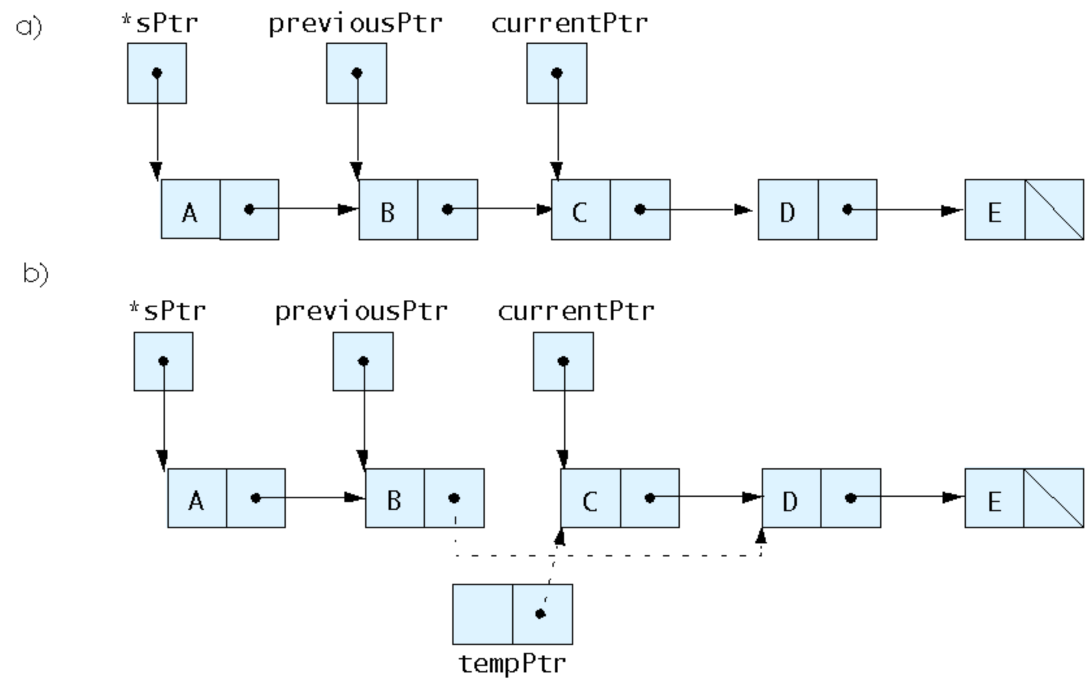
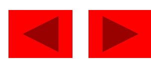


Fig. 12.6 Deleting a node from a list.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



12.5 Stacks

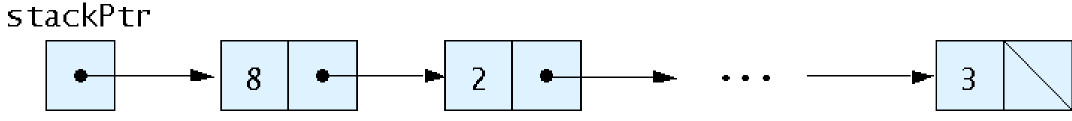
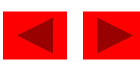
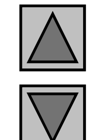


Fig. 12.7 Graphical representation of a stack.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.





Outline

fig12_08.c (Part 1 of 6)

```
1  /* Fig. 12.8: fig12_08.c
2     dynamic stack program */
3  #include <stdio.h>
4  #include <stdlib.h>
5
6  /* self-referential structure */
7  struct stackNode {
8     int data;                    /* define data as an int */
9     struct stackNode *nextPtr; /* stackNode pointer */
10 }; /* end structure stackNode */
11
12 typedef struct stackNode StackNode;
13 typedef StackNode *StackNodePtr;
14
15 /* prototypes */
16 void push( StackNodePtr *topPtr, int info );
17 int pop( StackNodePtr *topPtr );
18 int isEmpty( StackNodePtr topPtr );
19 void printStack( StackNodePtr currentPtr );
20 void instructions( void );
21
22 /* function main begins program execution */
23 int main()
24 {
25    StackNodePtr stackPtr = NULL;  /* points to stack top */
26    int choice;                    /* user's menu choice */
27    int value;                     /* int input by user */
28
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
29    instructions(); /* display the menu */
30    printf( "? " );
31    scanf( "%d", &choice );
32
33    /* while user does not enter 3 */
34    while ( choice != 3 ) {
35
36       switch ( choice ) {
37
38          /* push value onto stack */
39          case 1:
40             printf( "Enter an integer: " );
41             scanf( "%d", &value );
42             push( &stackPtr, value );
43             printStack( stackPtr );
44             break;
45
46          /* pop value off stack */
47          case 2:
48
49             /* if stack is not empty */
50             if ( !isEmpty( stackPtr ) ) {
51                printf( "The popped value is %d.\n", pop( &stackPtr ) );
52             } /* end if */
53
```

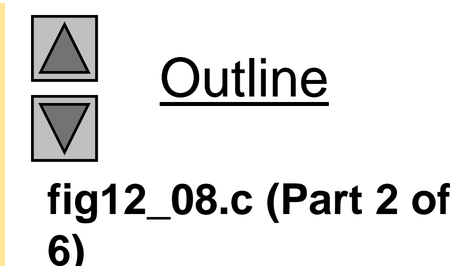


fig12_08.c (Part 2 of 6)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
54              printStack( stackPtr );
55              break;
56
57           default:
58              printf( "Invalid choice.\n\n" );
59              instructions();
60              break;
61
62        } /* end switch */
63
64        printf( "? " );
65        scanf( "%d", &choice );
66     } /* end while */
67
68     printf( "End of run.\n" );
69
70     return 0; /* indicates successful termination */
71
72  } /* end main */
73
74  /* display program instructions to user */
75  void instructions( void )
76  {
77     printf( "Enter choice:\n"
78            "1 to push a value on the stack\n"
79            "2 to pop a value off the stack\n"
80            "3 to end program\n" );
81  } /* end function instructions */
82
```

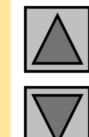
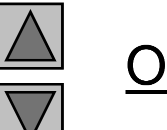


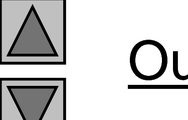
fig12_08.c (Part 3 of 6)

```
83 /* Insert a node at the stack top */
84 void push( StackNodePtr *topPtr, int info )
85 {
86    StackNodePtr newPtr; /* pointer to new node */
87
88    newPtr = malloc( sizeof( StackNode ) );
89
90    /* insert the node at stack top */
91    if ( newPtr != NULL ) {
92       newPtr->data = info;
93       newPtr->nextPtr = *topPtr;
94       *topPtr = newPtr;
95    } /* end if */
96    else { /* no space available */
97       printf( "%d not inserted. No memory available.\n", info );
98    } /* end else */
99
100 } /* end function push */
101
102 /* Remove a node from the stack top */
103 int pop( StackNodePtr *topPtr )
104 {
105    StackNodePtr tempPtr; /* temporary node pointer */
106    int popValue;         /* node value */
107
```



© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
108    tempPtr = *topPtr;
109    popValue = ( *topPtr )->data;
110    *topPtr = ( *topPtr )->nextPtr;
111    free( tempPtr );
112
113    return popValue;
114
115 } /* end function pop */
116
117 /* Print the stack */
118 void printStack( StackNodePtr currentPtr )
119 {
120
121    /* if stack is empty */
122    if ( currentPtr == NULL ) {
123       printf( "The stack is empty.\n\n" );
124    } /* end if */
125    else {
126       printf( "The stack is:\n" );
127
128       /* while not the end of the stack */
129       while ( currentPtr != NULL ) {
130          printf( "%d --> ", currentPtr->data );
131          currentPtr = currentPtr->nextPtr;
132       } /* end while */
133
```



© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

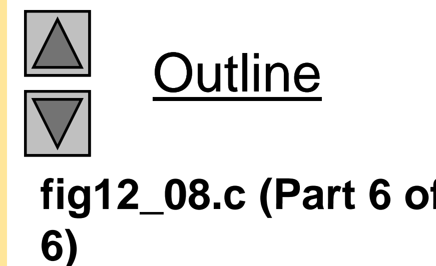


fig12_08.c (Part 6 of 6)

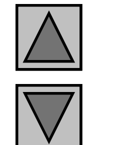
```
134       printf( "NULL\n\n" );
135    } /* end else */
136
137 } /* end function printList */
138
139 /* Return 1 if the stack is empty, 0 otherwise */
140 int isEmpty( StackNodePtr topPtr )
141 {
142    return topPtr == NULL;
143
144 } /* end function isEmpty */
```

Program Output (Part 1 of 2)

```
Enter choice:
1 to push a value on the stack
2 to pop a value off the stack
3 to end program
? 1
Enter an integer: 5
The stack is:
5 --> NULL

? 1
Enter an integer: 6
The stack is:
6 --> 5 --> NULL
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
? 1
Enter an integer: 4
The stack is:
4 --> 6 --> 5 --> NULL

? 2
The popped value is 4.
The stack is:
6 --> 5 --> NULL

? 2
The popped value is 6.
The stack is:
5 --> NULL

? 2
The popped value is 5.
The stack is empty.

? 2
The stack is empty.

? 4
Invalid choice.

Enter choice:
1 to push a value on the stack
2 to pop a value off the stack
3 to end program
? 3
End of run.
```



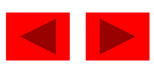
Program Output (Part 2 of 2)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

12.5 Stacks

- Stack
  - New nodes can be added and removed only at the top
  - Similar to a pile of dishes
  - Last-in, first-out (LIFO)
  - Bottom of stack indicated by a link member to `NULL`
  - Constrained version of a linked list
- `push`
  - Adds a new node to the top of the stack
- `pop`
  - Removes a node from the top
  - Stores the popped value
  - Returns `true` if `pop` was successful

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



12.5 Stacks

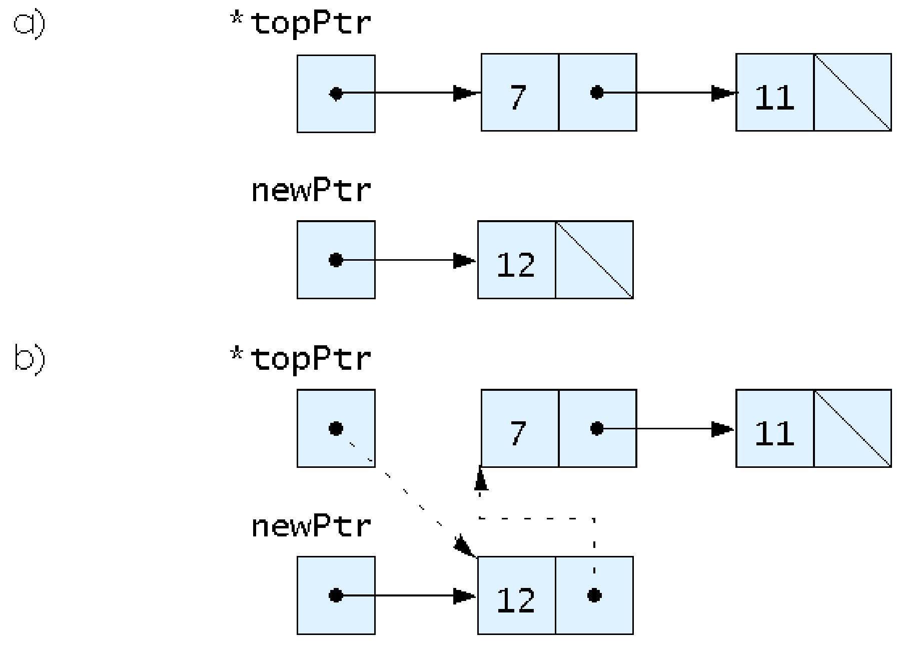
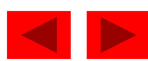


Fig. 12.10 push operation.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



12.5 Stacks

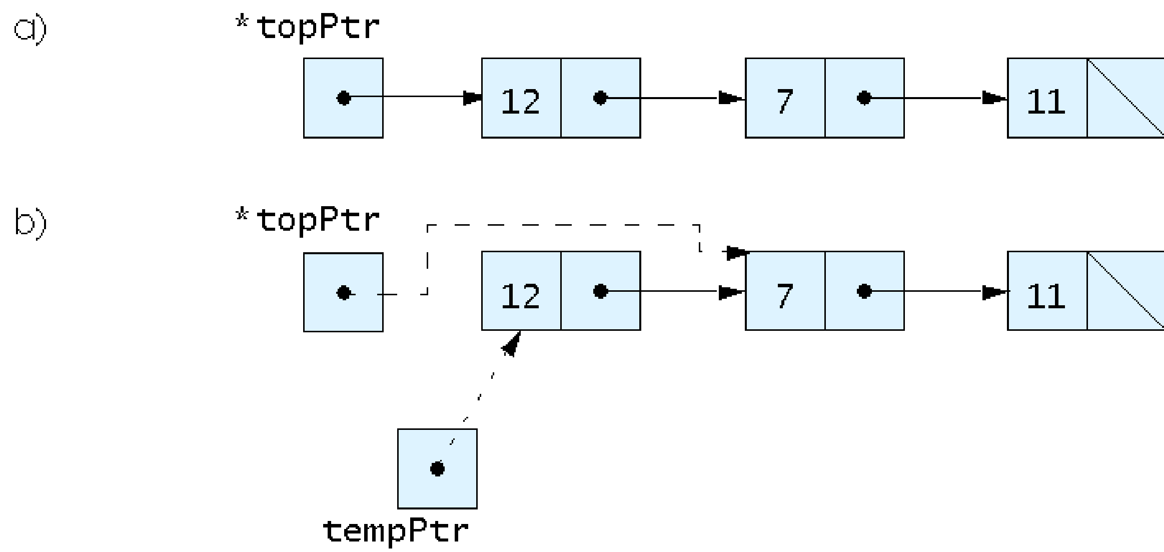
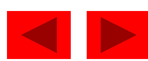


Fig. 12.11 pop operation.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



12.6 Queues

- Queue
  - Similar to a supermarket checkout line
  - First-in, first-out (FIFO)
  - Nodes are removed only from the head
  - Nodes are inserted only at the tail
- Insert and remove operations
  - Enqueue (insert) and dequeue (remove)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

12.6 Queues

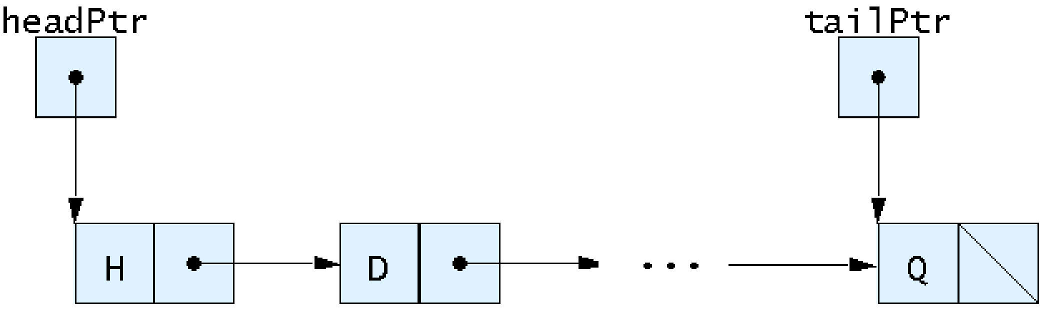
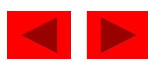
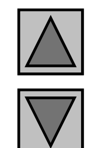


Fig. 12.12 A graphical representation of a queue.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



```
1  /* Fig. 12.13: fig12_13.c
2     Operating and maintaining a queue */
3
4  #include <stdio.h>
5  #include <stdlib.h>
6
7  /* self-referential structure */
8  struct queueNode {
9     char data;                   /* define data as a char */
10    struct queueNode *nextPtr; /* queueNode pointer */
11 }; /* end structure queueNode */
12
13 typedef struct queueNode QueueNode;
14 typedef QueueNode *QueueNodePtr;
15
16 /* function prototypes */
17 void printQueue( QueueNodePtr currentPtr );
18 int isEmpty( QueueNodePtr headPtr );
19 char dequeue( QueueNodePtr *headPtr, QueueNodePtr *tailPtr );
20 void enqueue( QueueNodePtr *headPtr, QueueNodePtr *tailPtr,
21               char value );
22 void instructions( void );
23
24 /* function main begins program execution */
25 int main()
26 {
```

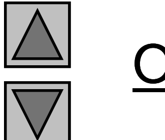


Outline

fig12_13.c (Part 1 of 7)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
27    QueueNodePtr headPtr = NULL; /* initialize headPtr */
28    QueueNodePtr tailPtr = NULL; /* initialize tailPtr */
29    int choice;                  /* user's menu choice */
30    char item;                   /* char input by user */
31
32    instructions(); /* display the menu */
33    printf( "? " );
34    scanf( "%d", &choice );
35
36    /* while user does not enter 3 */
37    while ( choice != 3 ) {
38
39       switch( choice ) {
40
41          /* enqueue value */
42          case 1:
43             printf( "Enter a character: " );
44             scanf( "\n%c", &item );
45             enqueue( &headPtr, &tailPtr, item );
46             printQueue( headPtr );
47             break;
48
49          /* dequeue value */
50          case 2:
51
```



Outline

fig12_13.c (Part 2 of 7)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
52            /* if queue is not empty */
53            if ( !isEmpty( headPtr ) ) {
54               item = dequeue( &headPtr, &tailPtr );
55               printf( "%c has been dequeued.\n", item );
56            } /* end if */
57
58            printQueue( headPtr );
59            break;
60
61         default:
62            printf( "Invalid choice.\n\n" );
63            instructions();
64            break;
65
66      } /* end switch */
67
68      printf( "? " );
69      scanf( "%d", &choice );
70   } /* end while */
71
72   printf( "End of run.\n" );
73
74   return 0; /* indicates successful termination */
75
76 } /* end main */
77
```

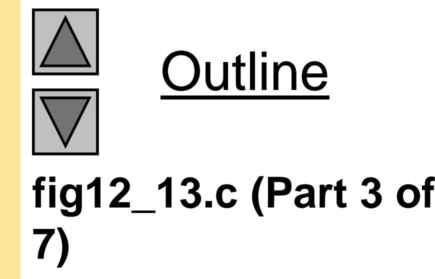


fig12_13.c (Part 3 of 7)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Outline

fig12_13.c (Part 4 of 7)

```
78 /* display program instructions to user */
79 void instructions( void )
80 {
81    printf ( "Enter your choice:\n"
82            "   1 to add an item to the queue\n"
83            "   2 to remove an item from the queue\n"
84            "   3 to end\n" );
85 } /* end function instructions */
86
87 /* insert a node a queue tail */
88 void enqueue( QueueNodePtr *headPtr, QueueNodePtr *tailPtr,
89               char value )
90 {
91    QueueNodePtr newPtr; /* pointer to new node */
92
93    newPtr = malloc( sizeof( QueueNode ) );
94
95    if ( newPtr != NULL ) { /* is space available */
96       newPtr->data = value;
97       newPtr->nextPtr = NULL;
98
99       /* if empty, insert node at head */
100      if ( isEmpty( *headPtr ) ) {
101         *headPtr = newPtr;
102      } /* end if */
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
103       else {
104          ( *tailPtr )->nextPtr = newPtr;
105       } /* end else */
106
107       *tailPtr = newPtr;
108    } /* end if */
109    else {
110       printf( "%c not inserted. No memory available.\n", value );
111    } /* end else */
112
113 } /* end function enqueue */
114
115 /* remove node from queue head */
116 char dequeue( QueueNodePtr *headPtr, QueueNodePtr *tailPtr )
117 {
118    char value;           /* node value */
119    QueueNodePtr tempPtr; /* temporary node pointer */
120
121    value = ( *headPtr )->data;
122    tempPtr = *headPtr;
123    *headPtr = ( *headPtr )->nextPtr;
124
125    /* if queue is empty */
126    if ( *headPtr == NULL ) {
127       *tailPtr = NULL;
128    } /* end if */
129
```

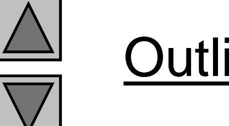


fig12_13.c (Part 5 of 7)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
130    free( tempPtr );
131
132    return value;
133
134 } /* end function dequeue */
135
136 /* Return 1 if the list is empty, 0 otherwise */
137 int isEmpty( QueueNodePtr headPtr )
138 {
139    return headPtr == NULL;
140
141 } /* end function isEmpty */
142
143 /* Print the queue */
144 void printQueue( QueueNodePtr currentPtr )
145 {
146
147    /* if queue is empty */
148    if ( currentPtr == NULL ) {
149       printf( "Queue is empty.\n\n" );
150    } /* end if */
151    else {
152       printf( "The queue is:\n" );
153
```

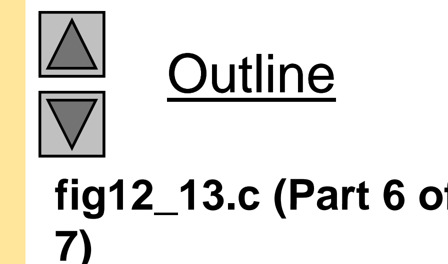


fig12_13.c (Part 6 of 7)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

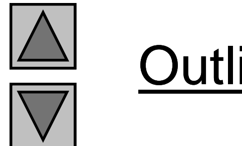
```
154       /* while not end of queue */
155       while ( currentPtr != NULL ) {
156          printf( "%c --> ", currentPtr->data );
157          currentPtr = currentPtr->nextPtr;
158       } /* end while */
159
160       printf( "NULL\n\n" );
161    } /* end else */
162
163 } /* end function printQueue */
```

```
Enter your choice:
   1 to add an item to the queue
   2 to remove an item from the queue
   3 to end
? 1
Enter a character: A
The queue is:
A --> NULL

? 1
Enter a character: B
The queue is:
A --> B --> NULL

? 1
Enter a character: C
The queue is:
A --> B --> C --> NULL
```


Program Output (Part 1 of 2)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
? 2
A has been dequeued.
The queue is:
B --> C --> NULL

? 2
B has been dequeued.
The queue is:
C --> NULL

? 2
C has been dequeued.
Queue is empty.

? 2
Queue is empty.

? 4
Invalid choice.

Enter your choice:
   1 to add an item to the queue
   2 to remove an item from the queue
   3 to end
? 3
End of run.
```



Program Output (Part 2 of 2)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

12.6 Queues

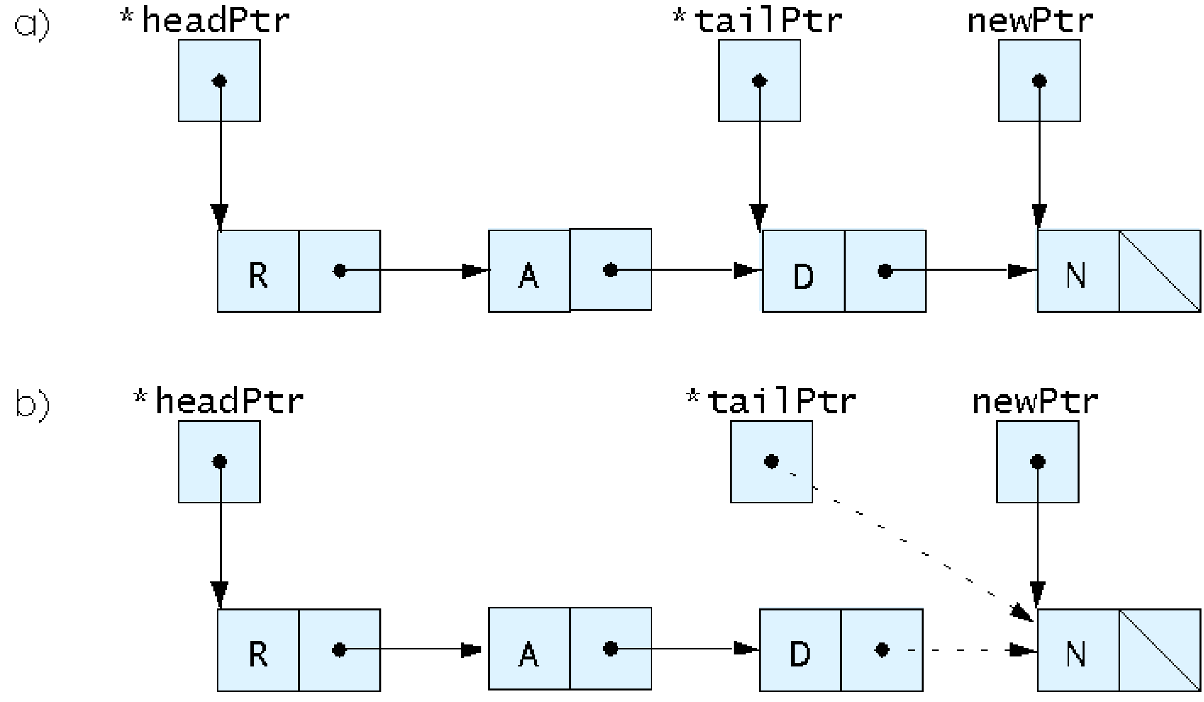
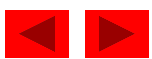


Fig. 12.15 A graphical representation of the enqueue operation

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



12.6 Queues

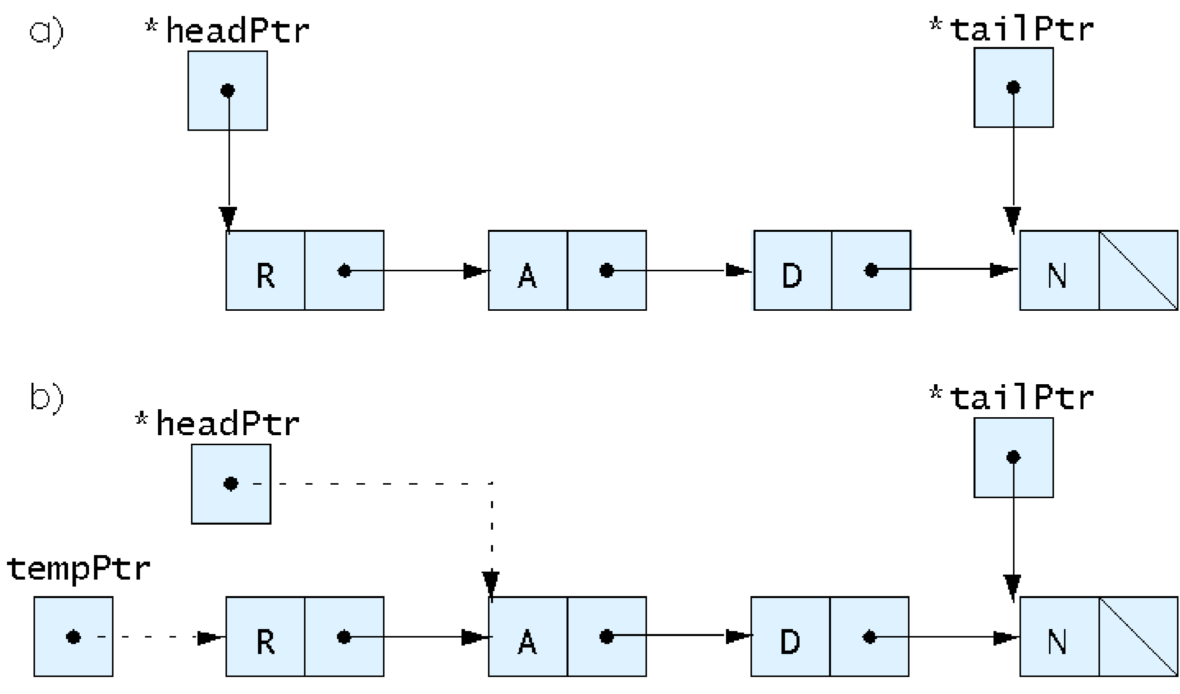
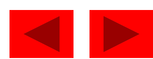


Fig. 12.16 A graphical representation of the dequeue operation.

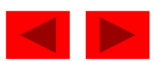
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



12.7 Trees

- Tree nodes contain two or more links
  - All other data structures we have discussed only contain one
- Binary trees
  - All nodes contain two links
    - None, one, or both of which may be NULL
  - The root node is the first node in a tree.
  - Each link in the root node refers to a child
  - A node with no children is called a leaf node

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



12.7 Trees

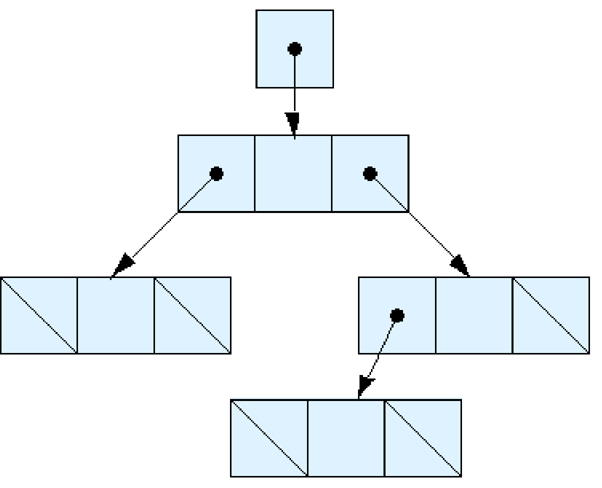
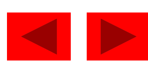


Fig. 12.17 A graphical representation of a binary tree.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



12.7 Trees

- Binary search tree
  - Values in left subtree less than parent
  - Values in right subtree greater than parent
  - Facilitates duplicate elimination
  - Fast searches - for a balanced tree, maximum of $\log_2 n$ comparisons

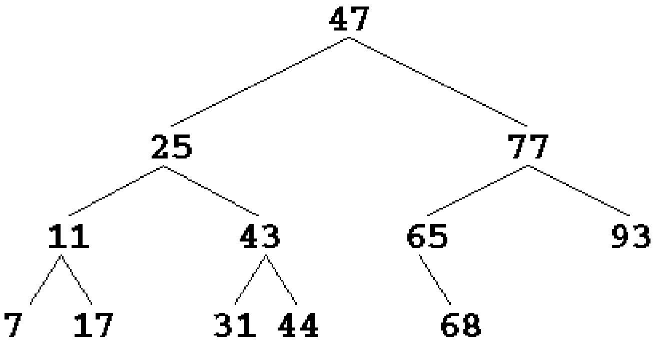
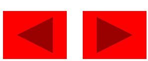


Fig. 12.18 A binary search tree.

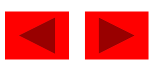
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



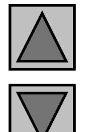
12.7 Trees

- Tree traversals:
  - Inorder traversal – prints the node values in ascending order
    1. Traverse the left subtree with an inorder traversal
    2. Process the value in the node (i.e., print the node value)
    3. Traverse the right subtree with an inorder traversal
  - Preorder traversal
    1. Process the value in the node
    2. Traverse the left subtree with a preorder traversal
    3. Traverse the right subtree with a preorder traversal
  - Postorder traversal
    1. Traverse the left subtree with a postorder traversal
    2. Traverse the right subtree with a postorder traversal
    3. Process the value in the node

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

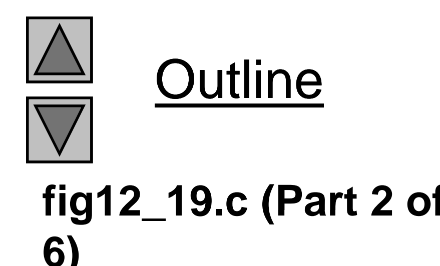


```
1  /* Fig. 12.19: fig12_19.c
2     Create a binary tree and traverse it
3     preorder, inorder, and postorder */
4  #include <stdio.h>
5  #include <stdlib.h>
6  #include <time.h>
7
8  /* self-referential structure */
9  struct treeNode {
10    struct treeNode *leftPtr;  /* treeNode pointer */
11    int data;                  /* define data as an int */
12    struct treeNode *rightPtr; /* treeNode pointer */
13 }; /* end structure treeNode */
14
15 typedef struct treeNode TreeNode;
16 typedef TreeNode *TreeNodePtr;
17
18 /* prototypes */
19 void insertNode( TreeNodePtr *treePtr, int value );
20 void inOrder( TreeNodePtr treePtr );
21 void preOrder( TreeNodePtr treePtr );
22 void postOrder( TreeNodePtr treePtr );
23
24 /* function main begins program execution */
25 int main()
26 {
```



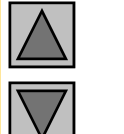
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
27    int i;                        /* counter */
28    int item;                     /* variable to hold random values */
29    TreeNodePtr rootPtr = NULL; /* initialize rootPtr */
30
31    srand( time( NULL ) );
32    printf( "The numbers being placed in the tree are:\n" );
33
34    /* insert random values between 1 and 15 in the tree */
35    for ( i = 1; i <= 10; i++ ) {
36       item = rand() % 15;
37       printf( "%3d", item );
38       insertNode( &rootPtr, item );
39    } /* end for */
40
41    /* traverse the tree preOrder */
42    printf( "\n\nThe preOrder traversal is:\n" );
43    preOrder( rootPtr );
44
45    /* traverse the tree inOrder */
46    printf( "\n\nThe inOrder traversal is:\n" );
47    inOrder( rootPtr );
48
49    /* traverse the tree postOrder */
50    printf( "\n\nThe postOrder traversal is:\n" );
51    postOrder( rootPtr );
52
```



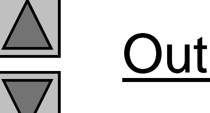
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
53    return 0; /* indicates successful termination */
54
55 } /* end main */
56
57 /* insert node into tree */
58 void insertNode( TreeNodePtr *treePtr, int value )
59 {
60
61    /* if tree is empty */
62    if ( *treePtr == NULL ) {
63       *treePtr = malloc( sizeof( TreeNode ) );
64
65       /* if memory was allocated then assign data */
66       if ( *treePtr != NULL ) {
67          ( *treePtr )->data = value;
68          ( *treePtr )->leftPtr = NULL;
69          ( *treePtr )->rightPtr = NULL;
70       } /* end if */
71       else {
72          printf( "%d not inserted. No memory available.\n", value );
73       } /* end else */
74
75    } /* end if */
```



© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
76     else { /* tree is not empty */
77
78        /* data to insert is less than data in current node */
79        if ( value < ( *treePtr )->data ) {
80           insertNode( &( ( *treePtr )->leftPtr ), value );
81        } /* end if */
82
83        /* data to insert is greater than data in current node */
84        else if ( value > ( *treePtr )->data ) {
85           insertNode( &( ( *treePtr )->rightPtr ), value );
86        } /* end else if */
87        else { /* duplicate data value ignored */
88           printf( "dup" );
89        } /* end else */
90
91     } /* end else */
92
93  } /* end function insertNode */
94
95  /* begin inorder traversal of tree */
96  void inOrder( TreeNodePtr treePtr )
97  {
98
```

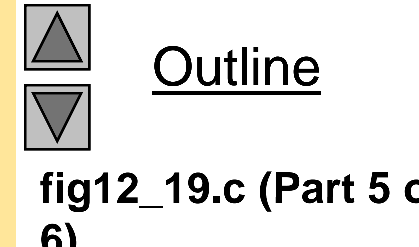


Outline

fig12_19.c (Part 4 of 6)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
99    /* if tree is not empty then traverse */
100   if ( treePtr != NULL ) {
101      inOrder( treePtr->leftPtr );
102      printf( "%3d", treePtr->data );
103      inOrder( treePtr->rightPtr );
104   } /* end if */
105
106 } /* end function inOrder */
107
108 /* begin preorder traversal of tree */
109 void preOrder( TreeNodePtr treePtr )
110 {
111
112   /* if tree is not empty then traverse */
113   if ( treePtr != NULL ) {
114      printf( "%3d", treePtr->data );
115      preOrder( treePtr->leftPtr );
116      preOrder( treePtr->rightPtr );
117   } /* end if */
118
119 } /* end function preOrder */
120
```



© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
121 /* begin postorder traversal of tree */
122 void postOrder( TreeNodePtr treePtr )
123 {
124
125    /* if tree is not empty then traverse */
126    if ( treePtr != NULL ) {
127       postOrder( treePtr->leftPtr );
128       postOrder( treePtr->rightPtr );
129       printf( "%3d", treePtr->data );
130    } /* end if */
131
132 } /* end function postOrder */
```

Outline

fig12_19.c (Part 6 of 6)

Program Output

```
The numbers being placed in the tree are:
  6  7  4 12  7dup  2  2dup  5  7dup 11

The preOrder traversal is:
  6  4  2  5  7 12 11

The inOrder traversal is:
  2  4  5  6  7 11 12

The postOrder traversal is:
  2  5  4 11 12  7  6
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

12.7 Trees

```
6 13 17 27 33 42 48
```

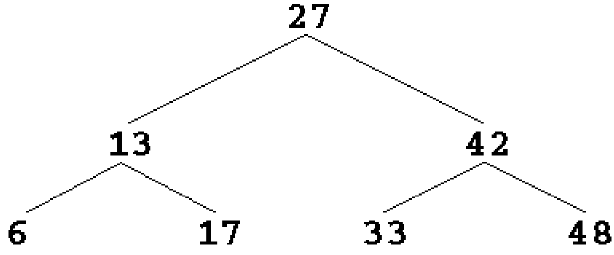
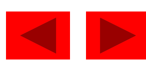


Fig. 12.21 A binary search tree.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

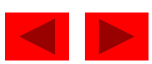


Chapter 13 - The Preprocessor

Outline

13.1 Introduction
13.2 The `#include` Preprocessor Directive
13.3 The `#define` Preprocessor Directive: Symbolic Constants
13.4 The `#define` Preprocessor Directive: Macros
13.5 Conditional Compilation
13.6 The `#error` and `#pragma` Preprocessor Directives
13.7 The # and ## Operators
13.8 Line Numbers
13.9 Predefined Symbolic Constants
13.10 Assertions

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Objectives

- In this chapter, you will learn:
  - To be able to use #include for developing large programs.
  - To be able to use #define to create macros and macros with arguments.
  - To understand conditional compilation.
  - To be able to display error messages during conditional compilation.
  - To be able to use assertions to test if the values of expressions are correct.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

13.1 Introduction

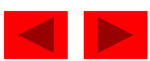
- Preprocessing
  - Occurs before a program is compiled
  - Inclusion of other files
  - Definition of symbolic constants and macros
  - Conditional compilation of program code
  - Conditional execution of preprocessor directives
- Format of preprocessor directives
  - Lines begin with #
  - Only whitespace characters before directives on a line

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

13.2 The #include Preprocessor Directive

- #include
  - Copy of a specified file included in place of the directive
  - #include <filename>
    - Searches standard library for file
    - Use for standard library files
  - #include "filename"
    - Searches current directory, then standard library
    - Use for user-defined files
  - Used for:
    - Programs with multiple source files to be compiled together
    - Header file – has common declarations and definitions (classes, structures, function prototypes)
      - #include statement in each file

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



13.3 The #define Preprocessor Directive: Symbolic Constants

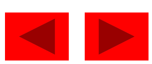
- #define
  - Preprocessor directive used to create symbolic constants and macros
  - Symbolic constants
    - When program compiled, all occurrences of symbolic constant replaced with replacement text
  - Format

    #define *identifier replacement-text*
  - Example:

```
#define PI 3.14159
```
  - Everything to right of identifier replaces text

```
#define PI = 3.14159
```
    - Replaces “PI” with "= 3.14159"
  - Cannot redefine symbolic constants once they have been created

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



13.4 The #define Preprocessor Directive: Macros

- Macro
  - Operation defined in #define
  - A macro without arguments is treated like a symbolic constant
  - A macro with arguments has its arguments substituted for replacement text, when the macro is expanded
  - Performs a text substitution – no data type checking
  - The macro

```
#define CIRCLE_AREA( x ) ( PI * ( x ) * ( x ) )
```

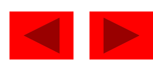
would cause

```
area = CIRCLE_AREA( 4 );
```

to become

```
area = ( 3.14159 * ( 4 ) * ( 4 ) );
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



13.4 The `#define` Preprocessor Directive: Macros

- Use parenthesis
  - Without them the macro

```
#define CIRCLE_AREA( x )  PI * ( x ) * ( x )
```

would cause

```
area = CIRCLE_AREA( c + 2 );
```

to become

```
area = 3.14159 * c + 2 * c + 2;
```

- Multiple arguments

```
#define RECTANGLE_AREA( x, y )  ( ( x ) * ( y ) )
```

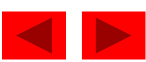
would cause

```
rectArea = RECTANGLE_AREA( a + 4, b + 7 );
```

to become

```
rectArea = ( ( a + 4 ) * ( b + 7 ) );
```

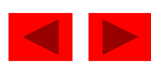
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



13.4 The #define Preprocessor Directive: Macros

- #undef
  - Undefines a symbolic constant or macro
  - If a symbolic constant or macro has been undefined it can later be redefined

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



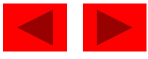
13.5 Conditional Compilation

- Conditional compilation
  - Control preprocessor directives and compilation
  - Cast expressions, `sizeof`, enumeration constants cannot be evaluated in preprocessor directives
  - Structure similar to `if`

```
#if !defined( NULL )
   #define NULL 0
#endif
```

    - Determines if symbolic constant `NULL` has been defined
      - If `NULL` is defined, `defined( NULL )` evaluates to `1`
      - If `NULL` is not defined, this function defines `NULL` to be `0`
  - Every `#if` must end with `#endif`
  - `#ifdef` short for `#if defined( name )`
  - `#ifndef` short for `#if !defined( name )`

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



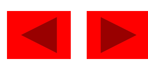
13.5 Conditional Compilation

- Other statements
  - `#elif` – equivalent of `else if` in an `if` statement
  - `#else` – equivalent of `else` in an `if` statement
- "Comment out" code
  - Cannot use `/*` ... `*/`
  - Use

```
#if 0
    code commented out
#endif
```

  - To enable code, change `0` to `1`

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



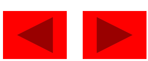
13.5 Conditional Compilation

- Debugging

```
#define DEBUG  1
#ifdef DEBUG
     cerr << "Variable x = " << x << endl;
 #endif
```

  – Defining `DEBUG` to `1` enables code
  – After code corrected, remove `#define` statement
  – Debugging statements are now ignored

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



13.6 The `#error` and `#pragma` Preprocessor Directives

- `#error` tokens
  - Tokens are sequences of characters separated by spaces
    - `"I like C++"` has 3 tokens
  - Displays a message including the specified tokens as an error message
  - Stops preprocessing and prevents program compilation
- `#pragma` tokens
  - Implementation defined action (consult compiler documentation)
  - Pragmas not recognized by compiler are ignored

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

13.7 The # and ## Operators

- #
  - Causes a replacement text token to be converted to a string surrounded by quotes
  - The statement

```
#define HELLO( x ) printf( "Hello, " #x  "\n" );
```

would cause

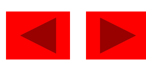
```
HELLO( John )
```

to become

```
printf( "Hello, " "John" "\n" );
```

  - Strings separated by whitespace are concatenated when using `printf`

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



13.7 The # and ## Operators

- ##
  - Concatenates two tokens
  - The statement

```
#define TOKENCONCAT( x, y )  x ## y
```

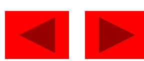
would cause

```
TOKENCONCAT( O, K )
```

to become

```
OK
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



13.8 Line Numbers

- `#line`
  - Renumbers subsequent code lines, starting with integer value
  - File name can be included
  - `#line 100 "myFile.c"`
    - Lines are numbered from 100 beginning with next source code file
    - Compiler messages will think that the error occurred in `"myfile.C"`
    - Makes errors more meaningful
    - Line numbers do not appear in source file

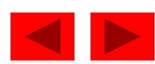
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

13.9 Predefined Symbolic Constants

- Four predefined symbolic constants
  - Cannot be used in `#define` or `#undef`

Symbolic constant	Description
`__LINE__`	The line number of the current source code line (an integer constant).
`__FILE__`	The presumed name of the source file (a string).
`__DATE__`	The date the source file is compiled (a string of the form **"Mmm dd yyyy"** such as **"Jan 19 2001"**).
`__TIME__`	The time the source file is compiled (a string literal of the form **"hh:mm:ss"**).

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



13.10 Assertions

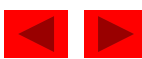
- `assert` macro
  - Header <`assert.h`>
  - Tests value of an expression
  - If `0` (`false`) prints error message and calls `abort`
  - Example:

```
assert( x <= 10 );
```

  - If `NDEBUG` is defined
    - All subsequent assert statements ignored

```
#define NDEBUG
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

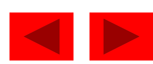


Chapter 14 - Advanced C Topics

Outline

14.1 Introduction
14.2 Redirecting Input/Output on UNIX and DOS Systems
14.3 Variable-Length Argument Lists
14.4 Using Command-Line Arguments
14.5 Notes on Compiling Multiple-Source-File Programs
14.6 Program Termination with `exit` and `atexit`
14.7 The `volatile` Type Qualifier
14.8 Suffixes for Integer and Floating-Point Constants
14.9 More on Files
14.10 Signal Handling
14.11 Dynamic Memory Allocation with `calloc` and `realloc`
14.12 The Unconditional Branch: `goto`

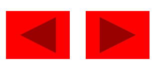
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Objectives

- In this chapter, you will learn:
  - To be able to redirect keyboard input to come from a file.
  - To be able to redirect screen output to be placed in a file.
  - To be able to write functions that use variable-length argument lists.
  - To be able to process command-line arguments.
  - To be able to assign specific types to numeric constants
  - To be able to use temporary files.
  - To be able to process unexpected events within a program.
  - To be able to allocate memory dynamically for arrays.
  - To be able to change the size of memory that was dynamically allocated previously.

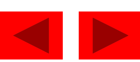
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



14.1 Introduction

- Several advanced topics in this chapter
- Operating system specific
  - Usually UNIX or DOS

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



14.2 Redirecting Input/Output on UNIX and DOS Systems

- Standard I/O - keyboard and screen
  - Redirect input and output
- Redirect symbol(<)
  - Operating system feature, not a C feature
  - UNIX and DOS
  - $ or % represents command line
  - Example:

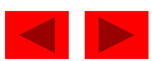
```
$ sum < input
```

  - Rather than inputting values by hand, read them from a file
- Pipe command(|)
  - Output of one program becomes input of another

```
$ random | sum
```

  - Output of `random` goes to `sum`

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



14.2 Redirecting Input/Output on UNIX and DOS Systems

- Redirect output (>)
  - Determines where output of a program goes
  - Example:

    ```
    $ random > out
    ```

    - Output goes into `out` (erases previous contents)
- Append output (>>)
  - Add output to end of file (preserve previous contents)
  - Example:

    ```
    $ random >> out
    ```

    - Output is added onto the end of `out`

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

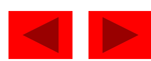
14.3 Variable-Length Argument Lists

- Functions with unspecified number of arguments
  - Load `<stdarg.h>`
  - Use ellipsis(. . .) at end of parameter list
  - Need at least one defined parameter
  - Example:

    ```
    double myfunction ( int i, ... );
    ```
  - The ellipsis is only used in the prototype of a function with a variable length argument list
  - `printf` is an example of a function that can take multiple arguments
  - The prototype of `printf` is defined as

    ```
    int printf( const char* format, ... );
    ```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

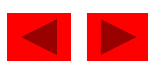


14.3 Variable-Length Argument Lists

Identifier	Explanation
`va_list`	A type suitable for holding information needed by macros `va_start`, `va_arg` and `va_end`. To access the arguments in a variable-length argument list, an object of type `va_list` must be declared.
`va_start`	A macro that is invoked before the arguments of a variable-length argument list can be accessed. The macro initializes the object declared with `va_list` for use by the `va_arg` and `va_end` macros.
`va_arg`	A macro that expands to an expression of the value and type of the next argument in the variable-length argument list. Each invocation of `va_arg` modifies the object declared with `va_list` so that the object points to the next argument in the list.
`va_end`	A macro that facilitates a normal return from a function whose variable-length argument list was referred to by the `va_start` macro.

Fig. 14.1 The type and the macros defined in header `stdarg.h`.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



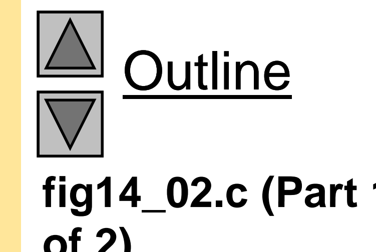
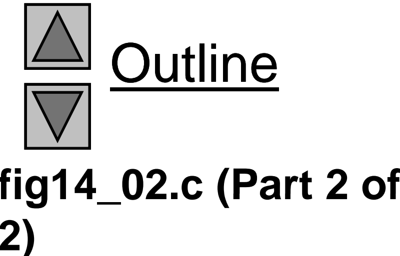


fig14_02.c (Part 1 of 2)

```
1  /* Fig. 14.2: fig14_02.c
2     Using variable-length argument lists */
3  #include <stdio.h>
4  #include <stdarg.h>
5
6  double average( int i, ... ); /* prototype */
7
8  int main()
9  {
10    double w = 37.5; /* initialize w */
11    double x = 22.5; /* initialize x */
12    double y = 1.7;  /* initialize y */
13    double z = 10.2; /* initialize z */
14
15    printf( "%s%.1f\n%s%.1f\n%s%.1f\n%s%.1f\n\n",
16            "w = ", w, "x = ", x, "y = ", y, "z = ", z );
17    printf( "%s%.3f\n%s%.3f\n%s%.3f\n",
18            "The average of w and x is ", average( 2, w, x ),
19            "The average of w, x, and y is ", average( 3, w, x, y ),
20            "The average of w, x, y, and z is ",
21            average( 4, w, x, y, z ) );
22
23    return 0; /* indicates successful termination */
24
25 } /* end main */
26
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.




```
27 /* calculate average */
28 double average( int i, ... )
29 {
30    double total = 0; /* initialize total */
31    int j;            /* counter */
32    va_list ap;       /* for storing information needed by va_start */
33
34    va_start( ap, i ); /* initialize ap for use in va_arg and va_end */
35
36    /* process variable length argument list */
37    for ( j = 1; j <= i; j++ ) {
38       total += va_arg( ap, double );
39    } /* end for */
40
41    va_end( ap ); /* end the va_start */
42
43    return total / i; /* calculate average */
44
45 } /* end function average */
```

Program Output

```
w = 37.5
x = 22.5
y = 1.7
z = 10.2

The average of w and x is 30.000
The average of w, x, and y is 20.567
The average of w, x, y, and z is 17.975
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

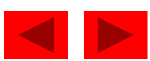
14.4 Using Command-Line Arguments

- Pass arguments to `main` on DOS or UNIX
  - Define main as

    ```
    int main( int argc, char *argv[] )
    ```

  - `int argc`
    - Number of arguments passed
  - `char *argv[]`
    - Array of strings
    - Has names of arguments in order
      - `argv[ 0 ]` is first argument
  - Example: `$ mycopy input output`
    - `argc: 3`
    - `argv[ 0 ]: "mycopy"`
    - `argv[ 1 ]: "input"`
    - `argv[ 2 ]: "output"`

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



```
1  /* Fig. 14.3: fig14_03.c
2     Using command-line arguments */
3  #include <stdio.h>
4
5  int main( int argc, char *argv[] )
6  {
7     FILE *inFilePtr;  /* input file pointer */
8     FILE *outFilePtr; /* output file pointer */
9     int c;            /* define c to hold characters input by user */
10
11    /* check number of command-line arguments */
12    if ( argc != 3 ) {
13       printf( "Usage: copy infile outfile\n" );
14    } /* end if */
15    else {
16
17       /* if input file can be opened */
18       if ( ( inFilePtr = fopen( argv[ 1 ], "r" ) ) != NULL ) {
19
20          /* if output file can be opened */
21          if ( ( outFilePtr = fopen( argv[ 2 ], "w" ) ) != NULL ) {
22
23             /* read and output characters */
24             while ( ( c = fgetc( inFilePtr ) ) != EOF ) {
25                fputc( c, outFilePtr );
26             } /* end while */
27
28          } /* end if */
```

Notice `argc` and `argv[]` in `main`

`argv[1]` is the second argument, and is being read.

`argv[2]` is the third argument, and is being written to.

Loop until `End Of File`. `fgetc` a character from `inFilePtr` and `fputc` it into `outFilePtr`.

Outline

fig14_03.c (Part 1 of 2)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
29          else { /* output file could not be opened */
30             printf( "File \"%s\" could not be opened\n", argv[ 2 ] );
31          } /* end else */
32
33       } /* end if */
34       else { /* input file could not be opened */
35          printf( "File \"%s\" could not be opened\n", argv[ 1 ] );
36       } /* end else */
37
38    } /* end else */
39
40    return 0; /* indicates successful termination */
41
42 } /* end main */
```

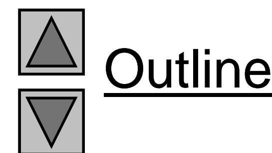


fig14_03.c (Part 2 of 2)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

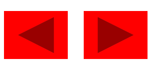
14.5 Notes on Compiling Multiple-Source-File Programs

- Programs with multiple source files
  - Function definition must be in one file (cannot be split up)
  - Global variables accessible to functions in same file
    - Global variables must be defined in every file in which they are used
  - Example:
    - If integer `flag` is defined in one file
    - To use it in another file you must include the statement

```
extern int flag;
```

  - `extern`
    - States that the variable is defined in another file
  - Function prototypes can be used in other files without an extern statement
    - Have a prototype in each file that uses the function

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



14.5 Notes on Compiling Multiple-Source-File Programs

- Keyword `static`
  - Specifies that variables can only be used in the file in which they are defined
- Programs with multiple source files
  - Tedious to compile everything if small changes have been made to only one file
  - Can recompile only the changed files
  - Procedure varies on system
    - UNIX: `make` utility

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

14.6 Program Termination with `exit` and `atexit`

- Function `exit`
  - Forces a program to terminate
  - Parameters – symbolic constants `EXIT_SUCCESS` or `EXIT_FAILURE`
  - Returns an implementation-defined value
  - Example:

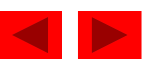
```
exit( EXIT_SUCCESS );
```

- Function `atexit`

```
atexit( functionToRun );
```

  - Registers `functionToRun` to execute upon successful program termination
    - `atexit` itself does not terminate the program
  - Register up to 32 functions (multiple `atexit()` statements)
    - Functions called in reverse register order
  - Called function cannot take arguments or return values

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



```c
1  /* Fig. 14.4: fig14_04.c
2     Using the exit and atexit functions */
3  #include <stdio.h>
4  #include <stdlib.h>
5
6  void print( void ); /* prototype */
7
8  int main()
9  {
10    int answer; /* user's menu choice */
11
12    atexit( print ); /* register function print */
13    printf( "Enter 1 to terminate program with function exit"
14            "\nEnter 2 to terminate program normally\n" );
15    scanf( "%d", &answer );
16
17    /* exit if answer is 1 */
18    if ( answer == 1 ) {
19       printf( "\nTerminating program with function exit\n" );
20       exit( EXIT_SUCCESS );
21    } /* end if */
22
23    printf( "\nTerminating program by reaching the end of main\n" );
24
```

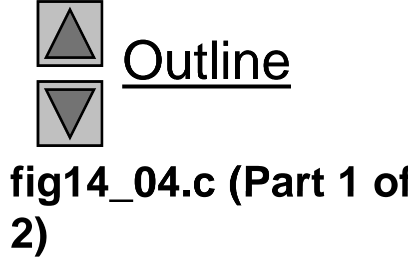


fig14_04.c (Part 1 of 2)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
25    return 0; /* indicates successful termination */
26
27 } /* end main */
28
29 /* display message before termination */
30 void print( void )
31 {
32    printf( "Executing function print at program "
33            "termination\nProgram terminated\n" );
34 } /* end function print */
```

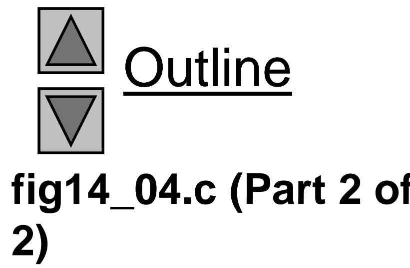


fig14_04.c (Part 2 of 2)

Program Output

```
Enter 1 to terminate program with function exit
Enter 2 to terminate program normally
: 1

Terminating program with function exit
Executing function print at program termination
Program terminated
```

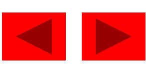
```
Enter 1 to terminate program with function exit
Enter 2 to terminate program normally
: 2

Terminating program by reaching the end of main
Executing function print at program termination
Program terminated
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

14.7 The `volatile` Type Qualifier

- `volatile` qualifier
  - Variable may be altered outside program
  - Variable not under control of program
  - Variable cannot be optimized

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



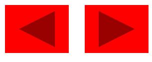
14.8 Suffixes for Integer and Floating-Point Constants

- C provides suffixes for constants
  - `unsigned` integer – `u` or `U`
  - `long` integer – `l` or `L`
  - `unsigned long` integer – `ul`, `lu`, `UL` or `LU`
  - `float` – `f` or `F`
  - `long double` – `l` or `L`
  - Examples:

```
174u
467L
3451ul
```

  - If integer constant is not suffixed type determined by first type capable of storing a value of that size (`int`, `long int`, `unsigned long int`)
  - If floating point not suffixed of type `double`

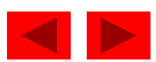
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



14.9 More on Files

- C can process binary files
  - Not all systems support binary files
    - Files opened as text files if binary mode not supported
  - Binary files should be used when rigorous speed, storage, and compatibility conditions demand it
  - Otherwise, text files are preferred
    - Inherent portability, can use standard tools to examine data
- Function `tmpfile`
  - Opens a temporary file in mode `"wb+"`
    - Some systems may process temporary files as text files
  - Temporary file exists until closed with `fclose` or until program terminates
- Function `rewind`
  - Positions file pointers to the beginning of the file

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

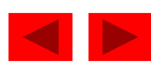


14.9 More on Files

- File open modes:

Mode	Description
`rb`	Open a binary file for reading.
`wb`	Create a binary file for writing. If the file already exists, discard the current contents.
`ab`	Append; open or create a binary file for writing at end-of-file.
`rb+`	Open a binary file for update (reading and writing).
`wb+`	Create a binary file for update. If the file already exists, discard the current contents.
`ab+`	Append; open or create a binary file for update; all writing is done at the end of the file

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



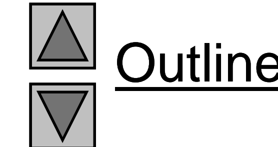


fig14_06.c (Part 1 of 2)

```
1  /* Fig. 14.6: fig14_06.c
2     Using temporary files */
3  #include <stdio.h>
4
5  int main()
6  {
7     FILE *filePtr;       /* pointer to file being modified */
8     FILE *tempFilePtr;   /* temporary file pointer */
9     int c;               /* define c to hold characters input by user */
10    char fileName[ 30 ]; /* create char array */
11
12    printf( "This program changes tabs to spaces.\n"
13            "Enter a file to be modified: " );
14    scanf( "%29s", fileName );
15
16    /* fopen opes the file */
17    if ( ( filePtr = fopen( fileName, "r+" ) ) != NULL ) {
18
19       /* create temporary file */
20       if ( ( tempFilePtr = tmpfile() ) != NULL ) {
21          printf( "\nThe file before modification is:\n" );
22
23          /* read characters */
24          while ( ( c = getc(filePtr) ) != EOF ) {
25             putchar( c );
26             putc( c == '\t' ? ' ': c, tempFilePtr );
27          } /* end while */
28
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
29          rewind( tempFilePtr );
30          rewind( filePtr );
31          printf( "\n\nThe file after modification is:\n" );
32
33          /* read characters */
34          while ( ( c = getc(tempFilePtr) ) != EOF ) {
35             putchar( c );
36             putc( c, filePtr );
37          } /* end while */
38
39       } /* end if */
40       else { /* if temporary file could not be opened */
41          printf( "Unable to open temporary file\n" );
42       } /* end else */
43
44    } /* end if */
45    else { /* if file could not be opened */
46       printf( "Unable to open %s\n", fileName );
47    } /* end else */
48
49    return 0; /* indicates successful termination */
50
51 } /* end main */
```

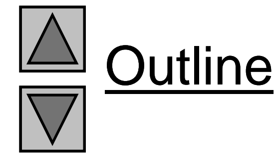
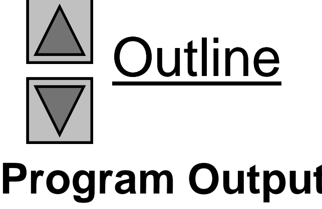


fig14_06.c (Part 2 of 2)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
This program changes tabs to spaces.
Enter a file to be modified: data.txt

The file before modification is:
0       1       2       3       4
        5       6       7       8       9


The file after modification is:
0 1 2 3 4
 5 6 7 8 9
```



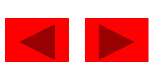
Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

14.10 Signal Handling

- Signal
  - Unexpected event, can terminate program
    - Interrupts (<*ctrl*> *c*), illegal instructions, segmentation violations, termination orders, floating-point exceptions (division by zero, multiplying large floats)
- Function `signal`
  - Traps unexpected events
  - Header `<signal.h>`
  - Receives two arguments a signal number and a pointer to the signal handling function
- Function `raise`
  - Takes an integer signal number and creates a signal

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

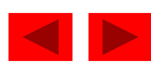


14.10 Signal Handling

- Signals defined in `signal.h`

Signal	Explanation
`SIGABRT`	Abnormal termination of the program (such as a call to `abort`).
`SIGFPE`	An erroneous arithmetic operation, such as a divide by zero or an operation resulting in overflow.
`SIGILL`	Detection of an illegal instruction.
`SIGINT`	Receipt of an interactive attention signal.
`SIGSEGV`	An invalid access to storage.
`SIGTERM`	A termination request sent to the program.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Outline

fig14_08.c (Part 1 of 3)

```
1  /* Fig. 14.8: fig14_08.c
2     Using signal handling */
3  #include <stdio.h>
4  #include <signal.h>
5  #include <stdlib.h>
6  #include <time.h>
7
8  void signal_handler( int signalValue ); /* prototype */
9
10 int main()
11 {
12    int i; /* counter */
13    int x; /* variable to hold random values between 1-50 */
14
15    signal( SIGINT, signal_handler );
16    srand( clock() );
17
18    /* output numbers 1 to 100 */
19    for ( i = 1; i <= 100; i++ ) {
20       x = 1 + rand() % 50; /* generate random number to raise SIGINT */
21
22       /* raise SIGINT when x is 25 */
23       if ( x == 25 ) {
24          raise( SIGINT );
25       } /* end if */
26
```

signal set to call function signal_handler when a signal of type SIGINT occurs.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
27       printf( "%4d", i );
28
29       /* output \n when i is a multiple of 10 */
30       if ( i % 10 == 0 ) {
31          printf( "\n" );
32       } /* end if */
33
34    } /* end for */
35
36    return 0; /* indicates successful termination */
37
38 } /* end main */
39
40 /* handles signal */
41 void signal_handler( int signalValue )
42 {
43    int response; /* user's response to signal (1 or 2) */
44
45    printf( "%s%d%s\n%s",
46       "\nInterrupt signal ( ", signalValue, " ) received.",
47       "Do you wish to continue ( 1 = yes or 2 = no )? " );
48
49    scanf( "%d", &response );
50
```

Outline

fig14_08.c (Part 2 of 3)

User given option of terminating program

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Outline

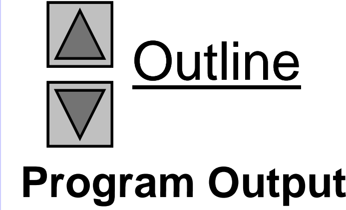
fig14_08.c (Part 3 of 3)

```
51    /* check for invalid responses */
52    while ( response != 1 && response != 2 ) {
53       printf( "( 1 = yes or 2 = no )? " );
54       scanf( "%d", &response );
55    } /* end while */
56
57    /* determine if it is time to exit */
58    if ( response == 1 ) {
59
60       /* call signal and pass it SIGINT and address of signalHandler */
61       signal( SIGINT, signal_handler );
62    } /* end if */
63    else {
64       exit( EXIT_SUCCESS );
65    } /* end else */
66
67 } /* end function signalHandler */
```

Signal handler reinitialized by calling `signal` again

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
   1   2   3   4   5   6   7   8   9  10
  11  12  13  14  15  16  17  18  19  20
  21  22  23  24  25  26  27  28  29  30
  31  32  33  34  35  36  37  38  39  40
  41  42  43  44  45  46  47  48  49  50
  51  52  53  54  55  56  57  58  59  60
  61  62  63  64  65  66  67  68  69  70
  71  72  73  74  75  76  77  78  79  80
  81  82  83  84  85  86  87  88  89  90
  91  92  93
Interrupt signal ( 2 ) received.
Do you wish to continue ( 1 = yes or 2 = no )? 1
  94  95  96
Interrupt signal ( 2 ) received.
Do you wish to continue ( 1 = yes or 2 = no )? 2
```



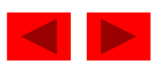
Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

14.11 Dynamic Memory Allocation with `calloc` and `realloc`

- Dynamic memory allocation
  - Can create dynamic arrays
- `calloc(` *nmembers, size* `)`
  - *nmembers* – number of elements
  - *size* – size of each element
  - Returns a pointer to a dynamic array
- `realloc(` *pointerToObject, newSize* `)`
  - *pointerToObject* – pointer to the object being reallocated
  - *newSize* – new size of the object
  - Returns pointer to reallocated memory
  - Returns `NULL` if cannot allocate space
  - If `newSize` equals `0` then the object pointed to is freed
  - If `pointerToObject` equals `0` then it acts like `malloc`

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



14.12 The Unconditional Branch: `goto`

- Unstructured programming
  - Use when performance crucial
  - `break` to exit loop instead of waiting until condition becomes `false`
- `goto` statement
  - Changes flow control to first statement after specified label
  - A label is an identifier followed by a colon (i.e. `start:`)
  - Quick escape from deeply nested loop

```
goto start;
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```c
1  /* Fig. 14.9: fig14_09.c
2     Using goto */
3  #include <stdio.h>
4
5  int main()
6  {
7     int count = 1; /* initialize count */
8
9     start: /* label */
10
11       if ( count > 10 ) {
12          goto end;
13       } /* end if */
14
15       printf( "%d  ", count );
16       count++;
17
18       goto start; /* goto start on line 9 */
19
20    end: /* label */
21       putchar( '\n' );
22
23    return 0; /* indicates successful termination */
24
25 } /* end main */
```

Notice how `start:` , `end:` and `goto` are used

```
1  2  3  4  5  6  7  8  9  10
```

Outline

fig14_09.c

Program Output

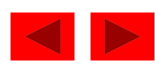
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Chapter 15 - C++ As A "Better C"

Outline

15.1 Introduction
15.2 C++
15.3 A Simple Program: Adding Two Integers
15.4 C++ Standard Library
15.5 Header Files
15.6 Inline Functions
15.7 References and Reference Parameters
15.8 Default Arguments and Empty Parameter Lists
15.9 Unary Scope Resolution Operator
15.10 Function Overloading
15.11 Function Templates

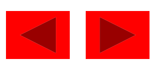
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Objectives

- In this chapter, you will learn:
  - To become familiar with the C++ enhancements to C.
  - To become familiar with the C++ standard library.
  - To understand the concept of `inline` functions.
  - To be able to create and manipulate references.
  - To understand the concept of default arguments.
  - To understand the role the unary scope resolution operator has in scoping.
  - To be able to overload functions.
  - To be able to define functions that can perform similar operations on different types of data.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



15.1 Introduction

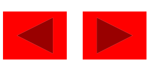
- First 14 Chapters
  - Procedural programming
  - Top-down program design with C
- Chapters 15 to 23
  - C++ portion of book
  - Object based programming (classes, objects, encapsulation)
  - Object oriented programming (inheritance, polymorphism)
  - Generic programming (class and function templates)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

15.2 C++

- C++
  - Improves on many of C's features
  - Has object-oriented capabilities
    - Increases software quality and reusability
  - Developed by Bjarne Stroustrup at Bell Labs
    - Called "C with classes"
    - C++ (increment operator) - enhanced version of C
  - Superset of C
    - Can use a C++ compiler to compile C programs
    - Gradually evolve the C programs to C++
- ANSI C++
  - Final version at http://www.ansi.org/
  - Free, older version at http://www.cygnus.com/misc/wp/

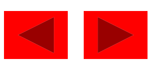
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



15.3 A Simple Program: Adding Two Integers

- File extensions
  - C files: `.c`
  - C++ files: `.cpp` (which we use), `.cxx`, `.C` (uppercase)
- Differences
  - C++ allows you to "comment out" a line by preceding it with `//`
  - For example: `// text to ignore`
  - `<iostream>` - input/output stream header file
  - Return types - all functions must declare their return type
    - C does not require it, but C++ does
  - Variables in C++ can be defined almost anywhere
    - In C, required to defined variables in a block, before any executable statements

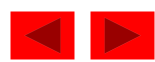
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



15.3 A Simple Program: Adding Two Integers (II)

- Input/Output in C++
  - Performed with streams of characters
  - Streams sent to input/output objects
- Output
  - `std::cout` - standard output stream (connected to screen)
  - `<<` stream insertion operator ("put to")
  - `std::cout << "hi";`
    - Puts `"hi"` to `std::cout`, which prints it on the screen
- Input
  - `std::cin` - standard input object (connected to keyboard)
  - `>>` stream extraction operator ("get from")
  - `std::cin >> myVariable;`
    - Gets stream from keyboard and puts it into `myVariable`

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



15.3 A Simple Program: Adding Two Integers (III)

- `std::endl`
  - "end line"
  - Stream manipulator - prints a newline and flushes output buffer
    - Some systems do not display output until "there is enough text to be worthwhile"
    - `std::endl` forces text to be displayed
- `using` statements
  - Allow us to remove the `std::` prefix
  - Discussed later
- Cascading
  - Can have multiple `<<` or `>>` operators in a single statement

```
std::cout << "Hello " << "there" << std::endl;
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

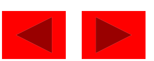


fig15_01.cpp

```
1  // Fig. 15.1: fig15_01.cpp
2  // Addition program
3  #include <iostream>
4
5  int main()
6  {
7     int integer1;
8
9     std::cout << "Enter first integer\n";
10    std::cin >> integer1;
11
12    int integer2, sum;           // declaration
13
14    std::cout << "Enter second integer\n";
15    std::cin >> integer2;
16    sum = integer1 + integer2;
17    std::cout << "Sum is " << sum << std::endl;
18
19    return 0;   // indicate that program ended successfully
20 } // end function main
```

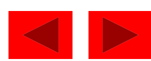
```
Enter first integer
45
Enter second integer
72
Sum is 117
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

15.4 C++ Standard Library

- C++ programs built from
  - Functions
  - Classes
    - Most programmers use library functions
- Two parts to learning C++
  - Learn the language itself
  - Learn the library functions
- Making your own functions
  - Advantage: you know exactly how they work
  - Disadvantage: time consuming, difficult to maintain efficiency and design well

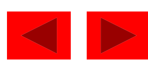
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



15.5 Header Files

- Header files
  - Each standard library has header files
    - Contain function prototypes, data type definitions, and constants
  - Files ending with `.h` are "old-style" headers
- User defined header files
  - Create your own header file
    - End it with `.h`
  - Use `#include "myFile.h"` in other files to load your header

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

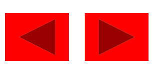


15.5 Header Files

Standard library header file	Explanation
<cassert>	Contains macros and information for adding diagnostics that aid program debugging. The old version of this header file is <assert.h>.
<cctype>	Contains function prototypes for functions that test characters for certain properties, that can be used to convert lowercase letters to uppercase letters and vice versa. This header file replaces header file <ctype.h>.
<cfloat>	Contains the floating-point size limits of the system. This header file replaces header file <float.h>.
<climits>	Contains the integral size limits of the system. This header file replaces header file <limits.h>.
<cmath>	Contains function prototypes for math library functions. This header file replaces header file <math.h>.
<cstdio>	Contains function prototypes for the standard input/output library functions and information used by them. This header file replaces header file <stdio.h>.
<cstdlib>	Contains function prototypes for conversions of numbers to text, text to numbers, memory allocation, random numbers and various other utility functions. This header file replaces header file <stdlib.h>.
<cstring>	Contains function prototypes for C-style string processing functions. This header file replaces header file <string.h>.
<ctime>	Contains function prototypes and types for manipulating the time and date. This header file replaces header file <time.h>.
<iostream>	Contains function prototypes for the standard input and standard output functions. This header file replaces header file <iostream.h>.

Fig. 15.2 Standard library header files. (Part 1 of 3)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

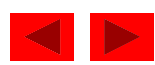


15.5 Header Files

Standard library header file	Explanation
<iomanip>	Contains function prototypes for the stream manipulators that enable formatting of streams of data. This header file replaces <iomanip.h>.
<fstream>	Contains function prototypes for functions that perform input from files on disk and output to files on disk. This header file replaces header file <fstream.h>.
<utility>	Contains classes and functions that are used by many standard library header files.
<vector>, <list>, <deque>, <queue>, <stack>, <map>, <set>, <bitset>	These header files contain classes that implement the standard library containers. Containers are used to store data during a program's execution.
<functional>	Contains classes and functions used by standard library algorithms.
<memory>	Contains classes and functions used by the standard library to allocate memory to the standard library containers.
<iterator>	Contains classes for accessing data in the standard library containers.
<algorithm>	Contains functions for manipulating data in standard library containers.
<exception>, <stdexcept>	These header files contain classes that are used for exception handling (discussed in Chapter 23).

Fig. 15.2 Standard library header files. (Part 2 of 3)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

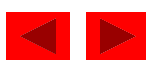


15.5 Header Files

Standard library header file	Explanation
<string>	Contains the definition of class `string` from the standard library.
<sstream>	Contains prototypes for functions that perform input from strings in memory and output to strings in memory.
<locale>	Contains classes and functions normally used by stream processing to process data in the natural form for different languages (e.g., monetary formats, sorting strings, character presentation, etc.).
<limits>	Contains classes for defining the numerical data type limits on each computer platform.
<typeinfo>	Contains classes for run-time type identification (determining data types at execution time).

Fig. 15.2 Standard library header files. (Part 3 of 3)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



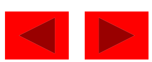
15.6 Inline Functions

- Function calls
  - Cause execution-time overhead
  - Qualifier `inline` before function return type "advises" a function to be inlined
    - Puts copy of function's code in place of function call
  - Speeds up performance but increases file size
  - Compiler can ignore the `inline` qualifier
    - Ignores all but the smallest functions

```
inline double cube( const double s )
   { return s * s * s; }
```

- Using statements
  - By writing `using std::cout;` we can write `cout` instead of `std::cout` in the program
  - Same applies for `std::cin` and `std::endl`

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



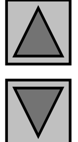
Outline

fig15_03.cpp

```
1  // Fig. 15.3: fig15_03.cpp
2  // Using an inline function to calculate
3  // the volume of a cube.
4  #include <iostream>
5
6  using std::cout;
7  using std::cin;
8  using std::endl;
9
10 inline double cube( const double s ) { return s * s * s; }
11
12 int main()
13 {
14    double side;
15
16    for ( int k = 1; k < 4; k++ ) {
17       cout << "Enter the side length of your cube:  ";
18       cin >> side;
19       cout << "Volume of cube with side "
20            << side << " is " << cube( side ) << endl;
21    } // end for
22
23    return 0;
24 } // end function main
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
Enter the side length of your cube:  1.0
Volume of cube with side 1 is 1
Enter the side length of your cube:  2.3
Volume of cube with side 2.3 is 12.167
Enter the side length of your cube:  5.4
Volume of cube with side 5.4 is 157.464
```



Program Output

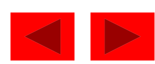
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

15.6 Inline Functions (II)

- bool
  - Boolean - new data type, can either be `true` or `false`

C++ Keywords				
Keywords common to the C and C++ programming languages				
auto	break	case	char	const
continue	default	do	double	else
enum	extern	float	for	goto
if	int	long	register	return
short	signed	sizeof	static	struct
switch	typedef	union	unsigned	void
volatile	while			
C++ only keywords				
asm	bool	catch	class	const_cast
delete	dynamic_cast	explicit	false	friend
inline	mutable	namespace	new	operator
private	protected	public	reinterpret_cast	
static_cast	template	this	throw	true
try	typeid	typename	using	virtual
wchar_t				

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



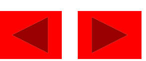
15.7 References and Reference Parameters

- Call by value
  - Copy of data passed to function
  - Changes to copy do not change original
- Call by reference
  - Function can directly access data
  - Changes affect original
- Reference parameter alias for argument
  - Use &

```
void change(int &variable)
  {
        variable += 3;
  }
```

    - Adds 3 to the original variable input
  - `int y = &x`
    - Changing `y` changes `x` as well

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



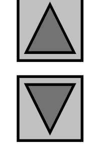
15.7 References and Reference Parameters (II)

- Dangling references
  - Make sure to assign references to variables
  - If a function returns a reference to a variable, make sure the variable is `static`
    - Otherwise, it is automatic and destroyed after function ends
- Multiple references
  - Like pointers, each reference needs an &

```
int &a, &b, &c;
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
1  // Fig. 15.5: fig15_05.cpp
2  // Comparing call-by-value and call-by-reference
3  // with references.
4  #include <iostream>
5
6  using std::cout;
7  using std::endl;
8
9  int squareByValue( int );
10 void squareByReference( int & );
11
12 int main()
13 {
14    int x = 2, z = 4;
15
16    cout << "x = " << x << " before squareByValue\n"
17         << "Value returned by squareByValue: "
18         << squareByValue( x ) << endl
19         << "x = " << x << " after squareByValue\n" << endl;
20
```



Outline

fig15_05.cpp (Part 1 of 2)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
21    cout << "z = " << z << " before squareByReference" << endl;
22    squareByReference( z );
23    cout << "z = " << z << " after squareByReference" << endl;
24
25    return 0;
26 } // end function main
27
28 int squareByValue( int a )
29 {
30    return a *= a;    // caller's argument not modified
31 } // end function squareByValue
32
33 void squareByReference( int &cRef )
34 {
35    cRef *= cRef;     // caller's argument modified
36 } // end function squareByReference
```

```
x = 2 before squareByValue
Value returned by squareByValue: 4
x = 2 after squareByValue

z = 4 before squareByReference
z = 16 after squareByReference
```

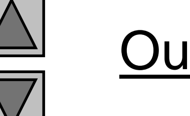


fig15_05.cpp (Part 2 of 2)

Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

fig15_06.cpp

```
1  // Fig. 15.6: fig15_06.cpp
2  // References must be initialized
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  int main()
9  {
10    int x = 3, &y = x;  // y is now an alias for x
11
12    cout << "x = " << x << endl << "y = " << y << endl;
13    y = 7;
14    cout << "x = " << x << endl << "y = " << y << endl;
15
16    return 0;
17 } // end function main
```

```
x = 3
y = 3
x = 7
y = 7
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

fig15_.07.cpp

```
1  // Fig. 15.7: fig15_07.cpp
2  // References must be initialized
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  int main()
9  {
10    int x = 3, &y;    // Error: y must be initialized
11
12    cout << "x = " << x << endl << "y = " << y << endl;
13    y = 7;
14    cout << "x = " << x << endl << "y = " << y << endl;
15
16    return 0;
17 } // end function main
```

Borland C++ command-line compiler error message

```
Error E2304 Fig15_07.cpp 10: Reference variable 'y' must be initialized
in function main()
```

Microsoft Visual C++ compiler error message

```
Fig15_07.cpp(10) : error C2530: 'y' : references must be initialized
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

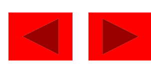
15.8 Default Arguments and Empty Parameter Lists

- If function parameter omitted, gets default value
  - Can be constants, global variables, or function calls
  - If not enough parameters specified, rightmost go to their defaults

- Set defaults in function prototype

```
int myFunction( int x = 1, int y = 2, int z = 3 );
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

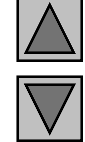


15.8 Default Arguments and Empty Parameter Lists (II)

- Empty parameter lists
  - In C, empty parameter list means function takes any argument
    - In C++ it means function takes no arguments
  - To declare that a function takes no parameters:
    - Write `void` or nothing in parentheses
    - Prototypes:

```
void print1( void );
void print2();
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Outline

fig15_08.cpp (Part 1 of 2)

```
1  // Fig. 15.8: fig15_08.cpp
2  // Using default arguments
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  int boxVolume( int length = 1, int width = 1, int height = 1 );
9
10 int main()
11 {
12    cout << "The default box volume is: " << boxVolume()
13         << "\n\nThe volume of a box with length 10,\n"
14         << "width 1 and height 1 is: " << boxVolume( 10 )
15         << "\n\nThe volume of a box with length 10,\n"
16         << "width 5 and height 1 is: " << boxVolume( 10, 5 )
17         << "\n\nThe volume of a box with length 10,\n"
18         << "width 5 and height 2 is: " << boxVolume( 10, 5, 2 )
19         << endl;
20
21    return 0;
22 } // end function main
23
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
24 // Calculate the volume of a box
25 int boxVolume( int length, int width, int height )
26 {
27    return length * width * height;
28 } // end function boxVolume
```

```
The default box volume is: 1

The volume of a box with length 10,
width 1 and height 1 is: 10

The volume of a box with length 10,
width 5 and height 1 is: 50

The volume of a box with length 10,
width 5 and height 2 is: 100
```

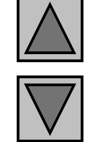


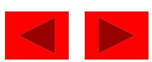
fig15_08.cpp (Part 2 of 2)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

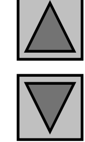
15.9 Unary Scope Resolution Operator

- Unary scope resolution operator (`::`)
  - Access global variables if a local variable has same name
  - Instead of `variable` use `::variable`
- `static_cast<newType> (variable)`
  - Creates a copy of `variable` of type `newType`
  - Convert `int`s to `float`s, etc.
- Stream manipulators
  - Can change how output is formatted
  - `setprecision` - set precision for `float`s (default 6 digits)
  - `setiosflags` - formats output
  - `setwidth` - set field width
  - Discussed in depth in Chapter 21

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



```
1 // Fig. 15.9: fig15_09.cpp
2 // Using the unary scope resolution operator
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
7 using std::ios;
8
9 #include <iomanip>
10
11 using std::setprecision;
12 using std::setiosflags;
13 using std::setw;
14
15 const double PI = 3.14159265358979;
16
17 int main()
18 {
19    const float PI = static_cast< float >( ::PI );
20
21    cout << setprecision( 20 )
22         << "  Local float value of PI = " << PI
23         << "\nGlobal double value of PI = " << ::PI << endl;
24
```



Outline

fig15_09.cpp (Part 1 of 2)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

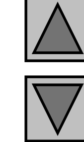
```
25    cout << setw( 28 ) << "Local float value of PI = "
26         << setiosflags( ios::fixed | ios::showpoint )
27         << setprecision( 10 ) << PI << endl;
28    return 0;
29 } // end function main
```

Borland C++ command-line compiler output

```
  Local float value of PI = 3.141592741012573242
Global double value of PI = 3.141592653589790007
  Local float value of PI = 3.1415927410
```

Microsoft Visual C++ compiler output

```
  Local float value of PI = 3.1415927410125732
Global double value of PI = 3.14159265358979
  Local float value of PI = 3.1415927410
```



© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

15.10 Function Overloading

- Function overloading:
  - Functions with same name and different parameters
  - Overloaded functions should perform similar tasks
    - Function to square `int`s and function to square `float`s

```
int square( int x) {return x * x;}
float square(float x) { return x * x; }
```

  - Program chooses function by signature
    - Signature determined by function name and parameter types
    - Type safe linkage - ensures proper overloaded function called

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Outline

fig15_10.cpp

```
1  // Fig. 15.10: fig15_10.cpp
2  // Using overloaded functions
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  int square( int x ) { return x * x; }
9
10 double square( double y ) { return y * y; }
11
12 int main()
13 {
14    cout << "The square of integer 7 is " << square( 7 )
15         << "\nThe square of double 7.5 is " << square( 7.5 )
16         << endl;
17
18    return 0;
19 } // end function main
```

```
The square of integer 7 is 49
The square of double 7.5 is 56.25
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

15.11 Function Templates

- Function templates
  - Compact way to make overloaded functions
  - Keyword `template`
  - Keyword `class` or `typename` before every formal type parameter (built in or user defined)

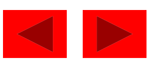
```
template < class T >  //or template< typename T >
T square( T value1)
{
   return value1 * value1;
}
```

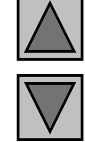
  - `T` replaced by type parameter in function call

```
int x;
int y = square(x);
```

    - If `int` parameter, all `T`'s become `int`s
    - Can use `float`, `double`, `long`...

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.





Outline

fig15_11.cpp (Part 1 of 2)

```
1  // Fig. 15.11: fig15_11.cpp
2  // Using a function template
3  #include <iostream>
4
5  using std::cout;
6  using std::cin;
7  using std::endl;
8
9  template < class T >
10 T maximum( T value1, T value2, T value3 )
11 {
12    T max = value1;
13
14    if ( value2 > max )
15       max = value2;
16
17    if ( value3 > max )
18       max = value3;
19
20    return max;
21 } // end function template maximum
22
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

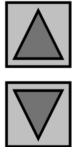
```
23 int main()
24 {
25    int int1, int2, int3;
26
27    cout << "Input three integer values: ";
28    cin >> int1 >> int2 >> int3;
29    cout << "The maximum integer value is: "
30         << maximum( int1, int2, int3 );          // int version
31
32    double double1, double2, double3;
33
34    cout << "\nInput three double values: ";
35    cin >> double1 >> double2 >> double3;
36    cout << "The maximum double value is: "
37         << maximum( double1, double2, double3 ); // double version
38
39    char char1, char2, char3;
40
41    cout << "\nInput three characters: ";
42    cin >> char1 >> char2 >> char3;
43    cout << "The maximum character value is: "
44         << maximum( char1, char2, char3 )        // char version
45         << endl;
46
47    return 0;
48 } // end function main
```

Outline

fig15_11.cpp (Part 2 of 2)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
Input three integer values: 1 2 3
The maximum integer value is: 3
Input three double values: 3.3 2.2 1.1
The maximum double value is: 3.3
Input three characters: A C B
The maximum character value is: C
```



Outline

Program Output

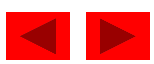
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Chapter 16: Classes and Data Abstraction

Outline

16.1 Introduction
16.2 Implementing a Time Abstract Data Type with a Class
16.3 Class Scope and Accessing Class Members
16.4 Separating Interface from Implementation
16.5 Controlling Access to Members
16.6 Access Functions and Utility Functions
16.7 Initializing Class Objects: Constructors
16.8 Using Default Arguments with Constructors
16.9 Using Destructors
16.10 When Constructors and Destructors Are Called
16.11 Using Data Members and Member Functions
16.12 A Subtle Trap: Returning a Reference to a private Data Member
16.13 Assignment by Default Memberwise Copy
16.14 Software Reusability

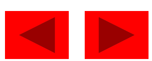
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Objectives

- In this chapter, you will learn:
  - To understand the software engineering concepts of en-capsulation and data hiding.
  - To understand the notions of data abstraction and ab-stract data types (ADTs).
  - To be able to create C++ ADTs, namely classes.
  - To understand how to create, use, and destroy class objects.
  - To be able to control access to object data members and member functions.
  - To begin to appreciate the value of object orientation.

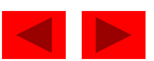
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



16.1 Introduction

- Object-oriented programming (OOP)
  - *Encapsulates* data (attributes) and functions (behavior) into packages called *classes*
  - Data and functions closely related
- Information hiding
  - Implementation details are hidden within the classes themselves
- Unit of C++ programming: the class
  - A class is like a blueprint – reusable
  - Objects are *instantiated* (created) from the class
  - For example, a house is an instance of a "blueprint class"
  - C programmers concentrate on functions

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



16.2 Implementing a Time Abstract Data Type with a Class

- Classes
  - Model objects that have attributes (data members) and behaviors (member functions)
  - Defined using keyword `class`

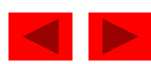
```
1  class Time {
2  public:
3     Time();
4     void setTime( int, int, int );
5     void printMilitary();
6     void printStandard();
7  private:
8     int hour;     // 0 - 23
9     int minute;   // 0 - 59
10    int second;   // 0 - 59
11 }; // end class Time
```

`Public:` and `Private:` are member-access specifiers.

`setTime`, `printMilitary`, and `printStandard` are member functions. `Time` is the constructor.

`hour`, `minute`, and `second` are data members.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



16.2 Implementing a Time Abstract Data Type with a Class (II)

- Format
  - Body delineated with braces ({ and })
  - Class definition terminates with a semicolon

- Member functions and data

  Public - accessible wherever the program has access to an object of class Time

  Private - accessible only to member functions of the class

  Protected - discussed later in the course

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

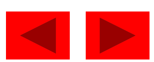
16.2 Implementing a Time Abstract Data Type with a Class (III)

- Constructor
  - Special member function that initializes data members of a class object
  - Constructors cannot return values
  - Same name as the class
- Definitions
  - Once class defined, can be used as a data type

```
Time sunset,                  // object of type Time
     arrayOfTimes[ 5 ],       // array of Time objects
     *pointerToTime,          // pointer to a Time object
     &dinnerTime = sunset;    // reference to a Time object
```

Note: The class name becomes the new type specifier.

...itel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



16.2 Implementing a Time Abstract Data Type with a Class (IV)

- Binary scope resolution operator (: :)
  - Specifies which class owns the member function
  - Different classes can have the same name for member functions
- Format for definition class member functions

ReturnType ClassName: : *MemberFunctionName( ){*

…

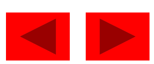
}

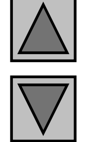
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

16.2 Implementing a Time Abstract Data Type with a Class (V)

- If member function is defined *inside* the class
  - Scope resolution operator and class name are not needed
  - Defining a function outside a class does not change it being `public` or `private`
- Classes encourage software reuse
  - Inheritance allows new classes to be derived from old ones
- In following program
  - `Time` constructor initializes the data members to 0
    - Ensures that the object is in a consistent state when it is created

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.





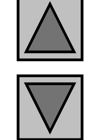
Outline

fig16_02.cpp (Part 1 of 3)

```
1  // Fig. 16.2: fig16_02.cpp
2  // Time class.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  // Time abstract data type (ADT) definition
9  class Time {
10  public:
11     Time();                          // constructor
12     void setTime( int, int, int ); // set hour, minute, second
13     void printMilitary();            // print military time format
14     void printStandard();            // print standard time format
15  private:
16     int hour;      // 0 - 23
17     int minute;    // 0 - 59
18     int second;    // 0 - 59
19  }; // end class Time
20
21  // Time constructor initializes each data member to zero.
22  // Ensures all Time objects start in a consistent state.
23  Time::Time() { hour = minute = second = 0; }
24
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
25 // Set a new Time value using military time. Perform validity
26 // checks on the data values. Set invalid values to zero.
27 void Time::setTime( int h, int m, int s )
28 {
29    hour = ( h >= 0 && h < 24 ) ? h : 0;
30    minute = ( m >= 0 && m < 60 ) ? m : 0;
31    second = ( s >= 0 && s < 60 ) ? s : 0;
32 } // end function setTime
33
34 // Print Time in military format
35 void Time::printMilitary()
36 {
37    cout << ( hour < 10 ? "0" : "" ) << hour << ":"
38         << ( minute < 10 ? "0" : "" ) << minute;
39 } // end function printMilitary
40
41 // Print Time in standard format
42 void Time::printStandard()
43 {
44    cout << ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 )
45         << ":" << ( minute < 10 ? "0" : "" ) << minute
46         << ":" << ( second < 10 ? "0" : "" ) << second
47         << ( hour < 12 ? " AM" : " PM" );
48 } // end function printStandard
49
```



© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
50 // Driver to test simple class Time
51 int main()
52 {
53    Time t;  // instantiate object t of class Time
54
55    cout << "The initial military time is ";
56    t.printMilitary();
57    cout << "\nThe initial standard time is ";
58    t.printStandard();
59
60    t.setTime( 13, 27, 6 );
61    cout << "\n\nMilitary time after setTime is ";
62    t.printMilitary();
63    cout << "\nStandard time after setTime is ";
64    t.printStandard();
65
66    t.setTime( 99, 99, 99 );  // attempt invalid settings
67    cout << "\n\nAfter attempting invalid settings:"
68         << "\nMilitary time: ";
69    t.printMilitary();
70    cout << "\nStandard time: ";
71    t.printStandard();
72    cout << endl;
73    return 0;
74 } // end function main
```

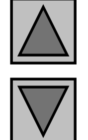
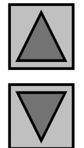


fig16_02.cpp (Part 3 of 3)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
The initial military time is 00:00
The initial standard time is 12:00:00 AM

Military time after setTime is 13:27
Standard time after setTime is 1:27:06 PM

After attempting invalid settings:
Military time: 00:00
Standard time: 12:00:00 AM
```



Outline

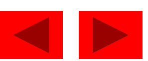
Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

16.3 Class Scope and Accessing Class Members

- Class scope
  - Data members and member functions
- File scope
  - Nonmember functions
- Function scope
  - Variables defined in member functions, destroyed after function completes
- Inside a scope
  - Members accessible by all member functions
  - Referenced by name

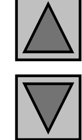
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



16.3 Class Scope and Accessing Class Members (II)

- Outside a scope
  - Use handles
    - An object name, a reference to an object or a pointer to an object
- Accessing class members
  - Same as `structs`
  - Dot (`.`) for objects and arrow (`->`) for pointers
  - Example: `t.hour` is the hour element of `t`
  - `TimePtr->hour` is the hour element

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Outline

fig16_03.cpp (Part 1 of 2)

```
1  // Fig. 16.3: fig16_03.cpp
2  // Demonstrating the class member access operators . and ->
3  //
4  // CAUTION: IN FUTURE EXAMPLES WE AVOID PUBLIC DATA!
5  #include <iostream>
6
7  using std::cout;
8  using std::endl;
9
10 // Simple class Count
11 class Count {
12 public:
13    int x;
14    void print() { cout << x << endl; }
15 }; // end class Count
16
17 int main()
18 {
19    Count counter,                // create counter object
20          *counterPtr = &counter, // pointer to counter
21          &counterRef = counter;  // reference to counter
22
23    cout << "Assign 7 to x and print using the object's name: ";
24    counter.x = 7;        // assign 7 to data member x
25    counter.print();      // call member function print
26
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
27    cout << "Assign 8 to x and print using a reference: ";
28    counterRef.x = 8;     // assign 8 to data member x
29    counterRef.print();   // call member function print
30
31    cout << "Assign 10 to x and print using a pointer: ";
32    counterPtr->x = 10;   // assign 10 to data member x
33    counterPtr->print();  // call member function print
34    return 0;
35 } // end function main
```

```
Assign 7 to x and print using the object's name: 7
Assign 8 to x and print using a reference: 8
Assign 10 to x and print using a pointer: 10
```

Outline

fig16_03.cpp (Part 2 of 2)

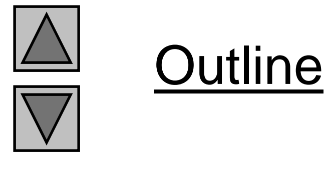
Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

16.4 Separating Interface from Implementation

- Separating interface from implementation
  - Easier to modify programs
  - C++ programs can be split into
    - *Header files* – contains class definitions and function prototypes
    - *Source-code files* – contains member function definitions
- Program Outline:
  - Using the same `Time` class as before, create a header file
  - Create a source code file
    - Load the header file to get the class definitions
    - Define the member functions of the class

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

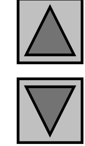


time1.h

```
1  // Fig. 16.4: time1.h
2  // Declaration of the Time class.
3  // Member functions are defined in time1.cpp
4
5  // prevent multiple inclusions of header file
6  #ifndef TIME1_H
7  #define TIME1_H
8
9  // Time abstract data type definition
10 class Time {
11 public:
12    Time();                        // constructor
13    void setTime( int, int, int ); // set hour, minute, second
14    void printMilitary();          // print military time format
15    void printStandard();          // print standard time format
16 private:
17    int hour;     // 0 - 23
18    int minute;   // 0 - 59
19    int second;   // 0 - 59
20 }; // end class Time
21
22 #endif
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

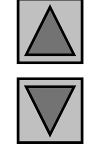
```
23 // Fig. 16.4: time1.cpp
24 // Member function definitions for Time class.
25 #include <iostream>
26
27 using std::cout;
28
29 #include "time1.h"
30
31 // Time constructor initializes each data member to zero.
32 // Ensures all Time objects start in a consistent state.
33 Time::Time() { hour = minute = second = 0; }
34
35 // Set a new Time value using military time. Perform validity
36 // checks on the data values. Set invalid values to zero.
37 void Time::setTime( int h, int m, int s )
38 {
39    hour   = ( h >= 0 && h < 24 ) ? h : 0;
40    minute = ( m >= 0 && m < 60 ) ? m : 0;
41    second = ( s >= 0 && s < 60 ) ? s : 0;
42 } // end function setTime
43
```



time1.cpp (Part 1 of 2)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
44 // Print Time in military format
45 void Time::printMilitary()
46 {
47    cout << ( hour < 10 ? "0" : "" ) << hour << ":"
48         << ( minute < 10 ? "0" : "" ) << minute;
49 } // end function printMilitary
50
51 // Print time in standard format
52 void Time::printStandard()
53 {
54    cout << ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 )
55         << ":" << ( minute < 10 ? "0" : "" ) << minute
56         << ":" << ( second < 10 ? "0" : "" ) << second
57         << ( hour < 12 ? " AM" : " PM" );
58 } // end function printStandard
```



time1.cpp (Part 2 of 2)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
59 // Fig. 16.4: fig16_04.cpp
60 // Driver for Time1 class
61 // NOTE: Compile with time1.cpp
62 #include <iostream>
63
64 using std::cout;
65 using std::endl;
66
67 #include "time1.h"
68
69 // Driver to test simple class Time
70 int main()
71 {
72    Time t;  // instantiate object t of class time
73
74    cout << "The initial military time is ";
75    t.printMilitary();
76    cout << "\nThe initial standard time is ";
77    t.printStandard();
78
79    t.setTime( 13, 27, 6 );
80    cout << "\n\nMilitary time after setTime is ";
81    t.printMilitary();
82    cout << "\nStandard time after setTime is ";
83    t.printStandard();
84
```

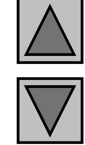


fig16_04.cpp (Part 1 of 2)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
85     t.setTime( 99, 99, 99 );  // attempt invalid settings
86     cout << "\n\nAfter attempting invalid settings:\n"
87          << "Military time: ";
88     t.printMilitary();
89     cout << "\nStandard time: ";
90     t.printStandard();
91     cout << endl;
92     return 0;
93  } // end function main
```

```
The initial military time is 00:00
The initial standard time is 12:00:00 AM

Military time after setTime is 13:27
Standard time after setTime is 1:27:06 PM

After attempting invalid settings:
Military time: 00:00
Standard time: 12:00:00 AM
```

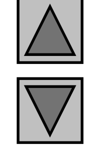


fig16_04.cpp (Part 2 of 2)

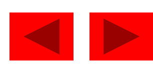
Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

16.5 Controlling Access to Members

- Purpose of `public`
  - Give clients a view of the *services* the class provides (interface)
- Purpose of `private`
  - Default setting
  - Hide details of how the class accomplishes its tasks (implementation)
  - `Private` members only accessible through the `public` interface using `public` member functions

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

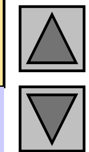


Outline

fig16_05.cpp

```
1  // Fig. 16.5: fig16_05.cpp
2  // Demonstrate errors resulting from attempts
3  // to access private class members.
4  #include <iostream>
5
6  using std::cout;
7
8  #include "time1.h"
9
10 int main()
11 {
12    Time t;
13
14    // Error: 'Time::hour' is not accessible
15    t.hour = 7;
16
17    // Error: 'Time::minute' is not accessible
18    cout << "minute = " << t.minute;
19
20    return 0;
21 } // end function main
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Outline

Program Output

Borland C++ command-line compiler error messages

```
Time1.cpp:
Fig16_05.cpp:
Error E2247 Fig16_05.cpp 15:
   'Time::hour' is not accessible in function main()
Error E2247 Fig16_05.cpp 18:
   'Time::minute' is not accessible in function main()

*** 2 errors in Compile ***
```

Microsoft Visual C++ compiler error messages

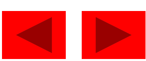
```
Compiling...
Fig16_05.cpp
D:\Fig16_05.cpp(15) : error C2248: 'hour' : cannot access private
member declared in class 'Time'
D:\Fig16_05\time1.h(18) : see declaration of 'hour'
D:\Fig16_05.cpp(18) : error C2248: 'minute' : cannot access private
member declared in class 'Time'
D:\time1.h(19) : see declaration of 'minute'
Error executing cl.exe.

test.exe - 2 error(s), 0 warning(s)
```

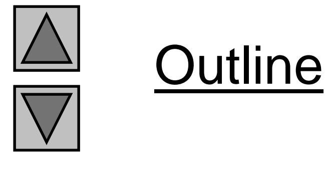
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

16.6 Access Functions and Utility Functions

- Utility functions
  - `private` functions that support the operation of public functions
  - Not intended to be used directly by clients
- Access functions
  - `public` functions that read/display data or check conditions
  - For a container, it could call the `isEmpty` function
- Next
  - Program to take in monthly sales and output the total
  - Implementation not shown, only access functions

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.





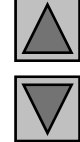
Outline

salesp.h

```
1  // Fig. 16.6: salesp.h
2  // SalesPerson class definition
3  // Member functions defined in salesp.cpp
4  #ifndef SALESP_H
5  #define SALESP_H
6
7  class SalesPerson {
8  public:
9     SalesPerson();                // constructor
10    void getSalesFromUser();      // get sales figures from keyboard
11    void setSales( int, double ); // User supplies one month's
12                                  // sales figures.
13    void printAnnualSales();
14
15 private:
16    double totalAnnualSales();    // utility function
17    double sales[ 12 ];           // 12 monthly sales figures
18 }; // end class SalesPerson
19
20 #endif
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

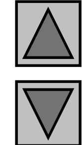
```
21 // Fig. 16.6: salesp.cpp
22 // Member functions for class SalesPerson
23 #include <iostream>
24
25 using std::cout;
26 using std::cin;
27 using std::endl;
28
29 #include <iomanip>
30
31 using std::setprecision;
32 using std::setiosflags;
33 using std::ios;
34
35 #include "salesp.h"
36
37 // Constructor function initializes array
38 SalesPerson::SalesPerson()
39 {
40    for ( int i = 0; i < 12; i++ )
41       sales[ i ] = 0.0;
42 } // end SalesPerson constructor
43
```



salesp.cpp (Part 1 of 3)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

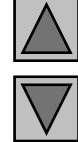
```
44 // Function to get 12 sales figures from the user
45 // at the keyboard
46 void SalesPerson::getSalesFromUser()
47 {
48    double salesFigure;
49
50    for ( int i = 1; i <= 12; i++ ) {
51       cout << "Enter sales amount for month " << i << ": ";
52
53       cin >> salesFigure;
54       setSales( i, salesFigure );
55    } // end for
56 } // end function getSalesFromUser
57
58 // Function to set one of the 12 monthly sales figures.
59 // Note that the month value must be from 0 to 11.
60 void SalesPerson::setSales( int month, double amount )
61 {
62    if ( month >= 1 && month <= 12 && amount > 0 )
63       sales[ month - 1 ] = amount; // adjust for subscripts 0-11
64    else
65       cout << "Invalid month or sales figure" << endl;
66 } // end function setSales
67
```



salesp.cpp (Part 2 of 3)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
68 // Print the total annual sales
69 void SalesPerson::printAnnualSales()
70 {
71    cout << setprecision( 2 )
72         << setiosflags( ios::fixed | ios::showpoint )
73         << "\nThe total annual sales are: $"
74         << totalAnnualSales() << endl;
75 } // end function printAnnualSales
76
77 // Private utility function to total annual sales
78 double SalesPerson::totalAnnualSales()
79 {
80    double total = 0.0;
81
82    for ( int i = 0; i < 12; i++ )
83       total += sales[ i ];
84
85    return total;
86 } // end function totalAnnualSales
```



salesp.cpp (Part 3 of 3)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
87 // Fig. 16.6: fig16_06.cpp
88 // Demonstrating a utility function
89 // Compile with salesp.cpp
90 #include "salesp.h"
91
92 int main()
93 {
94    SalesPerson s;         // create SalesPerson object s
95
96    s.getSalesFromUser();  // note simple sequential code
97    s.printAnnualSales();  // no control structures in main
98    return 0;
99 } // end function main
```

```
Enter sales amount for month 1: 5314.76
Enter sales amount for month 2: 4292.38
Enter sales amount for month 3: 4589.83
Enter sales amount for month 4: 5534.03
Enter sales amount for month 5: 4376.34
Enter sales amount for month 6: 5698.45
Enter sales amount for month 7: 4439.22
Enter sales amount for month 8: 5893.57
Enter sales amount for month 9: 4909.67
Enter sales amount for month 10: 5123.45
Enter sales amount for month 11: 4024.97
Enter sales amount for month 12: 5923.92

The total annual sales are: $60120.59
```

Outline

fig16_06.cpp

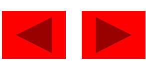
Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

16.7 Initializing Class Objects: Constructors

- Constructor function
  - Can initialize class members
  - Same name as the class, no return type
  - Member variables can be initialized by the constructor or set afterwards
- Defining objects
  - Initializers can be provided
  - Initializers passed as arguments to the class' constructor

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



16.7 Initializing Class Objects: Constructors (II)

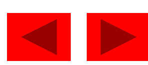
- Format

  Type ObjectName(*value1, value2, …*);

  - Constructor assigns *value1*, *value2*, etc. to its member variables
  - If not enough values specified, rightmost parameters set to their default (specified by programmer)

```
myClass myObject( 3, 4.0 );
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



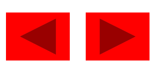
16.8 Using Default Arguments with Constructors

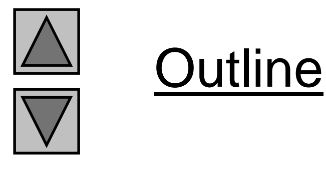
- Default constructor
  - One per class
  - Can be invoked without arguments
  - Has default arguments
- Default arguments
  - Set in default constructor function prototype (in class definition)
    - Do not set defaults in the function definition, outside of a class
  - Example:

```
SampleClass( int = 0, float = 0);
```

    - Constructor has same name as class

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



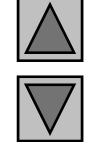


time2.h

```
1  // Fig. 16.7: time2.h
2  // Declaration of the Time class.
3  // Member functions are defined in time2.cpp
4
5  // preprocessor directives that
6  // prevent multiple inclusions of header file
7  #ifndef TIME2_H
8  #define TIME2_H
9
10 // Time abstract data type definition
11 class Time {
12 public:
13    Time( int = 0, int = 0, int = 0 );  // default constructor
14    void setTime( int, int, int ); // set hour, minute, second
15    void printMilitary();          // print military time format
16    void printStandard();          // print standard time format
17 private:
18    int hour;     // 0 - 23
19    int minute;   // 0 - 59
20    int second;   // 0 - 59
21 }; // end class Time
22
23 #endif
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

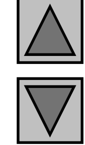
```
24 // Fig. 16.7: time2.cpp
25 // Member function definitions for Time class.
26 #include <iostream>
27
28 using std::cout;
29
30 #include "time2.h"
31
32 // Time constructor initializes each data member to zero.
33 // Ensures all Time objects start in a consistent state.
34 Time::Time( int hr, int min, int sec )
35    { setTime( hr, min, sec ); }
36
37 // Set a new Time value using military time. Perform validity
38 // checks on the data values. Set invalid values to zero.
39 void Time::setTime( int h, int m, int s )
40 {
41    hour   = ( h >= 0 && h < 24 ) ? h : 0;
42    minute = ( m >= 0 && m < 60 ) ? m : 0;
43    second = ( s >= 0 && s < 60 ) ? s : 0;
44 } // end function setTime
45
```



time2.cpp (Part 1 of 2)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
46 // Print Time in military format
47 void Time::printMilitary()
48 {
49    cout << ( hour < 10 ? "0" : "" ) << hour << ":"
50         << ( minute < 10 ? "0" : "" ) << minute;
51 } // end function printMilitary
52
53 // Print Time in standard format
54 void Time::printStandard()
55 {
56    cout << ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 )
57         << ":" << ( minute < 10 ? "0" : "" ) << minute
58         << ":" << ( second < 10 ? "0" : "" ) << second
59         << ( hour < 12 ? " AM" : " PM" );
60 } // end function printStandard
```

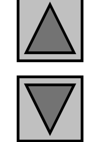


Outline

time2.cpp (Part 2 of 2)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
61 // Fig. 16.7: fig16_07.cpp
62 // Demonstrating a default constructor
63 // function for class Time.
64 #include <iostream>
65
66 using std::cout;
67 using std::endl;
68
69 #include "time2.h"
70
71 int main()
72 {
73    Time t1,             // all arguments defaulted
74         t2( 2 ),        // minute and second defaulted
75         t3( 21, 34 ),   // second defaulted
76         t4( 12, 25, 42 ), // all values specified
77         t5( 27, 74, 99 ); // all bad values specified
78
79    cout << "Constructed with:\n"
80         << "all arguments defaulted:\n   ";
81    t1.printMilitary();
82    cout << "\n   ";
83    t1.printStandard();
84
```

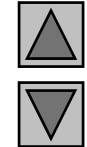


Outline

fig16_07.cpp (Part 1 of 2)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
85    cout << "\nhour specified; minute and second defaulted:"
86         << "\n   ";
87    t2.printMilitary();
88    cout << "\n   ";
89    t2.printStandard();
90
91    cout << "\nhour and minute specified; second defaulted:"
92         << "\n   ";
93    t3.printMilitary();
94    cout << "\n   ";
95    t3.printStandard();
96
97    cout << "\nhour, minute, and second specified:"
98         << "\n   ";
99    t4.printMilitary();
100   cout << "\n   ";
101   t4.printStandard();
102
103   cout << "\nall invalid values specified:"
104        << "\n   ";
105   t5.printMilitary();
106   cout << "\n   ";
107   t5.printStandard();
108   cout << endl;
109
110   return 0;
111 } // end function main
```



Outline

fig16_07.cpp (Part 2 of 2)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Outline

Program Output

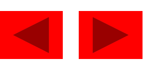
```
Constructed with:
all arguments defaulted:
   00:00
   12:00:00 AM
hour specified; minute and second defaulted:
   02:00
   2:00:00 AM
hour and minute specified; second defaulted:
   21:34
   9:34:00 PM
hour, minute, and second specified:
   12:25
   12:25:42 PM
all invalid values specified:
   00:00
   12:00:00 AM
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

16.9 Using Destructors

- Destructor
  - Member function of class
  - Performs termination housekeeping before the system reclaims the object's memory
  - Complement of the constructor
  - Name is *tilde* (~) followed by the class name
    - `~Time`
    - Recall that the constructor's name is the class name
  - Receives no parameters, returns no value
  - One destructor per class - no overloading allowed

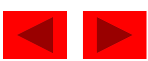
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



16.10 When Constructors and Destructors Are Called

- Constructors and destructors called automatically
  - Order depends on scope of objects
- Global scope objects
  - Constructors called before any other function (including `main`)
  - Destructors called when `main` terminates (or `exit` function called)
  - Destructors not called if program terminates with `abort`

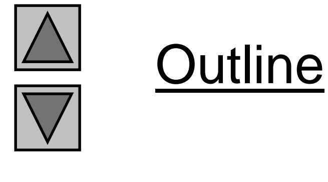
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



16.10 When Constructors and Destructors Are Called (II)

- Automatic local objects
  - Constructors called when objects defined
  - Destructors called when objects leave scope (when the block in which they are defined is exited)
  - Destructors not called if program ends with `exit` or `abort`
- `static` local objects
  - Constructors called when execution reaches the point where the objects are defined
  - Destructors called when `main` terminates or the `exit` function is called
  - Destructors not called if the program ends with `abort`

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

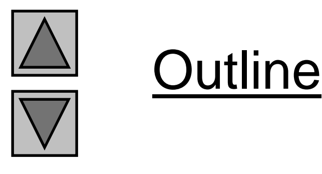


create.h

```
1  // Fig. 16.8: create.h
2  // Definition of class CreateAndDestroy.
3  // Member functions defined in create.cpp.
4  #ifndef CREATE_H
5  #define CREATE_H
6
7  class CreateAndDestroy {
8  public:
9     CreateAndDestroy( int );  // constructor
10    ~CreateAndDestroy();      // destructor
11 private:
12    int data;
13 }; // end class CreateAndDestroy
14
15 #endif
```

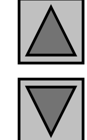
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
16 // Fig. 16.8: create.cpp
17 // Member function definitions for class CreateAndDestroy
18 #include <iostream>
19
20 using std::cout;
21 using std::endl;
22
23 #include "create.h"
24
25 CreateAndDestroy::CreateAndDestroy( int value )
26 {
27    data = value;
28    cout << "Object " << data << "   constructor";
29 } // end CreateAndDestroy constructor
30
31 CreateAndDestroy::~CreateAndDestroy()
32    { cout << "Object " << data << "   destructor " << endl; }
```



create.cpp

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Outline

fig16_08.cpp (Part 1 of 2)

```
33 // Fig. 16.8: fig16_08.cpp
34 // Demonstrating the order in which constructors and
35 // destructors are called.
36 #include <iostream>
37
38 using std::cout;
39 using std::endl;
40
41 #include "create.h"
42
43 void create( void );   // prototype
44
45 CreateAndDestroy first( 1 );  // global object
46
47 int main()
48 {
49    cout << "   (global created before main)" << endl;
50
51    CreateAndDestroy second( 2 );        // local object
52    cout << "   (local automatic in main)" << endl;
53
54    static CreateAndDestroy third( 3 );  // local object
55    cout << "   (local static in main)" << endl;
56
57    create();  // call function to create objects
58
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
59    CreateAndDestroy fourth( 4 );        // local object
60    cout << "   (local automatic in main)" << endl;
61    return 0;
62 } // end function main
63
64 // Function to create objects
65 void create( void )
66 {
67    CreateAndDestroy fifth( 5 );
68    cout << "   (local automatic in create)" << endl;
69
70    static CreateAndDestroy sixth( 6 );
71    cout << "   (local static in create)" << endl;
72
73    CreateAndDestroy seventh( 7 );
74    cout << "   (local automatic in create)" << endl;
75 } // end function create
```

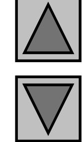
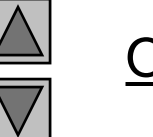


fig16_08.cpp (Part 2 of 2)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
Object 1   constructor   (global created before main)
Object 2   constructor   (local automatic in main)
Object 3   constructor   (local static in main)
Object 5   constructor   (local automatic in create)
Object 6   constructor   (local static in create)
Object 7   constructor   (local automatic in create)
Object 7   destructor
Object 5   destructor
Object 4   constructor   (local automatic in main)
Object 4   destructor
Object 2   destructor
Object 6   destructor
Object 3   destructor
Object 1   destructor
```



Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

16.11 Using Data Members and Member Functions

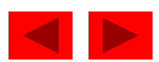
- Classes provide `public` member functions
  - Set (i.e., write) or *get* (i.e., read) values of `private` data members
  - Adjustment of bank balance (a `private` data member of class `BankAccount`) by member function `computeInterest`
- Naming
  - Member function that *set*s `interestRate` typically named `setInterestRate`
  - Member function that *get*s `interestRate` would typically be called `getInterestRate`

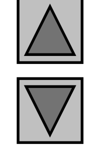
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

16.11 Using Data Members and Member Functions (II)

- Do *set* and *get* capabilities effectively make data members `public`?
  - No!
  - Programmer decides what the function can set and what information the function can get
- `public` set functions should
  - Check attempts to modify data members
  - Ensure that the new value is appropriate for that data item
  - Example: an attempt to *set* the day of the month to 37 would be rejected
  - Programmer must include these features

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.





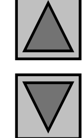
Outline

time3.h (Part 1 of 2)

```
1  // Fig. 16.9: time3.h
2  // Declaration of the Time class.
3  // Member functions defined in time3.cpp
4
5  // preprocessor directives that
6  // prevent multiple inclusions of header file
7  #ifndef TIME3_H
8  #define TIME3_H
9
10 class Time {
11 public:
12    Time( int = 0, int = 0, int = 0 );  // constructor
13
14    // set functions
15    void setTime( int, int, int );  // set hour, minute, second
16    void setHour( int );    // set hour
17    void setMinute( int ); // set minute
18    void setSecond( int ); // set second
19
20    // get functions
21    int getHour();          // return hour
22    int getMinute();        // return minute
23    int getSecond();        // return second
24
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
25    void printMilitary();  // output military time
26    void printStandard();  // output standard time
27
28 private:
29    int hour;              // 0 - 23
30    int minute;            // 0 - 59
31    int second;            // 0 - 59
32 }; // end class Time
33
34 #endif
```

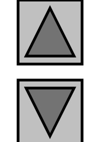


Outline

time3.h (Part 2 of 2)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
35 // Fig. 16.9: time3.cpp
36 // Member function definitions for Time class.
37 #include <iostream>
38
39 using std::cout;
40
41 #include "time3.h"
42
43 // Constructor function to initialize private data.
44 // Calls member function setTime to set variables.
45 // Default values are 0 (see class definition).
46 Time::Time( int hr, int min, int sec )
47    { setTime( hr, min, sec ); }
48
49 // Set the values of hour, minute, and second.
50 void Time::setTime( int h, int m, int s )
51 {
52    setHour( h );
53    setMinute( m );
54    setSecond( s );
55 } // end function setTime
56
57 // Set the hour value
58 void Time::setHour( int h )
59    { hour = ( h >= 0 && h < 24 ) ? h : 0; }
60
```

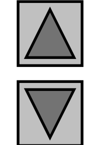


Outline

time3.cpp (Part 1 of 3)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
61 // Set the minute value
62 void Time::setMinute( int m )
63    { minute = ( m >= 0 && m < 60 ) ? m : 0; }
64
65 // Set the second value
66 void Time::setSecond( int s )
67    { second = ( s >= 0 && s < 60 ) ? s : 0; }
68
69 // Get the hour value
70 int Time::getHour() { return hour; }
71
72 // Get the minute value
73 int Time::getMinute() { return minute; }
74
75 // Get the second value
76 int Time::getSecond() { return second; }
77
78 // Print time is military format
79 void Time::printMilitary()
80 {
81    cout << ( hour < 10 ? "0" : "" ) << hour << ":"
82         << ( minute < 10 ? "0" : "" ) << minute;
83 } // end function printMilitary
84
```

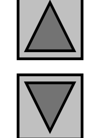


Outline

time3.cpp (Part 2 of 3)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

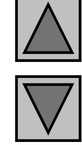
```
85 // Print time in standard format
86 void Time::printStandard()
87 {
88    cout << ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 )
89         << ":" << ( minute < 10 ? "0" : "" ) << minute
90         << ":" << ( second < 10 ? "0" : "" ) << second
91         << ( hour < 12 ? " AM" : " PM" );
92 } // end function printStandard
```



time3.cpp (Part 3 of 3)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
93 // Fig. 16.9: fig16_09.cpp
94 // Demonstrating the Time class set and get functions
95 #include <iostream>
96
97 using std::cout;
98 using std::endl;
99
100 #include "time3.h"
101
102 void incrementMinutes( Time &, const int );
103
104 int main()
105 {
106    Time t;
107
108    t.setHour( 17 );
109    t.setMinute( 34 );
110    t.setSecond( 25 );
111
```

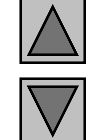


Outline

fig16_09.cpp (Part 1 of 3)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
112    cout << "Result of setting all valid values:\n"
113         << "  Hour: " << t.getHour()
114         << "  Minute: " << t.getMinute()
115         << "  Second: " << t.getSecond();
116
117    t.setHour( 234 );     // invalid hour set to 0
118    t.setMinute( 43 );
119    t.setSecond( 6373 ); // invalid second set to 0
120
121    cout << "\n\nResult of attempting to set invalid hour and"
122         << " second:\n  Hour: " << t.getHour()
123         << "  Minute: " << t.getMinute()
124         << "  Second: " << t.getSecond() << "\n\n";
125
126    t.setTime( 11, 58, 0 );
127    incrementMinutes( t, 3 );
128
129    return 0;
130 } // end function main
131
132 void incrementMinutes( Time &tt, const int count )
133 {
134    cout << "Incrementing minute " << count
135         << " times:\nStart time: ";
136    tt.printStandard();
137
```



Outline

fig16_09.cpp (Part 2 of 3)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
138    for ( int i = 0; i < count; i++ ) {
139       tt.setMinute( ( tt.getMinute() + 1 ) % 60 );
140
141       if ( tt.getMinute() == 0 )
142          tt.setHour( ( tt.getHour() + 1 ) % 24 );
143
144       cout << "\nminute + 1: ";
145       tt.printStandard();
146    } // end for
147
148    cout << endl;
149 } // end function incrementMinutes
```

```
Result of setting all valid values:
  Hour: 17  Minute: 34  Second: 25

Result of attempting to set invalid hour and second:
  Hour: 0  Minute: 43  Second: 0

Incrementing minute 3 times:
Start time: 11:58:00 AM
minute + 1: 11:59:00 AM
minute + 1: 12:00:00 PM
minute + 1: 12:01:00 PM
```

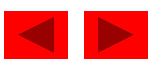

Program Output

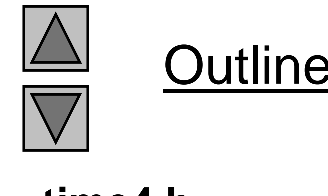
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

16.12 A Subtle Trap: Returning a Reference to a Private Data Member

- Reference to an object
  - Alias for the name of the object
  - May be used on the left side of an assignment statement
  - Reference can receive a value, which changes the original object as well
- One way to use this capability (unfortunately!)
  - Have a `public` member function of a class return a non-`const` reference to a `private` data member
  - This reference can be modified, which changes the original data

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



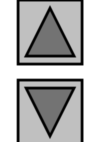


time4.h

```
1  // Fig. 16.10: time4.h
2  // Declaration of the Time class.
3  // Member functions defined in time4.cpp
4
5  // preprocessor directives that
6  // prevent multiple inclusions of header file
7  #ifndef TIME4_H
8  #define TIME4_H
9
10 class Time {
11 public:
12    Time( int = 0, int = 0, int = 0 );
13    void setTime( int, int, int );
14    int getHour();
15    int &badSetHour( int );  // DANGEROUS reference return
16 private:
17    int hour;
18    int minute;
19    int second;
20 }; // end class Time
21
22 #endif
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

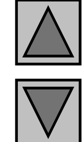
```
23 // Fig. 16.10: time4.cpp
24 // Member function definitions for Time class.
25 #include "time4.h"
26
27 // Constructor function to initialize private data.
28 // Calls member function setTime to set variables.
29 // Default values are 0 (see class definition).
30 Time::Time( int hr, int min, int sec )
31    { setTime( hr, min, sec ); }
32
33 // Set the values of hour, minute, and second.
34 void Time::setTime( int h, int m, int s )
35 {
36    hour   = ( h >= 0 && h < 24 ) ? h : 0;
37    minute = ( m >= 0 && m < 60 ) ? m : 0;
38    second = ( s >= 0 && s < 60 ) ? s : 0;
39 } // end function setTime
40
41 // Get the hour value
42 int Time::getHour() { return hour; }
43
```



time4.cpp (Part 1 of 2)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
44 // POOR PROGRAMMING PRACTICE:
45 // Returning a reference to a private data member.
46 int &Time::badSetHour( int hh )
47 {
48    hour = ( hh >= 0 && hh < 24 ) ? hh : 0;
49
50    return hour;  // DANGEROUS reference return
51 } // end function badSetHour
```

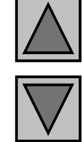


Outline

time4.cpp (Part 2 of 2)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
52 // Fig. 16.10: fig16_10.cpp
53 // Demonstrating a public member function that
54 // returns a reference to a private data member.
55 // Time class has been trimmed for this example.
56 #include <iostream>
57
58 using std::cout;
59 using std::endl;
60
61 #include "time4.h"
62
63 int main()
64 {
65    Time t;
66    int &hourRef = t.badSetHour( 20 );
67
68    cout << "Hour before modification: " << hourRef;
69    hourRef = 30;  // modification with invalid value
70    cout << "\nHour after modification: " << t.getHour();
71
```



Outline

fig16_10.cpp (Part 1 of 2)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
72     // Dangerous: Function call that returns
73     // a reference can be used as an lvalue!
74     t.badSetHour( 12 ) = 74;
75     cout << "\n\n*********************************\n"
76          << "POOR PROGRAMMING PRACTICE!!!!!!!!\n"
77          << "badSetHour as an lvalue, Hour: "
78          << t.getHour()
79          << "\n*********************************" << endl;
80
81     return 0;
82  }
```

Outline

fig16_10.cpp (Part 2 of 2)

```
Hour before modification: 20
Hour after modification: 30

*********************************
POOR PROGRAMMING PRACTICE!!!!!!!!
badSetHour as an lvalue, Hour: 74
*********************************
```

Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

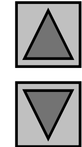
16.13 Assignment by Default Memberwise Copy

- Assignment operator (=)
  - Sets variables equal, i.e., `x = y;`
  - Can be used to assign an object to another object of the same type
  - Memberwise copy — member by member copy

    `myObject1 = myObject2;`

- Objects may be
  - Passed as function arguments
  - Returned from functions (call-by-value default)
    - Use pointers for call by reference

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

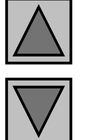


Outline

fig16_11.cpp (Part 1 of 2)

```
1  // Fig. 16.11: fig16_11.cpp
2  // Demonstrating that class objects can be assigned
3  // to each other using default memberwise copy
4  #include <iostream>
5
6  using std::cout;
7  using std::endl;
8
9  // Simple Date class
10 class Date {
11 public:
12    Date( int = 1, int = 1, int = 1990 ); // default constructor
13    void print();
14 private:
15    int month;
16    int day;
17    int year;
18 }; // end class Date
19
20 // Simple Date constructor with no range checking
21 Date::Date( int m, int d, int y )
22 {
23    month = m;
24    day = d;
25    year = y;
26 } // end Date constructor
27
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Outline

fig16_11.cpp (Part 2 of 2)

```
28 // Print the Date in the form mm-dd-yyyy
29 void Date::print()
30    { cout << month << '-' << day << '-' << year; }
31
32 int main()
33 {
34    Date date1( 7, 4, 1993 ), date2;  // d2 defaults to 1/1/90
35
36    cout << "date1 = ";
37    date1.print();
38    cout << "\ndate2 = ";
39    date2.print();
40
41    date2 = date1;   // assignment by default memberwise copy
42    cout << "\n\nAfter default memberwise copy, date2 = ";
43    date2.print();
44    cout << endl;
45
46    return 0;
47 } // end function main
```

Program Output

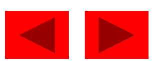
```
date1 = 7-4-1993
date2 = 1-1-1990

After default memberwise copy, date2 = 7-4-1993
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

16.14 Software Reusability

- Object-oriented programmers
  - Concentrate on implementing useful classes
- Tremendous opportunity to capture and catalog classes
  - Accessed by large segments of the programming community
  - Class libraries exist for this purpose
- Software
  - Constructed from existing, well-defined, carefully tested, portable, widely available components
  - Speeds development of powerful, high-quality software

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

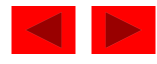


Chapter 17 - C++ Classes: Part II

Outline

17.1 Introduction
17.2 `const` **(Constant) Objects and** `const` **Member Functions**
17.3 Composition: Objects as Members of Classes
17.4 `friend` **Functions and** `friend` **Classes**
17.5 Using the `this` **Pointer**
17.6 Dynamic Memory Allocation with Operators `new` **and** `delete`
17.7 `static` **Class Members**
17.8 Data Abstraction and Information Hiding
17.8.1 Example: Array Abstract Data Type
17.8.2 Example: String Abstract Data Type
17.8.3 Example: Queue Abstract Data Type
17.9 Container Classes and Iterators

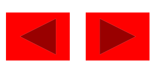
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Objectives

- In this chapter, you will learn:
  - To be able to create and destroy objects dynamically.
  - To be able to specify `const` (constant) objects and `const` member functions.
  - To understand the purpose of `friend` functions and `friend` classes.
  - To understand how to use `static` data members and member functions.
  - To understand the concept of a container class.
  - To understand the notion of iterator classes that walk through the elements of container classes.
  - To understand the use of the `this` pointer.

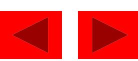
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



17.1 Introduction

- Chapters 16-18
  - Object-based programming
- Chapter 19-20
  - Polymorphism and inheritance
  - Object-oriented programming

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



17.2 const (Constant) Objects and const Member Functions

- Principle of least privilege
  - Only give objects permissions they need, no more
- Keyword `const`
  - Specify that an object is not modifiable
  - Any attempt to modify the object is a syntax error
  - For example:
    ```
    const time noon( 12, 0, 0 );
    ```
  - Defines a `const` object `noon` of class `time` and initializes it to 12 noon

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

17.2 const (Constant) Objects and const Member Functions (II)

- const objects require const functions
  - Functions declared const cannot modify the object
  - const specified in function prototype and definition

Prototype: *ReturnType FunctionName(param1,param2…)* const;

Definition: *ReturnType FunctionName(param1,param2…)* const { …};

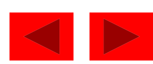
Example:

```
int A::getValue() const
   {return privateDataMember};
```

  - Returns the value of a data member, and is appropriately declared const

- Constructors / Destructors cannot be const
  - They need to initialize variables (therefore modifying them)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



```
1  // Fig. 17.1: time5.h
2  // Declaration of the class Time.
3  // Member functions defined in time5.cpp
4  #ifndef TIME5_H
5  #define TIME5_H
6
7  class Time {
8  public:
9     Time( int = 0, int = 0, int = 0 );  // default constructor
10
11    // set functions
12    void setTime( int, int, int );  // set time
13    void setHour( int );      // set hour
14    void setMinute( int );    // set minute
15    void setSecond( int );    // set second
16
17    // get functions (normally declared const)
18    int getHour() const;      // return hour
19    int getMinute() const;    // return minute
20    int getSecond() const;    // return second
21
22    // print functions (normally declared const)
23    void printMilitary() const;  // print military time
24    void printStandard();        // print standard time
25 private:
```

Outline

times.h (Part 1 of 2)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
26    int hour;               // 0 - 23
27    int minute;             // 0 - 59
28    int second;             // 0 - 59
29 }; // end class Time
30
31 #endif
```

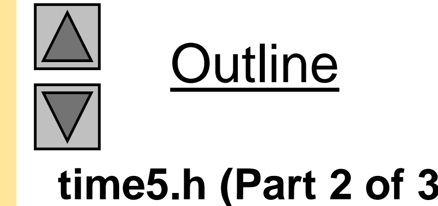
```
32 // Fig. 17.1: time5.cpp
33 // Member function definitions for Time class.
34 #include <iostream>
35
36 using std::cout;
37
38 #include "time5.h"
39
40 // Constructor function to initialize private data.
41 // Default values are 0 (see class definition).
42 Time::Time( int hr, int min, int sec )
43    { setTime( hr, min, sec ); }
44
45 // Set the values of hour, minute, and second.
46 void Time::setTime( int h, int m, int s )
47 {
48    setHour( h );
49    setMinute( m );
50    setSecond( s );
51 } // end function setTime
52
```


time.h (Part 2 of 2)

time5.h (Part 1 of 3)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

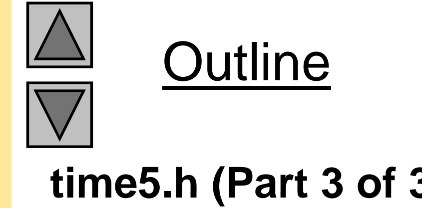
```
53 // Set the hour value
54 void Time::setHour( int h )
55    { hour = ( h >= 0 && h < 24 ) ? h : 0; }
56
57 // Set the minute value
58 void Time::setMinute( int m )
59    { minute = ( m >= 0 && m < 60 ) ? m : 0; }
60
61 // Set the second value
62 void Time::setSecond( int s )
63    { second = ( s >= 0 && s < 60 ) ? s : 0; }
64
65 // Get the hour value
66 int Time::getHour() const { return hour; }
67
68 // Get the minute value
69 int Time::getMinute() const { return minute; }
70
71 // Get the second value
72 int Time::getSecond() const { return second; }
73
```



time5.h (Part 2 of 3)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
74 // Display military format time: HH:MM
75 void Time::printMilitary() const
76 {
77    cout << ( hour < 10 ? "0" : "" ) << hour << ":"
78         << ( minute < 10 ? "0" : "" ) << minute;
79 } // end function printMilitary
80
81 // Display standard format time: HH:MM:SS AM (or PM)
82 void Time::printStandard()  // should be const
83 {
84    cout << ( ( hour == 12 ) ? 12 : hour % 12 ) << ":"
85         << ( minute < 10 ? "0" : "" ) << minute << ":"
86         << ( second < 10 ? "0" : "" ) << second
87         << ( hour < 12 ? " AM" : " PM" );
88 } // end function printStandard
```



time5.h (Part 3 of 3)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

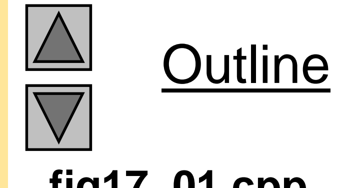
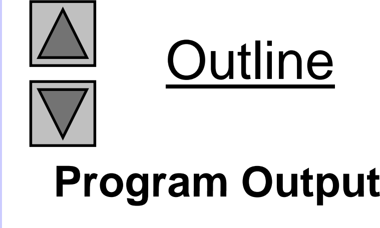


fig17_01.cpp

```
89 // Fig. 17.1: fig17_01.cpp
90 // Attempting to access a const object with
91 // non-const member functions.
92 #include "time5.h"
93
94 int main()
95 {
96    Time wakeUp( 6, 45, 0 );       // non-constant object
97    const Time noon( 12, 0, 0 );   // constant object
98
99                             // MEMBER FUNCTION   OBJECT
100    wakeUp.setHour( 18 );   // non-const         non-const
101
102    noon.setHour( 12 );     // non-const         const
103
104    wakeUp.getHour();       // const             non-const
105
106    noon.getMinute();       // const             const
107    noon.printMilitary();   // const             const
108    noon.printStandard();   // non-const         const
109    return 0;
110 } // end function main
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
Compiling...
Fig17_01.cpp
d:fig17_01.cpp(14) : error C2662: 'setHour' : cannot convert 'this'
pointer from 'const class Time' to 'class Time &'
Conversion loses qualifiers
d:\fig17_01.cpp(20) : error C2662: 'printStandard' : cannot convert
'this' pointer from 'const class Time' to 'class Time &'
Conversion loses qualifiers
Time5.cpp
Error executing cl.exe.

test.exe - 2 error(s), 0 warning(s)
```



Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

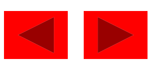
17.2 `const` (Constant) Objects and `const` Member Functions (III)

- Member initializer syntax
  - Data member `increment` in class `Increment`.
  - Constructor for `Increment` is modified as follows:

```
Increment::Increment( int c, int i )
      : increment( i )
        { count = c; }
```

  - `": increment( i )"` initializes `increment` to the value of `i`.
  - Any data member can be initialized using member initializer syntax
  - `const`s and references must be initialized this way
- Multiple member initializers
  - Use comma-separated list after the colon

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Outline

fig17_02.cpp (Part 1 of 2)

```
1  // Fig. 17.2: fig17_02.cpp
2  // Using a member initializer to initialize a
3  // constant of a built-in data type.
4  #include <iostream>
5
6  using std::cout;
7  using std::endl;
8
9  class Increment {
10 public:
11    Increment( int c = 0, int i = 1 );
12    void addIncrement() { count += increment; }
13    void print() const;
14
15 private:
16    int count;
17    const int increment;    // const data member
18 }; // end class Increment
19
20 // Constructor for class Increment
21 Increment::Increment( int c, int i )
22    : increment( i )    // initializer for const member
23 { count = c; }
24
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Outline

fig17_02.cpp (Part 2 of 2)

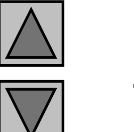
```
25 // Print the data
26 void Increment::print() const
27 {
28    cout << "count = " << count
29         << ", increment = " << increment << endl;
30 } // end function print
31
32 int main()
33 {
34    Increment value( 10, 5 );
35
36    cout << "Before incrementing: ";
37    value.print();
38
39    for ( int j = 0; j < 3; j++ ) {
40       value.addIncrement();
41       cout << "After increment " << j + 1 << ": ";
42       value.print();
43    } // end for
44
45    return 0;
46 } // end function main
```

Program Output

```
Before incrementing: count = 10, increment = 5
After increment 1: count = 15, increment = 5
After increment 2: count = 20, increment = 5
After increment 3: count = 25, increment = 5
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
1  // Fig. 17.3: fig17_03.cpp
2  // Attempting to initialize a constant of
3  // a built-in data type with an assignment.
4  #include <iostream>
5
6  using std::cout;
7  using std::endl;
8
9  class Increment {
10 public:
11    Increment( int c = 0, int i = 1 );
12    void addIncrement() { count += increment; }
13    void print() const;
14 private:
15    int count;
16    const int increment;
17 }; // end class Increment
18
19 // Constructor for class Increment
20 Increment::Increment( int c, int i )
21 {             // Constant member 'increment' is not initialized
22    count = c;
23    increment = i;  // ERROR: Cannot modify a const object
24 } // end Increment constructor
25
```

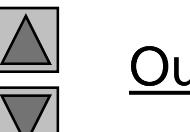


Outline

fig17_03.cpp (Part 1 of 2)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
26 // Print the data
27 void Increment::print() const
28 {
29    cout << "count = " << count
30         << ", increment = " << increment << endl;
31 } // end function print
32
33 int main()
34 {
35    Increment value( 10, 5 );
36
37    cout << "Before incrementing: ";
38    value.print();
39
40    for ( int j = 0; j < 3; j++ ) {
41       value.addIncrement();
42       cout << "After increment " << j << ": ";
43       value.print();
44    } // end for
45
46    return 0;
47 } // end function main
```

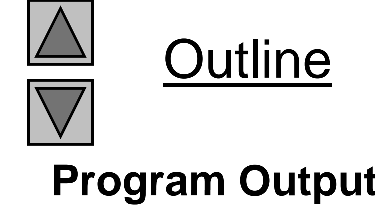


Outline

fig17_03.cpp (Part 1 of 2)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
Compiling...
Fig17_03.cpp
D:\Fig17_03.cpp(21) : error C2758: 'increment' : must be initialized in
constructor base/member initializer list
D:\Fig17_03.cpp(16) : see declaration of 'increment'
D:\Fig17_03.cpp(23) : error C2166: l-value specifies const object
Error executing cl.exe.

test.exe - 2 error(s), 0 warning(s)
```



Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

17.3 Composition: Objects as Members of Classes

- Composition
  - Class has objects of other classes as members
- Construction of objects
  - Member objects constructed in order declared
    - Not in order of constructor's member initializer list
  - Constructed before their enclosing class objects (host objects)
  - Constructors called inside out
  - Destructors called outside in

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

17.3 Composition: Objects as Members of Classes (II)

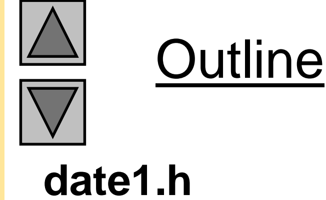
- Example:

```
Employee::Employee( char *fname, char *lname,
                    int bmonth, int bday, int byear,
                    int hmonth, int hday, int hyear )
   : birthDate( bmonth, bday, byear ),
     hireDate( hmonth, hday, hyear )
```

  - Insert objects from `Date` class (`birthDate and hireDate`) into `Employee` class
  - `birthDate` and `hireDate` have member initializers - they are probably `const`s in the `Employee` class

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
1 // Fig. 17.4: date1.h
2 // Declaration of the Date class.
3 // Member functions defined in date1.cpp
4 #ifndef DATE1_H
5 #define DATE1_H
6
7 class Date {
8 public:
9    Date( int = 1, int = 1, int = 1900 ); // default constructor
10    void print() const;  // print date in month/day/year format
11    ~Date();  // provided to confirm destruction order
12 private:
13    int month;  // 1-12
14    int day;    // 1-31 based on month
15    int year;   // any year
16
17    // utility function to test proper day for month and year
18    int checkDay( int );
19 }; // end class Date
20
21 #endif
```



date1.h

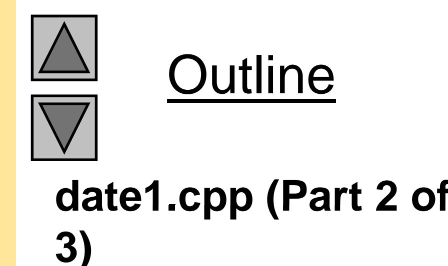
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
22 // Fig. 17.4: date1.cpp
23 // Member function definitions for Date class.
24 #include <iostream>
25
26 using std::cout;
27 using std::endl;
28
29 #include "date1.h"
30
31 // Constructor: Confirm proper value for month;
32 // call utility function checkDay to confirm proper
33 // value for day.
34 Date::Date( int mn, int dy, int yr )
35 {
36    if ( mn > 0 && mn <= 12 )       // validate the month
37       month = mn;
38    else {
39       month = 1;
40       cout << "Month " << mn << " invalid. Set to month 1.\n";
41    } // end else
42
43    year = yr;                      // should validate yr
44    day = checkDay( dy );           // validate the day
45
```

Outline

date1.cpp (Part 1 of 3)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



date1.cpp (Part 2 of 3)

```
46    cout << "Date object constructor for date ";
47    print();         // interesting: a print with no arguments
48    cout << endl;
49 } // end Date constructor
50
51 // Print Date object in form  month/day/year
52 void Date::print() const
53    { cout << month << '/' << day << '/' << year; }
54
55 // Destructor: provided to confirm destruction order
56 Date::~Date()
57 {
58    cout << "Date object destructor for date ";
59    print();
60    cout << endl;
61 } // end Date destructor
62
63 // Utility function to confirm proper day value
64 // based on month and year.
65 // Is the year 2000 a leap year?
66 int Date::checkDay( int testDay )
67 {
68    static const int daysPerMonth[ 13 ] =
69       {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
70
```

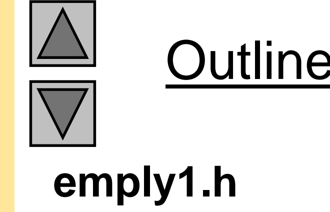
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
71     if ( testDay > 0 && testDay <= daysPerMonth[ month ] )
72        return testDay;
73
74     if ( month == 2 &&       // February: Check for leap year
75          testDay == 29 &&
76          ( year % 400 == 0 ||
77           ( year % 4 == 0 && year % 100 != 0 ) ) )
78        return testDay;
79
80     cout << "Day " << testDay << " invalid. Set to day 1.\n";
81
82     return 1;  // leave object in consistent state if bad value
83  } // end function checkDay
```

date1.cpp (Part 3 of 3)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

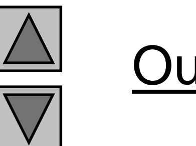
```
84 // Fig. 17.4: emply1.h
85 // Declaration of the Employee class.
86 // Member functions defined in emply1.cpp
87 #ifndef EMPLY1_H
88 #define EMPLY1_H
89
90 #include "date1.h"
91
92 class Employee {
93 public:
94    Employee( char *, char *, int, int, int, int, int, int );
95    void print() const;
96    ~Employee();  // provided to confirm destruction order
97 private:
98    char firstName[ 25 ];
99    char lastName[ 25 ];
100    const Date birthDate;
101    const Date hireDate;
102 }; // end Employee constructor
103
104 #endif
```



emply1.h

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

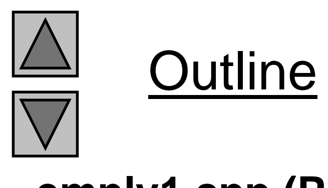
```
105 // Fig. 17.4: emply1.cpp
106 // Member function definitions for Employee class.
107 #include <iostream>
108
109 using std::cout;
110 using std::endl;
111
112 #include <cstring>
113 #include "emply1.h"
114 #include "date1.h"
115
116 Employee::Employee( char *fname, char *lname,
117                     int bmonth, int bday, int byear,
118                     int hmonth, int hday, int hyear )
119    : birthDate( bmonth, bday, byear ),
120      hireDate( hmonth, hday, hyear )
121 {
122    // copy fname into firstName and be sure that it fits
123    int length = strlen( fname );
124    length = ( length < 25 ? length : 24 );
125    strncpy( firstName, fname, length );
126    firstName[ length ] = '\0';
127
```



Outline

emply1.cpp (Part 1 of 2)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



emply1.cpp (Part 2 of 2)

```
128    // copy lname into lastName and be sure that it fits
129    length = strlen( lname );
130    length = ( length < 25 ? length : 24 );
131    strncpy( lastName, lname, length );
132    lastName[ length ] = '\0';
133
134    cout << "Employee object constructor: "
135         << firstName << ' ' << lastName << endl;
136 } // end Employee constructor
137
138 void Employee::print() const
139 {
140    cout << lastName << ", " << firstName << "\nHired: ";
141    hireDate.print();
142    cout << "  Birth date: ";
143    birthDate.print();
144    cout << endl;
145 } // end function print
146
147 // Destructor: provided to confirm destruction order
148 Employee::~Employee()
149 {
150    cout << "Employee object destructor: "
151         << lastName << ", " << firstName << endl;
152 } // end Employee destructor
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

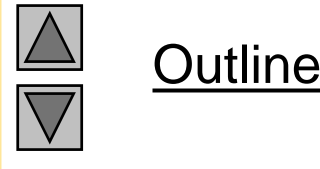


fig17_04.cpp

```
153 // Fig. 17.4: fig17_04.cpp
154 // Demonstrating composition: an object with member objects.
155 #include <iostream>
156
157 using std::cout;
158 using std::endl;
159
160 #include "emply1.h"
161
162 int main()
163 {
164    Employee e( "Bob", "Jones", 7, 24, 1949, 3, 12, 1988 );
165
166    cout << '\n';
167    e.print();
168
169    cout << "\nTest Date constructor with invalid values:\n";
170    Date d( 14, 35, 1994 );  // invalid Date values
171    cout << endl;
172    return 0;
173 } // end function main
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Program Output

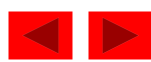
```
Date object constructor for date 7/24/1949
Date object constructor for date 3/12/1988
Employee object constructor: Bob Jones

Jones, Bob
Hired: 3/12/1988  Birth date: 7/24/1949

Test Date constructor with invalid values:
Month 14 invalid. Set to month 1.
Day 35 invalid. Set to day 1.
Date object constructor for date 1/1/1994

Date object destructor for date 1/1/1994
Employee object destructor: Jones, Bob
Date object destructor for date 3/12/1988
Date object destructor for date 7/24/1949
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

17.4 `friend` Functions and `friend` Classes

- `friend` function and `friend` classes
  - Can access `private` and `protected` (more later) members of another class
  - `friend` functions are not member functions of class
    - Defined outside of class scope
- Properties
  - Friendship is granted, not taken
  - NOT symmetric (if B a `friend` of A, A not necessarily a `friend` of B)
  - NOT transitive (if A a `friend` of B, B a `friend` of C, A not necessarily a `friend` of C)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



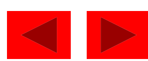
17.4 `friend` Functions and `friend` Classes (II)

- `friend` declarations
  - `friend` function
    - Keyword `friend` before function prototype in class that is giving friendship.
    - `friend int myFunction( int x );`
    - Appears in the class granting friendship
  - `friend` class
    - Type `friend class` *Classname* in class granting friendship
    - If `ClassOne` granting friendship to `ClassTwo`,

```
friend class ClassTwo;
```

appears in `ClassOne`'s definition

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Outline

fig17_05.cpp (Part 1 of 2)

```
1  // Fig. 17.5: fig17_05.cpp
2  // Friends can access private members of a class.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  // Modified Count class
9  class Count {
10    friend void setX( Count &, int ); // friend declaration
11  public:
12    Count() { x = 0; }                // constructor
13    void print() const { cout << x << endl; }  // output
14  private:
15    int x;  // data member
16  }; // end class Count
17
18  // Can modify private data of Count because
19  // setX is declared as a friend function of Count
20  void setX( Count &c, int val )
21  {
22    c.x = val;  // legal: setX is a friend of Count
23  } // end function setX
24
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

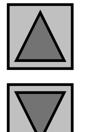
```
25 int main()
26 {
27    Count counter;
28
29    cout << "counter.x after instantiation: ";
30    counter.print();
31    cout << "counter.x after call to setX friend function: ";
32    setX( counter, 8 );  // set x with a friend
33    counter.print();
34    return 0;
35 } // end function main
```

```
counter.x after instantiation: 0
counter.x after call to setX friend function: 8
```


fig17_05.cpp (Part 1 of 2)

Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Outline

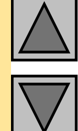
fig17_06.cpp (Part 1 of 2)

```
1  // Fig. 17.6: fig17_06.cpp
2  // Non-friend/non-member functions cannot access
3  // private data of a class.
4  #include <iostream>
5
6  using std::cout;
7  using std::endl;
8
9  // Modified Count class
10 class Count {
11 public:
12    Count() { x = 0; }                        // constructor
13    void print() const { cout << x << endl; }  // output
14 private:
15    int x;  // data member
16 }; // end class Count
17
18 // Function tries to modify private data of Count,
19 // but cannot because it is not a friend of Count.
20 void cannotSetX( Count &c, int val )
21 {
22    c.x = val;  // ERROR: 'Count::x' is not accessible
23 } // end function cannotSetX
24
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
25 int main()
26 {
27    Count counter;
28
29    cannotSetX( counter, 3 ); // cannotSetX is not a friend
30    return 0;
31 } // end function main
```

```
Compiling...
Fig17_06.cpp
D:\Fig17_06.cpp(22) :
   error C2248: 'x' : cannot access private member declared in
   class 'Count'
        D:\Fig17_06.cpp(15) : see declaration of 'x'
        Error executing cl.exe.

test.exe - 1 error(s), 0 warning(s)
```



Outline

fig17_06.cpp (Part 2 of 2)

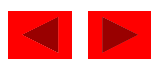
Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

17.5 Using the `this` Pointer

- `this` pointer
  - Allows objects to access their own address
  - Not part of the object itself
  - Implicit first argument on non-`static` member function call to the object
  - Implicitly reference member data and functions
- Example: class `Employee`
  - For non-`const` member functions: type `Employee * const`
    - Constant pointer to an `Employee` object
  - For `const` member functions: type `const Employee * const`
    - Constant pointer to an constant `Employee` object

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



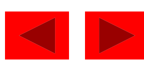
17.5 Using the `this` Pointer (II)

- Cascaded member function calls
  - Function returns a reference pointer to the same object

```
{return *this;}
```

  - Other functions can operate on that pointer
  - Functions that do not return references must be called last

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



17.5 Using the `this` Pointer (III)

- Example
  - Member functions `setHour`, `setMinute`, and `setSecond` all return `*this` (reference to an object)
  - For object `t`, consider

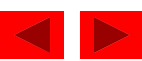
    `t.setHour(1).setMinute(2).setSecond(3);`
  - Executes `t.setHour(1)` and returns `*this` (reference to object), and expression becomes

    `t.setMinute(2).setSecond(3);`
  - Executes `t.setMinute(2)`, returns reference, and becomes

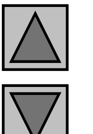
    `t.setSecond(3);`
  - Executes `t.setSecond(3)`, returns reference, and becomes

    `t;`
  - Has no effect

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



```
1  // Fig. 17.7: fig17_07.cpp
2  // Using the this pointer to refer to object members.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  class Test {
9  public:
10    Test( int = 0 );            // default constructor
11    void print() const;
12 private:
13    int x;
14 }; // end class Test
15
16 Test::Test( int a ) { x = a; }  // constructor
17
18 void Test::print() const   // ( ) around *this required
19 {
20    cout << "        x = " << x
21         << "\n  this->x = " << this->x
22         << "\n(*this).x = " << ( *this ).x << endl;
23 } // end function print
24
```



Outline

fig17_07.cpp (Part 1 of 2)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

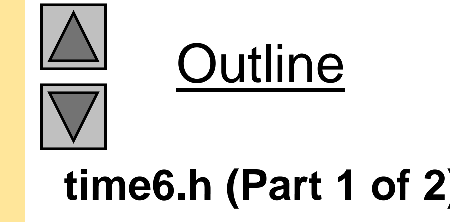
```
25 int main()
26 {
27    Test testObject( 12 );
28
29    testObject.print();
30
31    return 0;
32 } // end function main
```

```
        x = 12
  this->x = 12
(*this).x = 12
```

Outline

fig17_07.cpp (Part 2 of 2)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



time6.h (Part 1 of 2)

```
1  // Fig. 17.8: time6.h
2  // Cascading member function calls.
3
4  // Declaration of class Time.
5  // Member functions defined in time6.cpp
6  #ifndef TIME6_H
7  #define TIME6_H
8
9  class Time {
10 public:
11    Time( int = 0, int = 0, int = 0 );  // default constructor
12
13    // set functions
14    Time &setTime( int, int, int ); // set hour, minute, second
15    Time &setHour( int );    // set hour
16    Time &setMinute( int );  // set minute
17    Time &setSecond( int );  // set second
18
19    // get functions (normally declared const)
20    int getHour() const;     // return hour
21    int getMinute() const;   // return minute
22    int getSecond() const;   // return second
23
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
24    // print functions (normally declared const)
25    void printMilitary() const;  // print military time
26    void printStandard() const;  // print standard time
27 private:
28    int hour;            // 0 - 23
29    int minute;          // 0 - 59
30    int second;          // 0 - 59
31 }; // end class Time
32
33 #endif
```

```
34 // Fig. 17.8: time6.cpp
35 // Member function definitions for Time class.
36 #include <iostream>
37
38 using std::cout;
39
40 #include "time6.h"
41
42 // Constructor function to initialize private data.
43 // Calls member function setTime to set variables.
44 // Default values are 0 (see class definition).
45 Time::Time( int hr, int min, int sec )
46    { setTime( hr, min, sec ); }
47
```

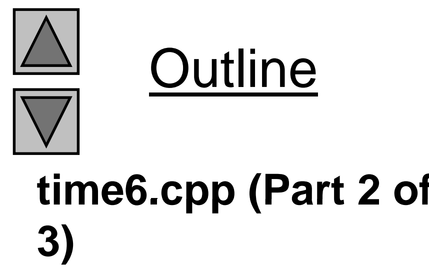
Outline

time6.h (Part 2 of 2)

time6.cpp (Part 1 of 3)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

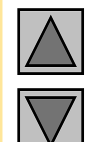
```
48 // Set the values of hour, minute, and second.
49 Time &Time::setTime( int h, int m, int s )
50 {
51    setHour( h );
52    setMinute( m );
53    setSecond( s );
54    return *this;    // enables cascading
55 } // end function setTime
56
57 // Set the hour value
58 Time &Time::setHour( int h )
59 {
60    hour = ( h >= 0 && h < 24 ) ? h : 0;
61
62    return *this;    // enables cascading
63 } // end function setHour
64
65 // Set the minute value
66 Time &Time::setMinute( int m )
67 {
68    minute = ( m >= 0 && m < 60 ) ? m : 0;
69
70    return *this;    // enables cascading
71 } // end function setMinute
72
```



time6.cpp (Part 2 of 3)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
78    return *this;   // enables cascading
79 } // end function setSecond
80
81 // Get the hour value
82 int Time::getHour() const { return hour; }
83
84 // Get the minute value
85 int Time::getMinute() const { return minute; }
86
87 // Get the second value
88 int Time::getSecond() const { return second; }
89
90 // Display military format time: HH:MM
91 void Time::printMilitary() const
92 {
93    cout << ( hour < 10 ? "0" : "" ) << hour << ":"
94         << ( minute < 10 ? "0" : "" ) << minute;
95 } // end function printMilitary
96
97 // Display standard format time: HH:MM:SS AM (or PM)
98 void Time::printStandard() const
99 {
100    cout << ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 )
101         << ":" << ( minute < 10 ? "0" : "" ) << minute
102         << ":" << ( second < 10 ? "0" : "" ) << second
103         << ( hour < 12 ? " AM" : " PM" );
104 } // end function printStandard
```



Outline

time6.cpp (Part 3 of 3)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

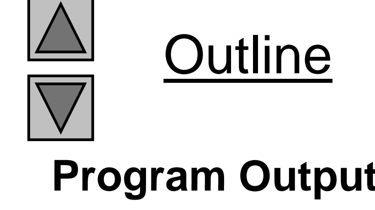
Outline

fig17_08.cpp

```
105 // Fig. 17.8: fig17_08.cpp
106 // Cascading member function calls together
107 // with the this pointer
108 #include <iostream>
109
110 using std::cout;
111 using std::endl;
112
113 #include "time6.h"
114
115 int main()
116 {
117    Time t;
118
119    t.setHour( 18 ).setMinute( 30 ).setSecond( 22 );
120    cout << "Military time: ";
121    t.printMilitary();
122    cout << "\nStandard time: ";
123    t.printStandard();
124
125    cout << "\n\nNew standard time: ";
126    t.setTime( 20, 20, 20 ).printStandard();
127    cout << endl;
128
129    return 0;
130 } // end function main
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
Military time: 18:30
Standard time: 6:30:22 PM

New standard time: 8:20:20 PM
```



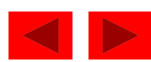
Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

17.6 Dynamic Memory Allocation with Operators `new` and `delete`

- `new` and `delete`
  - Better dynamic memory allocation than C's `malloc` and `free`
  - `new` - automatically creates object of proper size, calls constructor, returns pointer of the correct type
  - `delete` - destroys object and frees space
- Example:
  - `TypeName *typeNamePtr;`
    - Creates pointer to a `TypeName` object
  - `typeNamePtr = new TypeName;`
    - `new` creates `TypeName` object, returns pointer (which `typeNamePtr` is set equal to)
  - `delete typeNamePtr;`
    - Calls destructor for `TypeName` object and frees memory

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



17.6 Dynamic Memory Allocation with Operators new and delete (II)

- Initializing objects

```
double *thingPtr = new double( 3.14159 );
```

  - Initializes object of type `double` to `3.14159`

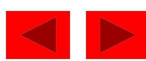
```
int *arrayPtr = new int[ 10 ];
```

  - Create ten element `int` array, assign to `arrayPtr`.
  - Use

    ```
    delete [] arrayPtr;
    ```

    to `delete` arrays

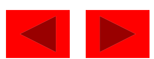
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



17.7 `static` Class Members

- `static` class members
  - Shared by all objects of a class
    - Normally, each object gets its own copy of each variable
  - Efficient when a single copy of data is enough
    - Only the `static` variable has to be updated
  - May seem like global variables, but have *class scope*
    - Only accessible to objects of same class
  - Initialized at file scope
  - Exist even if no instances (objects) of the class exist
  - Can be variables or functions
    - `public`, `private`, or `protected`

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



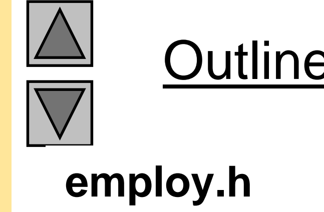
17.7 static Class Members (II)

- Accessing `static` members
  - `public static` variables: accessible through any object of the class
    - Or use class name and (`::`)

      `Employee::count`
  - `private static` variables: a `public static` member function must be used.
    - Prefix with class name and (`::`)

      `Employee::getCount()`
  - `static` member functions cannot access non-`static` data or functions
    - No `this` pointer, function exists independent of objects

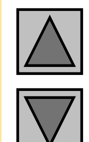
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



employ.h

```
1  // Fig. 17.9: employ1.h
2  // An employee class
3  #ifndef EMPLOY1_H
4  #define EMPLOY1_H
5
6  class Employee {
7  public:
8     Employee( const char*, const char* );  // constructor
9     ~Employee();                           // destructor
10    const char *getFirstName() const;  // return first name
11    const char *getLastName() const;   // return last name
12
13    // static member function
14    static int getCount();  // return # objects instantiated
15
16 private:
17    char *firstName;
18    char *lastName;
19
20    // static data member
21    static int count;  // number of objects instantiated
22 }; // end class Employee
23
24 #endif
```

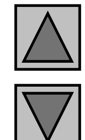
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



```
25 // Fig. 17.9: employ1.cpp
26 // Member function definitions for class Employee
27 #include <iostream>
28
29 using std::cout;
30 using std::endl;
31
32 #include <cstring>
33 #include <cassert>
34 #include "employ1.h"
35
36 // Initialize the static data member
37 int Employee::count = 0;
38
39 // Define the static member function that
40 // returns the number of employee objects instantiated.
41 int Employee::getCount() { return count; }
42
43 // Constructor dynamically allocates space for the
44 // first and last name and uses strcpy to copy
45 // the first and last names into the object
46 Employee::Employee( const char *first, const char *last )
47 {
48    firstName = new char[ strlen( first ) + 1 ];
49    assert( firstName != 0 );    // ensure memory allocated
50    strcpy( firstName, first );
51
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
52    lastName = new char[ strlen( last ) + 1 ];
53    assert( lastName != 0 );    // ensure memory allocated
54    strcpy( lastName, last );
55
56    ++count;  // increment static count of employees
57    cout << "Employee constructor for " << firstName
58         << ' ' << lastName << " called." << endl;
59 } // end Employee constructor
60
61 // Destructor deallocates dynamically allocated memory
62 Employee::~Employee()
63 {
64    cout << "~Employee() called for " << firstName
65         << ' ' << lastName << endl;
66    delete [] firstName;  // recapture memory
67    delete [] lastName;   // recapture memory
68    --count;  // decrement static count of employees
69 } // end Employee destructor
70
```



employ.cpp (Part 2 of 3)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
80 // Return last name of employee
81 const char *Employee::getLastName() const
82 {
83    // Const before return type prevents client from modifying
84    // private data. Client should copy returned string before
85    // destructor deletes storage to prevent undefined pointer.
86    return lastName;
87 } // end function getLastName
```

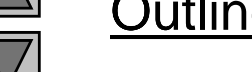
```
88 // Fig. 17.9: fig17_09.cpp
89 // Driver to test the employee class
90 #include <iostream>
91
92 using std::cout;
93 using std::endl;
94
95 #include "employ1.h"
96
97 int main()
98 {
99    cout << "Number of employees before instantiation is "
100         << Employee::getCount() << endl;   // use class name
101
102    Employee *e1Ptr = new Employee( "Susan", "Baker" );
103    Employee *e2Ptr = new Employee( "Robert", "Jones" );
104
105    cout << "Number of employees after instantiation is "
106         << e1Ptr->getCount();
107
```

Outline

employ.cpp (Part 3 of 3)

fig17_09.cpp (Part 1 of 2)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Outline

fig17_09.cpp (Part 1 of 2)

```
108    cout << "\n\nEmployee 1: "
109         << e1Ptr->getFirstName()
110         << " " << e1Ptr->getLastName()
111         << "\nEmployee 2: "
112         << e2Ptr->getFirstName()
113         << " " << e2Ptr->getLastName() << "\n\n";
114
115    delete e1Ptr;   // recapture memory
116    e1Ptr = 0;
117    delete e2Ptr;   // recapture memory
118    e2Ptr = 0;
119
120    cout << "Number of employees after deletion is "
121         << Employee::getCount() << endl;
122
123    return 0;
124 } // end function main
```

Program Output

```
Number of employees before instantiation is 0
Employee constructor for Susan Baker called.
Employee constructor for Robert Jones called.
Number of employees after instantiation is 2

Employee 1: Susan Baker
Employee 2: Robert Jones

~Employee() called for Susan Baker
~Employee() called for Robert Jones
Number of employees after deletion is 0
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

17.8 Data Abstraction and Information Hiding

- Information hiding
  - Classes hide implementation details from clients
  - Example: stack data structure
    - Data elements like a pile of dishes - added (pushed) and removed (popped) from top
    - Last-in, first-out (LIFO) data structure
  - Client does not care how stack is implemented, only wants LIFO data structure

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

17.8 Data Abstraction and Information Hiding (II)

- Abstract data types (ADTs)
  - Model real world objects
    - `int, float` are models for a number
    - Imperfect - finite size, precision, etc.
- C++ an extensible language
  - Base cannot be changed, but new data types can be created

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

17.8.1 Example: Array Abstract Data Type

- Array
  - Essentially a pointer and memory locations
- Programmer can make an ADT array
  - New capabilities
    - Subscript range checking, array assignment and comparison, dynamic arrays, arrays that know their sizes...
- New classes
  - Proprietary to an individual, to small groups or to companies, or placed in standard class libraries

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

17.8.2 Example: String Abstract Data Type

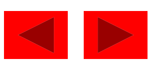
- C++ intentionally sparse
  - Reduce performance burdens
  - Use language to create what you need, i.e. a `string` class
- `string` not a built-in data type
  - Instead, C++ enables you to create your own `string` class

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

17.8.3 Example: Queue Abstract Data Type

- Queue - a waiting line
  - Used by computer systems internally
  - We need programs that simulate queues
- Queue has well-understood behavior
  - Enqueue - put things in a queue one at a time
  - Dequeue - get those things back one at a time on demand
  - Implementation hidden from clients
- Queue ADT - stable internal data structure
  - Clients may not manipulate data structure directly
  - Only queue member functions can access internal data

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

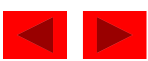


17.9 Container Classes and Iterators

- Container classes (collection classes)
  - Classes designed to hold collections of objects
  - Services such as insertion, deletion, searching, sorting, or testing an item

  Examples:
  - Arrays, stacks, queues, trees and linked lists
- Iterator objects (iterators)
  - Object that returns the next item of a collection (or some action)
  - Can have several iterators per container
    - Book with multiple bookmarks
  - Each iterator maintains its own "position" information

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

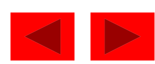


Chapter 18 - C++ Operator Overloading

<u>**Outline**</u>

18.1 Introduction
18.2 Fundamentals of Operator Overloading
18.3 Restrictions on Operator Overloading
18.4 Operator Functions as Class Members vs. as `friend` Functions
18.5 Overloading Stream-Insertion and Stream-Extraction Operators
18.6 Overloading Unary Operators
18.7 Overloading Binary Operators
18.8 Case Study: An `Array` Class
18.9 Converting between Types
18.10 Overloading `++` and `--`

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Objectives

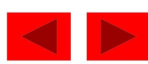
- In this chapter, you will learn:
  - To understand how to redefine (overload) operators to work with new types.
  - To understand how to convert objects from one class to another class.
  - To learn when to, and when not to, overload operators.
  - To study several interesting classes that use overloaded operators.
  - To create an `Array` class.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

18.1 Introduction

- Chapter 16 and 17
  - ADT's and classes
  - Function-call notation is cumbersome for certain kinds of classes, especially mathematical classes
- In this chapter
  - We use C++'s built-in operators to work with class objects

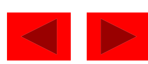
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



18.1 Introduction

- Operator overloading
  - Use traditional operators with user-defined objects
  - Straightforward and natural way to extend C++
  - Requires great care
    - When overloading is misused, programs become difficult to understand

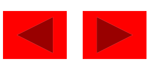
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



18.2 Fundamentals of Operator Overloading

- Use operator overloading to improve readability
  - Avoid excessive or inconsistent usage
- Format
  - Write function definition as normal
  - Function name is keyword `operator` followed by the symbol for the operator being overloaded.
  - `operator+` would be used to overload the addition operator (+)

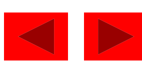
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



18.2 Fundamentals of Operator Overloading

- Assignment operator (**=**)
  - may be used with every class without explicit overloading
  - *memberwise assignment*
  - Same is true for the address operator (&)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



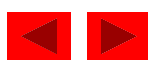
18.3 Restrictions on Operator Overloading

<table>
<tr><th colspan="8">Operators that can be overloaded</th></tr>
<tr><td>+</td><td>-</td><td>*</td><td>/</td><td>%</td><td>^</td><td>&</td><td>|</td></tr>
<tr><td>~</td><td>!</td><td>=</td><td><</td><td>></td><td>+=</td><td>-=</td><td>*=</td></tr>
<tr><td>/=</td><td>%=</td><td>^=</td><td>&=</td><td>|=</td><td><<</td><td>>></td><td>>>=</td></tr>
<tr><td><<=</td><td>==</td><td>!=</td><td><=</td><td>>=</td><td>&&</td><td>||</td><td>++</td></tr>
<tr><td>--</td><td>->*</td><td>,</td><td>-></td><td>[]</td><td>()</td><td>new</td><td>delete</td></tr>
<tr><td>new[]</td><td>delete[]</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
</table>

Fig. 18.1 Operators that can be overloaded.

- Most of C++’s operators can be overloaded

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

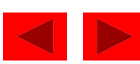


18.3 Restrictions on Operator Overloading

Operators that cannot be overloaded				
.	.*	::	?:	sizeof

Fig. 18.2 Operators that cannot be overloaded.

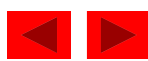
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



18.3 Restrictions on Operator Overloading

- Arity (number of operands) cannot be changed
  - Unary operators remain unary, and binary operators remain binary
  - Operators **&**, __*__, **+** and **-** each have unary and binary versions
    - Unary and binary versions can be overloaded separately

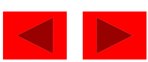
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



18.3 Restrictions on Operator Overloading

- No new operators can be created
  - Use only existing operators
- Built-in types
  - Cannot overload operators
  - You cannot change how two integers are added

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



18.4 Operator Functions as Class Members vs. as `friend` Functions

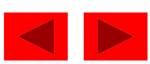
- Operator functions
  - Can be member or non-member functions
- Overloading the assignment operators
  - i.e: `()`, `[]`, `->`, `=`
  - Operator must be a member function

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

18.4 Operator Functions as Class Members vs. as `friend` Functions

- Operator functions as member functions
  - Leftmost operand must be an object (or reference to an object) of the class
  - If left operand of a different type, operator function must be a non-member function
  - A non-member operator function must be a `friend` if `private` or `protected` members of that class are accessed directly

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



18.4 Operator Functions as Class Members vs. as `friend` Functions

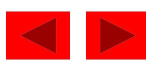
- Non-member overloaded operator functions
  - Enable the operator to be commutative

```
HugeInteger bigInteger1;
long int number;
bigInteger1 = number + bigInteger1;
```

or

```
bigInteger1 = biginteger1 + number;
```

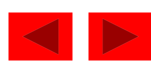
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

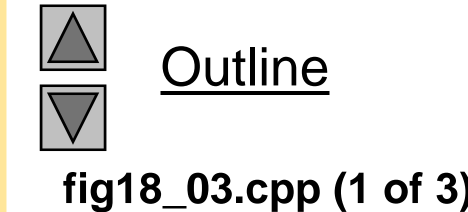


18.5 Overloading Stream-Insertion and Stream-Extraction Operators

- Overloaded `<<` and `>>` operators
  - Must have left operand of types `ostream &`, `istream &` respectively
  - It must be a non-member function (left operand not an object of the class)
  - It must be a `friend` function if it accesses private data members

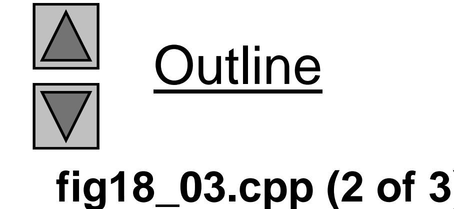
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.





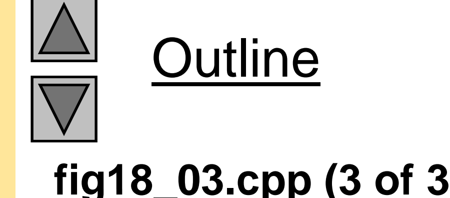

```
1  // Fig. 18.3: fig18_03.cpp
2  // Overloading the stream-insertion and
3  // stream-extraction operators.
4  #include <iostream>
5
6  using std::cout;
7  using std::cin;
8  using std::endl;
9  using std::ostream;
10 using std::istream;
11
12 #include <iomanip>
13
14 using std::setw;
15
16 class PhoneNumber {
17    friend ostream &operator<<( ostream&, const PhoneNumber & );
18    friend istream &operator>>( istream&, PhoneNumber & );
19
20 private:
21    char areaCode[ 4 ];  // 3-digit area code and null
22    char exchange[ 4 ];  // 3-digit exchange and null
23    char line[ 5 ];      // 4-digit line and null
24 }; // end class PhoneNumber
25
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.




```
26 // Overloaded stream-insertion operator (cannot be
27 // a member function if we would like to invoke it with
28 // cout << somePhoneNumber;).
29 ostream &operator<<( ostream &output, const PhoneNumber &num )
30 {
31    output << "(" << num.areaCode << ") "
32           << num.exchange << "-" << num.line;
33    return output;     // enables cout << a << b << c;
34 } // end operator<< function
35
36 istream &operator>>( istream &input, PhoneNumber &num )
37 {
38    input.ignore();                     // skip (
39    input >> setw( 4 ) >> num.areaCode; // input area code
40    input.ignore( 2 );                  // skip ) and space
41    input >> setw( 4 ) >> num.exchange; // input exchange
42    input.ignore();                     // skip dash (-)
43    input >> setw( 5 ) >> num.line;     // input line
44    return input;      // enables cin >> a >> b >> c;
45 } // end operator>> function
46
47 int main()
48 {
49    PhoneNumber phone; // create object phone
50
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.




```
51    cout << "Enter phone number in the form (123) 456-7890:\n";
52
53    // cin >> phone invokes operator>> function by
54    // issuing the call operator>>( cin, phone ).
55    cin >> phone;
56
57    // cout << phone invokes operator<< function by
58    // issuing the call operator<<( cout, phone ).
59    cout << "The phone number entered was: " << phone << endl;
60    return 0;
61 } // end function main
```

```
Enter phone number in the form (123) 456-7890:
(800) 555-1212
The phone number entered was: (800) 555-1212
```

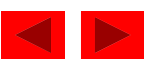
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

18.6 Overloading Unary Operators

- Overloading unary operators
  - Avoid `friend` functions and `friend` classes unless absolutely necessary.
  - Use of `friends` violates the encapsulation of a class.
  - As a member function:

```
class String {
  public:
     bool operator!() const;
     ...
  };
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



18.7 Overloading Binary Operators

- Overloaded binary operators
  - Non-`static` member function, one argument
  - Non-member function, two arguments

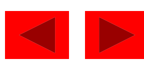
```
class String {
public:
   const String &operator+=( const String & );
   ...
}; // end class String
```

```
   y += z;
```

equivalent to

```
y.operator+=( z );
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



18.7 Overloading Binary Operators

- Example

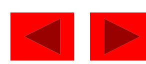
```
class String {
   friend const String &operator+=( String &,
                                    const String & );
   ...
}; // end class String
```

```
   y += z;
```

equivalent to

```
 operator+=( y, z );
```

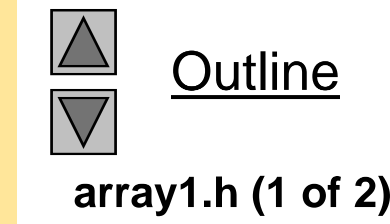
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



18.8 Case Study: An `Array` class

- Implement an Array class with
  - Range checking
  - Array assignment
  - Arrays that know their size
  - Outputting/inputting entire arrays with **`<<`** and **`>>`**
  - Array comparisons with **`==`** and **`!=`**

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



array1.h (1 of 2)

```
1  // Fig. 18.4: array1.h
2  // Simple class Array (for integers)
3  #ifndef ARRAY1_H
4  #define ARRAY1_H
5
6  #include <iostream>
7
8  using std::ostream;
9  using std::istream;
10
11 class Array {
12    friend ostream &operator<<( ostream &, const Array & );
13    friend istream &operator>>( istream &, Array & );
14 public:
15    Array( int = 10 );                       // default constructor
16    Array( const Array & );                  // copy constructor
17    ~Array();                                // destructor
18    int getSize() const;                     // return size
19    const Array &operator=( const Array & ); // assign arrays
20    bool operator==( const Array & ) const;  // compare equal
21
22    // Determine if two arrays are not equal and
23    // return true, otherwise return false (uses operator==).
24    bool operator!=( const Array &right ) const
25       { return ! ( *this == right ); }
26
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
27    int &operator[]( int );               // subscript operator
28    const int &operator[]( int ) const;   // subscript operator
29    static int getArrayCount();           // Return count of
30                                          // arrays instantiated.
31 private:
32    int size; // size of the array
33    int *ptr; // pointer to first element of array
34    static int arrayCount;  // # of Arrays instantiated
35 }; // end class Array
36
37 #endif
```

```
38 // Fig 18.4: array1.cpp
39 // Member function definitions for class Array
40 #include <iostream>
41
42 using std::cout;
43 using std::cin;
44 using std::endl;
45
46 #include <iomanip>
47
48 using std::setw;
49
50 #include <cstdlib>
51 #include <cassert>
52 #include "array1.h"
53
```

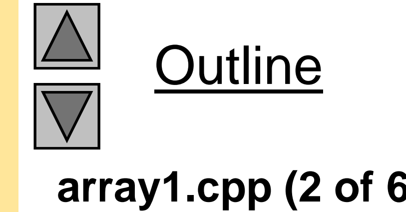
Outline

array1.h (2 of 2)

array1.cpp (1 of 6)

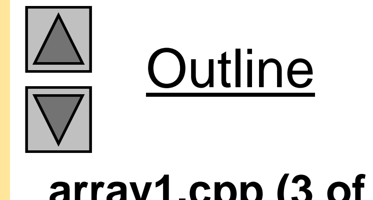
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
54 // Initialize static data member at file scope
55 int Array::arrayCount = 0;   // no objects yet
56
57 // Default constructor for class Array (default size 10)
58 Array::Array( int arraySize )
59 {
60    size = ( arraySize > 0 ? arraySize : 10 );
61    ptr = new int[ size ]; // create space for array
62    assert( ptr != 0 );    // terminate if memory not allocated
63    ++arrayCount;          // count one more object
64
65    for ( int i = 0; i < size; i++ )
66       ptr[ i ] = 0;          // initialize array
67 } // end Array constructor
68
69 // Copy constructor for class Array
70 // must receive a reference to prevent infinite recursion
71 Array::Array( const Array &init ) : size( init.size )
72 {
73    ptr = new int[ size ]; // create space for array
74    assert( ptr != 0 );    // terminate if memory not allocated
75    ++arrayCount;          // count one more object
76
77    for ( int i = 0; i < size; i++ )
78       ptr[ i ] = init.ptr[ i ];  // copy init into object
79 } // end Array constructor
80
```

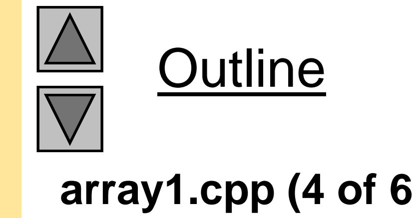


© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
81 // Destructor for class Array
82 Array::~Array()
83 {
84    delete [] ptr;            // reclaim space for array
85    --arrayCount;             // one fewer object
86 } // end Array destructor
87
88 // Get the size of the array
89 int Array::getSize() const { return size; }
90
91 // Overloaded assignment operator
92 // const return avoids: ( a1 = a2 ) = a3
93 const Array &Array::operator=( const Array &right )
94 {
95    if ( &right != this ) {  // check for self-assignment
96
97       // for arrays of different sizes, deallocate original
98       // left side array, then allocate new left side array.
99       if ( size != right.size ) {
100          delete [] ptr;         // reclaim space
101          size = right.size;     // resize this object
102          ptr = new int[ size ]; // create space for array copy
103          assert( ptr != 0 );    // terminate if not allocated
104       } // end if
105
106       for ( int i = 0; i < size; i++ )
107          ptr[ i ] = right.ptr[ i ];  // copy array into object
108    } // end if
109
```

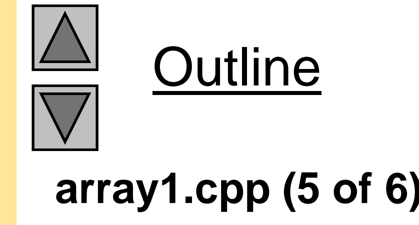


```
110    return *this;    // enables x = y = z;
111 } // end operator= function
112
113 // Determine if two arrays are equal and
114 // return true, otherwise return false.
115 bool Array::operator==( const Array &right ) const
116 {
117    if ( size != right.size )
118       return false;    // arrays of different sizes
119
120    for ( int i = 0; i < size; i++ )
121       if ( ptr[ i ] != right.ptr[ i ] )
122          return false; // arrays are not equal
123
124    return true;        // arrays are equal
125 } // end operator== function
126
127 // Overloaded subscript operator for non-const Arrays
128 // reference return creates an lvalue
129 int &Array::operator[]( int subscript )
130 {
131    // check for subscript out of range error
132    assert( 0 <= subscript && subscript < size );
133
134    return ptr[ subscript ]; // reference return
135 } // end operator[] function
136
```



© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
137 // Overloaded subscript operator for const Arrays
138 // const reference return creates an rvalue
139 const int &Array::operator[]( int subscript ) const
140 {
141    // check for subscript out of range error
142    assert( 0 <= subscript && subscript < size );
143
144    return ptr[ subscript ]; // const reference return
145 } // end operator[] function
146
147 // Return the number of Array objects instantiated
148 // static functions cannot be const
149 int Array::getArrayCount() { return arrayCount; }
150
151 // Overloaded input operator for class Array;
152 // inputs values for entire array.
153 istream &operator>>( istream &input, Array &a )
154 {
155    for ( int i = 0; i < a.size; i++ )
156       input >> a.ptr[ i ];
157
158    return input;   // enables cin >> x >> y;
159 } // end operator>> function
160
```



array1.cpp (5 of 6)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Outline

array1.cpp (6 of 6)

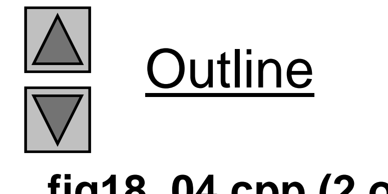
```
161 // Overloaded output operator for class Array
162 ostream &operator<<( ostream &output, const Array &a )
163 {
164    int i;
165
166    for ( i = 0; i < a.size; i++ ) {
167       output << setw( 12 ) << a.ptr[ i ];
168
169       if ( ( i + 1 ) % 4 == 0 ) // 4 numbers per row of output
170          output << endl;
171    } // end for
172
173    if ( i % 4 != 0 )
174       output << endl;
175
176    return output;   // enables cout << x << y;
177 } // end operator<< function
```

fig18_04.cpp (1 of 4)

```
178 // Fig. 18.4: fig18_04.cpp
179 // Driver for simple class Array
180 #include <iostream>
181
182 using std::cout;
183 using std::cin;
184 using std::endl;
185
186 #include "array1.h"
187
```

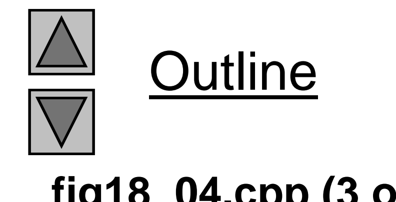
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
188 int main()
189 {
190    // no objects yet
191    cout << "# of arrays instantiated = "
192         << Array::getArrayCount() << '\n';
193
194    // create two arrays and print Array count
195    Array integers1( 7 ), integers2;
196    cout << "# of arrays instantiated = "
197         << Array::getArrayCount() << "\n\n";
198
199    // print integers1 size and contents
200    cout << "Size of array integers1 is "
201         << integers1.getSize()
202         << "\nArray after initialization:\n"
203         << integers1 << '\n';
204
205    // print integers2 size and contents
206    cout << "Size of array integers2 is "
207         << integers2.getSize()
208         << "\nArray after initialization:\n"
209         << integers2 << '\n';
210
```

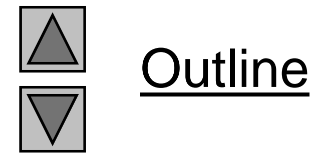


© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
211    // input and print integers1 and integers2
212    cout << "Input 17 integers:\n";
213    cin >> integers1 >> integers2;
214    cout << "After input, the arrays contain:\n"
215         << "integers1:\n" << integers1
216         << "integers2:\n" << integers2 << '\n';
217
218    // use overloaded inequality (!=) operator
219    cout << "Evaluating: integers1 != integers2\n";
220    if ( integers1 != integers2 )
221       cout << "They are not equal\n";
222
223    // create array integers3 using integers1 as an
224    // initializer; print size and contents
225    Array integers3( integers1 );
226
227    cout << "\nSize of array integers3 is "
228         << integers3.getSize()
229         << "\nArray after initialization:\n"
230         << integers3 << '\n';
231
232    // use overloaded assignment (=) operator
233    cout << "Assigning integers2 to integers1:\n";
234    integers1 = integers2;
235    cout << "integers1:\n" << integers1
236         << "integers2:\n" << integers2 << '\n';
237
```



© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Outline

fig18_04.cpp (4 of 4)

```
238    // use overloaded equality (==) operator
239    cout << "Evaluating: integers1 == integers2\n";
240    if ( integers1 == integers2 )
241       cout << "They are equal\n\n";
242
243    // use overloaded subscript operator to create rvalue
244    cout << "integers1[5] is " << integers1[ 5 ] << '\n';
245
246    // use overloaded subscript operator to create lvalue
247    cout << "Assigning 1000 to integers1[5]\n";
248    integers1[ 5 ] = 1000;
249    cout << "integers1:\n" << integers1 << '\n';
250
251    // attempt to use out of range subscript
252    cout << "Attempt to assign 1000 to integers1[15]" << endl;
253    integers1[ 15 ] = 1000;  // ERROR: out of range
254
255    return 0;
256 } // end function main
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
# of arrays instantiated = 0
# of arrays instantiated = 2

Size of array integers1 is 7
Array after initialization:
           0           0           0           0
           0           0           0

Size of array integers2 is 10
Array after initialization:
           0           0           0           0
           0           0           0           0
           0           0

Input 17 integers:
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
After input, the arrays contain:
integers1:
           1           2           3           4
           5           6           7
integers2:
           8           9          10          11
          12          13          14          15
          16          17

Evaluating: integers1 != integers2
They are not equal

Size of array integers3 is 7
Array after initialization:
           1           2           3           4
           5           6           7
```

Outline

Array output (1 of 1)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
Assigning integers2 to integers1:
integers1:
           8           9          10          11
          12          13          14          15
          16          17
integers2:
           8           9          10          11
          12          13          14          15
          16          17

Evaluating: integers1 == integers2
They are equal

integers1[5] is 13
Assigning 1000 to integers1[5]
integers1:
           8           9          10          11
          12        1000          14          15
          16          17

Attempt to assign 1000 to integers1[15]
Assertion failed: 0 <= subscript && subscript < size, file Array1.cpp,
line 95 abnormal program termination
```

Outline

Array output (2 of 2)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

18.9 Converting between Types

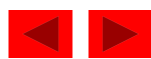
- Cast operator
  - Convert objects into built-in types or other objects
  - Conversion operator must be a non-`static` member function.
  - Cannot be a `friend` function
  - Do not specify return type

For user-defined class `A`

```
A::operator char *() const;
A::operator int() const;
A::operator otherClass() const;
```

- When compiler sees `(char *) s` it calls `s.operator char*()`

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



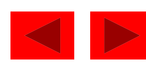
18.9 Converting between Types

- The compiler can call these functions to create temporary objects.
  - If `s` is not of type `char *`

Calls `A::operator char *() const;` for

```
cout << s;
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



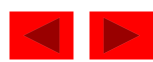
18.10 Overloading `++` and `--`

- Pre/post-incrementing/decrementing operators
  - Can be overloaded
  - How does the compiler distinguish between the two?
  - Prefix versions overloaded same as any other prefix unary operator would be. i.e. `d1.operator++();` for `++d1;`
- Postfix versions
  - When compiler sees postincrementing expression, such as `d1++;`
    - Generates the member-function call

      ```
      d1.operator++( 0 );
      ```
  - Prototype:

    ```
    Date::operator++( int );
    ```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

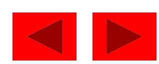


Chapter 19 - C++ Inheritance

Outline

19.1 Introduction
19.2 Inheritance: Base Classes and Derived Classes
19.3 Protected Members
19.4 Casting Base-Class Pointers to Derived-Class Pointers
19.5 Using Member Functions
19.6 Overriding Base-Class Members in a Derived Class
19.7 Public, Protected and Private Inheritance
19.8 Direct Base Classes and Indirect Base Classes
19.9 Using Constructors and Destructors in Derived Classes
19.10 Implicit Derived-Class Object to Base-Class Object Conversion
19.11 Software Engineering with Inheritance
19.12 Composition vs. Inheritance
19.13 *Uses A* and *Knows A* Relationships
19.14 Case Study: Point, Circle, Cylinder

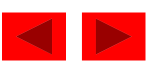
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Objectives

- In this chapter, you will learn:
  - To be able to create new classes by inheriting from existing classes.
  - To understand how inheritance promotes software reusability.
  - To understand the notions of base classes and derived classes.

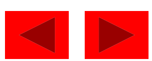
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



19.1 Introduction

- Inheritance
  - New classes created from existing classes
  - Absorb attributes and behaviors.
- Polymorphism
  - Write programs in a general fashion
  - Handle a wide variety of existing (and unspecified) related classes
- Derived class
  - Class that inherits data members and member functions from a previously defined base class

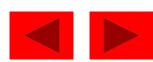
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



19.1 Introduction

- Inheritance
  - Single Inheritance
    - Class inherits from one base class
  - Multiple Inheritance
    - Class inherits from multiple base classes
  - Three types of inheritance:
    - `public`: Derived objects are accessible by the base class objects (focus of this chapter)
    - `private`: Derived objects are inaccessible by the base class
    - `protected`: Derived classes and `friends` can access protected members of the base class

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



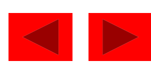
19.2 Base and Derived Classes

- Often an object from a derived class (subclass) “is an” object of a base class (superclass)

Base class	Derived classes
Student	GraduateStudent UndergraduateStudent
Shape	Circle Triangle Rectangle
Loan	CarLoan HomeImprovementLoan MortgageLoan
Employee	FacultyMember StaffMember
Account	CheckingAccount SavingsAccount

Fig. 19.1 Some simple inheritance examples.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



19.2 Base and Derived Classes

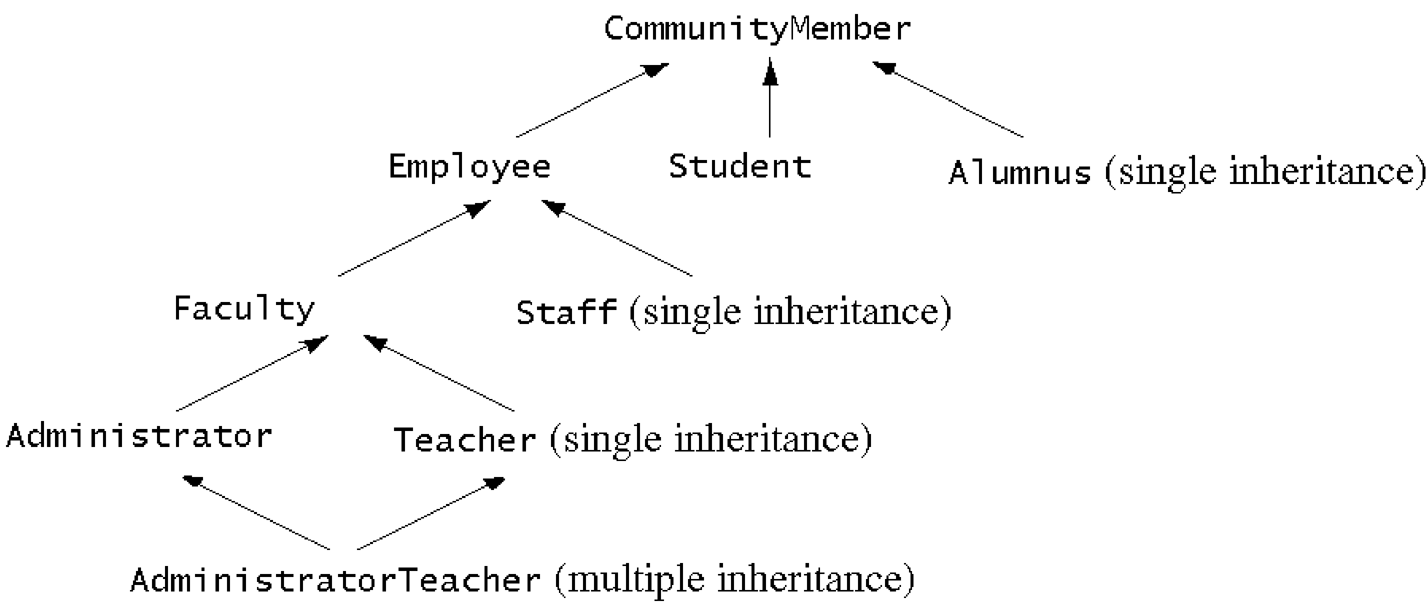
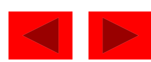


Fig. 19.2 An inheritance hierarchy for university community members.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



19.2 Base and Derived Classes

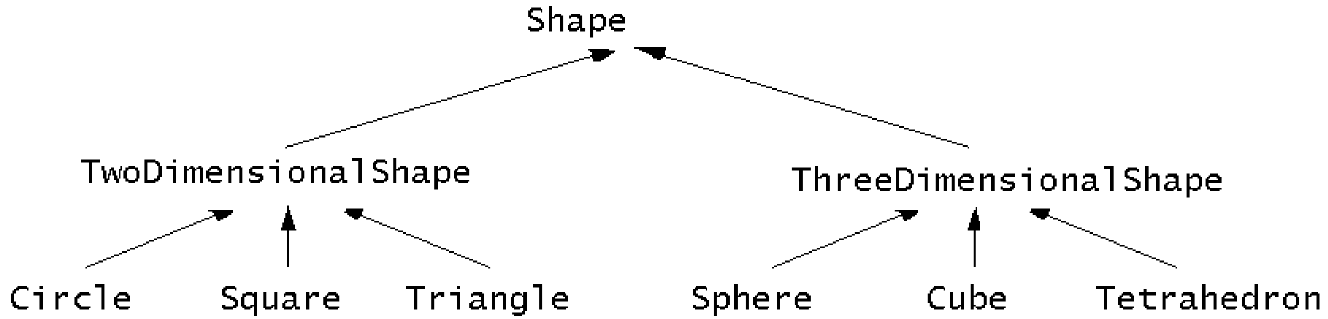
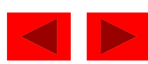


Fig. 19.3 A portion of a Shape class hierarchy.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



19.2 Base and Derived Classes

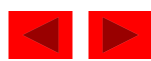
- Implementation of `public` inheritance

```
class CommissionWorker : public Employee {
    ...
};
```

Class `CommissionWorker` inherits from class `Employee`

- `friend` functions not inherited
- `private` members of base class not accessible from derived class

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



19.3 Protected Members

- `protected` inheritance
  - Intermediate level of protection between `public` and `private` inheritance
  - Derived-class members can refer to `public` and `protected` members of the base class simply by using the member names
  - Note that `protected` data "breaks" encapsulation

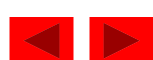
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

19.4 Casting Base Class Pointers to Derived Class Pointers

- Object of a derived class
  - Can be treated as an object of the base class
  - Reverse not true - base class objects not a derived-class object
- Downcasting a pointer
  - Use an explicit cast to convert a base-class pointer to a derived-class pointer
  - Be sure that the type of the pointer matches the type of object to which the pointer points

```
derivedPtr = static_cast< DerivedClass * > basePtr;
```

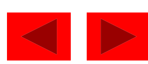
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

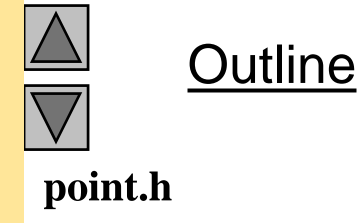


19.4 Casting Base-Class Pointers to Derived-Class Pointers

- Example
  - `Circle class` derived from the `Point` base `class`
  - We use pointer of type `Point` to reference a `Circle` object, and vice-versa

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.





point.h

```
1    // Fig. 19.4: point.h
2    // Definition of class Point
3    #ifndef POINT_H
4    #define POINT_H
5
6    #include <iostream>
7
8    using std::ostream;
9
10   class Point {
11      friend ostream &operator<<( ostream &, const Point & );
12   public:
13      Point( int = 0, int = 0 );      // default constructor
14      void setPoint( int, int );      // set coordinates
15      int getX() const { return x; }  // get x coordinate
16      int getY() const { return y; }  // get y coordinate
17   protected:         // accessible by derived classes
18      int x, y;       // x and y coordinates of the Point
19   }; // end class Point
20
21   #endif
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

point.cpp

```
22  // Fig. 19.4: point.cpp
23  // Member functions for class Point
24  #include <iostream>
25  #include "point.h"
26
27  // Constructor for class Point
28  Point::Point( int a, int b ) { setPoint( a, b ); }
29
30  // Set x and y coordinates of Point
31  void Point::setPoint( int a, int b )
32  {
33     x = a;
34     y = b;
35  } // end function setPoint
36
37  // Output Point (with overloaded stream insertion operator)
38  ostream &operator<<( ostream &output, const Point &p )
39  {
40     output << '[' << p.x << ", " << p.y << ']';
41
42     return output;   // enables cascaded calls
43  } // end operator<< function
```

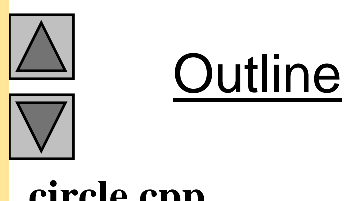
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
44  // Fig. 19.4: circle.h
45  // Definition of class Circle
46  #ifndef CIRCLE_H
47  #define CIRCLE_H
48
49  #include <iostream>
50
51  using std::ostream;
52
53  #include <iomanip>
54
55  using std::ios;
56  using std::setiosflags;
57  using std::setprecision;
58
59  #include "point.h"
60
61  class Circle : public Point {  // Circle inherits from Point
62     friend ostream &operator<<( ostream &, const Circle & );
63  public:
64     // default constructor
65     Circle( double r = 0.0, int x = 0, int y = 0 );
66
67     void setRadius( double );   // set radius
68     double getRadius() const;   // return radius
69     double area() const;        // calculate area
70  protected:
71     double radius;
72  }; // end class Circle
73
74  #endif
```

Outline

circle.h

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



circle.cpp

```
75   // Fig. 19.4: circle.cpp
76   // Member function definitions for class Circle
77   #include "circle.h"
78
79   // Constructor for Circle calls constructor for Point
80   // with a member initializer then initializes radius.
81   Circle::Circle( double r, int a, int b )
82      : Point( a, b )       // call base-class constructor
83   { setRadius( r ); }
84
85   // Set radius of Circle
86   void Circle::setRadius( double r )
87      { radius = ( r >= 0 ? r : 0 ); }
88
89   // Get radius of Circle
90   double Circle::getRadius() const { return radius; }
91
92   // Calculate area of Circle
93   double Circle::area() const
94      { return 3.14159 * radius * radius; }
95
96   // Output a Circle in the form:
97   // Center = [x, y]; Radius = #.##
98   ostream &operator<<( ostream &output, const Circle &c )
99   {
100     output << "Center = " << static_cast< Point >( c )
101            << "; Radius = "
102            << setiosflags( ios::fixed | ios::showpoint )
103            << setprecision( 2 ) << c.radius;
104
105     return output;   // enables cascaded calls
106  } // end operator<< function
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Outline

fig19_04.cpp (1 of 2)

```
107 // Fig. 19.4: fig19_04.cpp
108 // Casting base-class pointers to derived-class pointers
109 #include <iostream>
110
111 using std::cout;
112 using std::endl;
113
114 #include <iomanip>
115
116 #include "point.h"
117 #include "circle.h"
118
119 int main()
120 {
121    Point *pointPtr = 0, p( 30, 50 );
122    Circle *circlePtr = 0, c( 2.7, 120, 89 );
123
124    cout << "Point p: " << p << "\nCircle c: " << c << '\n';
125
126    // Treat a Circle as a Point (see only the base class part)
127    pointPtr = &c;   // assign address of Circle to pointPtr
128    cout << "\nCircle c (via *pointPtr): "
129         << *pointPtr << '\n';
130
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

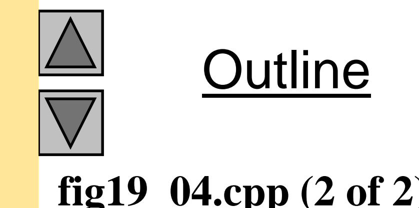


fig19_04.cpp (2 of 2)

```
131    // Treat a Circle as a Circle (with some casting)
132    // cast base-class pointer to derived-class pointer
133    circlePtr = static_cast< Circle * >( pointPtr );
134    cout << "\nCircle c (via *circlePtr):\n" << *circlePtr
135         << "\nArea of c (via circlePtr): "
136         << circlePtr->area() << '\n';
137
138    // DANGEROUS: Treat a Point as a Circle
139    pointPtr = &p;   // assign address of Point to pointPtr
140
141    // cast base-class pointer to derived-class pointer
142    circlePtr = static_cast< Circle * >( pointPtr );
143    cout << "\nPoint p (via *circlePtr):\n" << *circlePtr
144         << "\nArea of object circlePtr points to: "
145         << circlePtr->area() << endl;
146    return 0;
147 } // end function main
```

```
Point p: [30, 50]
Circle c: Center = [120, 89]; Radius = 2.70

Circle c (via *pointPtr): [120, 89]

Circle c (via *circlePtr):
Center = [120, 89]; Radius = 2.70
Area of c (via circlePtr): 22.90

Point p (via *circlePtr):
Center = [30, 50]; Radius = 0.00
Area of object circlePtr points to: 0.00
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

19.5 Using Member Functions

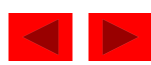
- Derived class
  - Cannot directly access `private` members of its base class
  - Hiding `private` members is a huge help in testing, debugging and correctly modifying systems

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

19.6 Overriding Base-Class Members in a Derived Class

- To override a base-class member function
  - In derived class, supply new version of that function
    - Same function name, different definition
  - The scope-resolution operator may be used to access the base class version from the derived class

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Outline

employ.h

```
1    // Fig. 19.5: employ.h
2    // Definition of class Employee
3    #ifndef EMPLOY_H
4    #define EMPLOY_H
5
6    class Employee {
7    public:
8       Employee( const char *, const char * );  // constructor
9       void print() const;  // output first and last name
10      ~Employee();         // destructor
11   private:
12      char *firstName;     // dynamically allocated string
13      char *lastName;      // dynamically allocated string
14   }; // end class Employee
15
16   #endif
```

employ.cpp (1 of 2)

```
17   // Fig. 19.5: employ.cpp
18   // Member function definitions for class Employee
19   #include <iostream>
20
21   using std::cout;
22
23   #include <cstring>
24   #include <cassert>
25   #include "employ.h"
26
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
27   // Constructor dynamically allocates space for the
28   // first and last name and uses strcpy to copy
29   // the first and last names into the object.
30   Employee::Employee( const char *first, const char *last )
31   {
32      firstName = new char[ strlen( first ) + 1 ];
33      assert( firstName != 0 ); // terminate if not allocated
34      strcpy( firstName, first );
35
36      lastName = new char[ strlen( last ) + 1 ];
37      assert( lastName != 0 );  // terminate if not allocated
38      strcpy( lastName, last );
39   } // end Employee constructor
40
41   // Output employee name
42   void Employee::print() const
43      { cout << firstName << ' ' << lastName; }
44
45   // Destructor deallocates dynamically allocated memory
46   Employee::~Employee()
47   {
48      delete [] firstName;   // reclaim dynamic memory
49      delete [] lastName;    // reclaim dynamic memory
50   } // end Employee destructor
```


© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Outline

hourly.h

```
51  // Fig. 19.5: hourly.h
52  // Definition of class HourlyWorker
53  #ifndef HOURLY_H
54  #define HOURLY_H
55
56  #include "employ.h"
57
58  class HourlyWorker : public Employee {
59  public:
60     HourlyWorker( const char*, const char*, double, double );
61     double getPay() const;  // calculate and return salary
62     void print() const;     // overridden base-class print
63  private:
64     double wage;            // wage per hour
65     double hours;           // hours worked for week
66  }; // end class HourlyWorker
67
68  #endif
```

hourly.cpp (1 of 2)

```
69  // Fig. 19.5: hourly.cpp
70  // Member function definitions for class HourlyWorker
71  #include <iostream>
72
73  using std::cout;
74  using std::endl;
75
76  #include <iomanip>
77
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
78   using std::ios;
79   using std::setiosflags;
80   using std::setprecision;
81
82   #include "hourly.h"
83
84   // Constructor for class HourlyWorker
85   HourlyWorker::HourlyWorker( const char *first,
86                               const char *last,
87                               double initHours, double initWage )
88      : Employee( first, last )   // call base-class constructor
89   {
90      hours = initHours;  // should validate
91      wage = initWage;    // should validate
92   } // end HourlyWorker constructor
93
94   // Get the HourlyWorker's pay
95   double HourlyWorker::getPay() const { return wage * hours; }
96
97   // Print the HourlyWorker's name and pay
98   void HourlyWorker::print() const
99   {
100     cout << "HourlyWorker::print() is executing\n\n";
101     Employee::print();   // call base-class print function
102
103     cout << " is an hourly worker with pay of $"
104          << setiosflags( ios::fixed | ios::showpoint )
105          << setprecision( 2 ) << getPay() << endl;
106  } // end function print
```

Outline

hourly.cpp (2 of 2)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Outline

fig19_05.cpp

```
107 // Fig. 19.5: fig19_05.cpp
108 // Overriding a base-class member function in a
109 // derived class.
110 #include "hourly.h"
111
112 int main()
113 {
114    HourlyWorker h( "Bob", "Smith", 40.0, 10.00 );
115    h.print();
116    return 0;
117 } // end function main
```

```
HourlyWorker::print() is executing

Bob Smith is an hourly worker with pay of $400.00
```

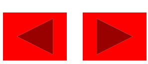
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

19.7 Public, Private, and Protected Inheritance

Base class member access specifier	Type of inheritance		
	public inheritance	protected inheritance	private inheritance
public	public in derived class. Can be accessed directly by any non-static member functions, friend functions and non-member functions.	protected in derived class. Can be accessed directly by all non-static member functions and friend functions.	private in derived class. Can be accessed directly by all non-static member functions and friend functions.
protected	protected in derived class. Can be accessed directly by all non-static member functions and friend functions.	protected in derived class. Can be accessed directly by all non-static member functions and friend functions.	private in derived class. Can be accessed directly by all non-static member functions and friend functions.
private	Hidden in derived class. Can be accessed by non-static member functions and friend functions through public or protected member functions of the base class.	Hidden in derived class. Can be accessed by non-static member functions and friend functions through public or protected member functions of the base class.	Hidden in derived class. Can be accessed by non-static member functions and friend functions through public or protected member functions of the base class.

Fig. 19.6 Summary of base-class member accessibility in a derived class.

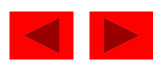
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



19.8 Direct and Indirect Base Classes

- Direct base class
  - Explicitly listed derived class' header with the colon (**`:`**) notation when that derived class is declared.
  - **`class HourlyWorker : public Employee`**
    - **`Employee`** is a direct base class of **`HourlyWorker`**
- Indirect base class
  - Inherited from two or more levels up the class hierarchy
  - **`class MinuteWorker : public HourlyWorker`**
    - **`Employee`** is an indirect base class of **`MinuteWorker`**

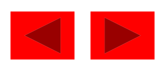
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



19.9 Using Constructors and Destructors in Derived Classes

- Base class initializer
  - Uses member-initializer syntax
  - Can be provided in the derived class constructor to call the base-class constructor explicitly
    - Otherwise base class' default constructor called implicitly
  - Base-class constructors and base-class assignment operators are not inherited by derived classes
    - However, derived-class constructors and assignment operators can call still them

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



19.9 Using Constructors and Destructors in Derived Classes

- Derived-class constructor
  - Calls the constructor for its base class first to initialize its base-class members
  - If the derived-class constructor is omitted, its default constructor calls the base-class' default constructor
- Destructors are called in the reverse order of constructor calls.
  - Derived-class destructor is called before its base-class destructor

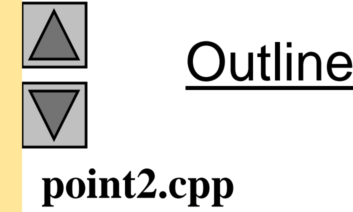
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
1    // Fig. 19.7: point2.h
2    // Definition of class Point
3    #ifndef POINT2_H
4    #define POINT2_H
5
6    class Point {
7    public:
8       Point( int = 0, int = 0 );  // default constructor
9       ~Point();     // destructor
10   protected:       // accessible by derived classes
11      int x, y;     // x and y coordinates of Point
12   }; // end class Point
13
14   #endif
```


point2.h

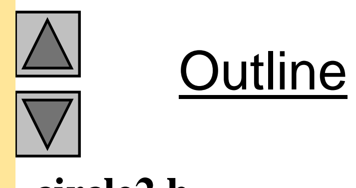
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
15   // Fig. 19.7: point2.cpp
16   // Member function definitions for class Point
17   #include <iostream>
18
19   using std::cout;
20   using std::endl;
21
22   #include "point2.h"
23
24   // Constructor for class Point
25   Point::Point( int a, int b )
26   {
27      x = a;
28      y = b;
29
30      cout << "Point  constructor:  "
31           << '[' << x << ", " << y << ']' << endl;
32   } // end Point constructor
33
34   // Destructor for class Point
35   Point::~Point()
36   {
37      cout << "Point  destructor:   "
38           << '[' << x << ", " << y << ']' << endl;
39   } // end Point destructor
```



point2.cpp

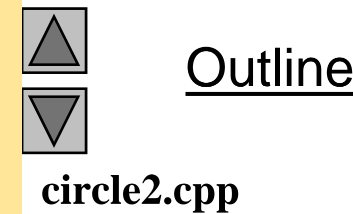
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



circle2.h

```
40  // Fig. 19.7: circle2.h
41  // Definition of class Circle
42  #ifndef CIRCLE2_H
43  #define CIRCLE2_H
44
45  #include "point2.h"
46
47  class Circle : public Point {
48  public:
49     // default constructor
50     Circle( double r = 0.0, int x = 0, int y = 0 );
51
52     ~Circle();
53  private:
54     double radius;
55  }; // end class Circle
56
57  #endif
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



circle2.cpp

```
58  // Fig. 19.7: circle2.cpp
59  // Member function definitions for class Circle
60  #include <iostream>
61
62  using std::cout;
63  using std::endl;
64
65  #include "circle2.h"
66
67  // Constructor for Circle calls constructor for Point
68  Circle::Circle( double r, int a, int b )
69     : Point( a, b )   // call base-class constructor
70  {
71     radius = r;  // should validate
72     cout << "Circle constructor: radius is "
73          << radius << " [" << x << ", " << y << ']' << endl;
74  } // end Circle constructor
75
76  // Destructor for class Circle
77  Circle::~Circle()
78  {
79     cout << "Circle destructor:  radius is "
80          << radius << " [" << x << ", " << y << ']' << endl;
81  } // end Circle destructor
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
82  // Fig. 19.7: fig19_07.cpp
83  // Demonstrate when base-class and derived-class
84  // constructors and destructors are called.
85  #include <iostream>
86
87  using std::cout;
88  using std::endl;
89
90  #include "point2.h"
91  #include "circle2.h"
92
93  int main()
94  {
95     // Show constructor and destructor calls for Point
96     {
97        Point p( 11, 22 );
98     } // end block
99
100    cout << endl;
101    Circle circle1( 4.5, 72, 29 );
102    cout << endl;
103    Circle circle2( 10, 5, 5 );
104    cout << endl;
105    return 0;
106 } // end function main
```

Outline

fig19_07.cpp (1 of 2)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
Point  constructor: [11, 22]
Point  destructor:  [11, 22]

Point  constructor: [72, 29]
Circle constructor: radius is 4.5 [72, 29]

Point  constructor: [5, 5]
Circle constructor: radius is 10 [5, 5]

Circle destructor:  radius is 10 [5, 5]
Point  destructor:  [5, 5]
Circle destructor:  radius is 4.5 [72, 29]
Point  destructor:  [72, 29]
```


© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

19.10 Implicit Derived-Class Object to Base-Class Object Conversion

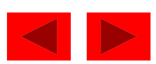
- `baseClassObject = derivedClassObject;`
  - This will work
    - Remember, the derived class object has more members than the base class object
  - Extra data is not given to the base class
- `derivedClassObject = baseClassObject;`
  - May not work properly
    - Unless an assignment operator is overloaded in the derived class, data members exclusive to the derived class will be unassigned
  - Base class has less data members than the derived class
    - Some data members missing in the derived class object

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

19.10 Implicit Derived-Class Object to Base-Class Object Conversion

- Four ways to mix base and derived class pointers and objects:
  - Referring to a base-class object with a base-class pointer
    - Allowed
  - Referring to a derived-class object with a derived-class pointer
    - Allowed
  - Referring to a derived-class object with a base-class pointer
    - Possible syntax error
    - Code can only refer to base-class members, or syntax error
  - Referring to a base-class object with a derived-class pointer
    - Syntax error
    - The derived-class pointer must first be cast to a base-class pointer

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



19.11 Software Engineering With Inheritance

- Classes are often closely related
  - "Factor out" common attributes and behaviors and place these in a base class
  - Use inheritance to form derived classes
- Modifications to a base class
  - Derived classes do not change as long as the `public` and `protected` interfaces are the same
  - Derived classes may need to be recompiled

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

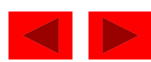
19.12 Composition vs. Inheritance

- "is a" relationship
  - Inheritance
- "has a" relationship
  - Composition - class has an object from another class as a data member

Employee “is a” BirthDate; //Wrong!

Employee “has a” BirthDate; //Composition

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



19.13 *Uses A* And *Knows A* Relationships

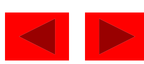
- "uses a" relationship
  - One object issues a function call to a member function of another object
- "knows a" relationship
  - One object is aware of another
    - Contains a pointer handle or reference handle to another object
  - Also called an association

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

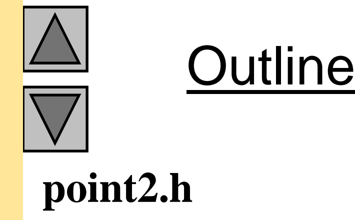
19.14 Case Study: Point, Circle, Cylinder

- Define class `Point`
  - Derive `Circle`
    - Derive `Cylinder`

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

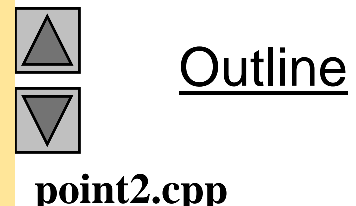


```
1   // Fig. 19.8: point2.h
2   // Definition of class Point
3   #ifndef POINT2_H
4   #define POINT2_H
5
6   #include <iostream>
7
8   using std::ostream;
9
10  class Point {
11     friend ostream &operator<<( ostream &, const Point & );
12  public:
13     Point( int = 0, int = 0 );      // default constructor
14     void setPoint( int, int );      // set coordinates
15     int getX() const { return x; }  // get x coordinate
16     int getY() const { return y; }  // get y coordinate
17  protected:        // accessible to derived classes
18     int x, y;      // coordinates of the point
19  }; // end class Point
20
21  #endif
```



point2.h

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



point2.cpp

```
22  // Fig. 19.8: point2.cpp
23  // Member functions for class Point
24  #include "point2.h"
25
26  // Constructor for class Point
27  Point::Point( int a, int b ) { setPoint( a, b ); }
28
29  // Set the x and y coordinates
30  void Point::setPoint( int a, int b )
31  {
32     x = a;
33     y = b;
34  } // end function setPoint
35
36  // Output the Point
37  ostream &operator<<( ostream &output, const Point &p )
38  {
39     output << '[' << p.x << ", " << p.y << ']';
40
41     return output;          // enables cascading
42  } // end operator<< function
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

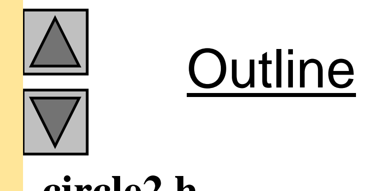
```
43   // Fig. 19.8: fig19_08.cpp
44   // Driver for class Point
45   #include <iostream>
46
47   using std::cout;
48   using std::endl;
49
50   #include "point2.h"
51
52   int main()
53   {
54      Point p( 72, 115 );   // instantiate Point object p
55
56      // protected data of Point inaccessible to main
57      cout << "X coordinate is " << p.getX()
58           << "\nY coordinate is " << p.getY();
59
60      p.setPoint( 10, 10 );
61      cout << "\n\nThe new location of p is " << p << endl;
62
63      return 0;
64   } // end function main
```

```
X coordinate is 72
Y coordinate is 115

The new location of p is [10, 10]
```

Outline

fig19_08.cpp

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



circle2.h

```
1    // Fig. 19.9: circle2.h
2    // Definition of class Circle
3    #ifndef CIRCLE2_H
4    #define CIRCLE2_H
5
6    #include <iostream>
7
8    using std::ostream;
9
10   #include "point2.h"
11
12   class Circle : public Point {
13      friend ostream &operator<<( ostream &, const Circle & );
14   public:
15      // default constructor
16      Circle( double r = 0.0, int x = 0, int y = 0 );
17      void setRadius( double );    // set radius
18      double getRadius() const;    // return radius
19      double area() const;         // calculate area
20   protected:          // accessible to derived classes
21      double radius;   // radius of the Circle
22   }; // end class Circle
23
24   #endif
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Outline

circle2.cpp (1 of 2)

```
25   // Fig. 19.9: circle2.cpp
26   // Member function definitions for class Circle
27   #include <iomanip>
28
29   using std::ios;
30   using std::setiosflags;
31   using std::setprecision;
32
33   #include "circle2.h"
34
35   // Constructor for Circle calls constructor for Point
36   // with a member initializer and initializes radius
37   Circle::Circle( double r, int a, int b )
38      : Point( a, b )       // call base-class constructor
39   { setRadius( r ); }
40
41   // Set radius
42   void Circle::setRadius( double r )
43      { radius = ( r >= 0 ? r : 0 ); }
44
45   // Get radius
46   double Circle::getRadius() const { return radius; }
47
48   // Calculate area of Circle
49   double Circle::area() const
50      { return 3.14159 * radius * radius; }
51
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Outline

circle2.cpp (2 of 2)

```
52   // Output a circle in the form:
53   // Center = [x, y]; Radius = #.##
54   ostream &operator<<( ostream &output, const Circle &c )
55   {
56      output << "Center = " << static_cast< Point > ( c )
57             << "; Radius = "
58             << setiosflags( ios::fixed | ios::showpoint )
59             << setprecision( 2 ) << c.radius;
60
61      return output;   //  enables cascaded calls
62   } // end operator<< function
```

fig19_09.cpp (1 of 2)

```
63   // Fig. 19.9: fig19_09.cpp
64   // Driver for class Circle
65   #include <iostream>
66
67   using std::cout;
68   using std::endl;
69
70   #include "point2.h"
71   #include "circle2.h"
72
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Outline

fig19_09.cpp (2 of 2)

```
73   int main()
74   {
75      Circle c( 2.5, 37, 43 );
76
77      cout << "X coordinate is " << c.getX()
78           << "\nY coordinate is " << c.getY()
79           << "\nRadius is " << c.getRadius();
80
81      c.setRadius( 4.25 );
82      c.setPoint( 2, 2 );
83      cout << "\n\nThe new location and radius of c are\n"
84           << c << "\nArea " << c.area() << '\n';
85
86      Point &pRef = c;
87      cout << "\nCircle printed as a Point is: " << pRef << endl;
88
89      return 0;
90   } // end function main
```

```
X coordinate is 37
Y coordinate is 43
Radius is 2.5

The new location and radius of c are
Center = [2, 2]; Radius = 4.25
Area 56.74

Circle printed as a Point is: [2, 2]
```

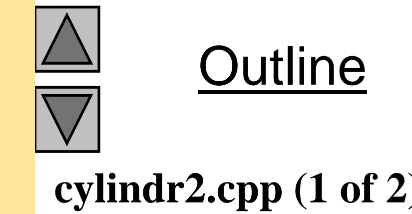
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
1    // Fig. 19.10: cylindr2.h
2    // Definition of class Cylinder
3    #ifndef CYLINDR2_H
4    #define CYLINDR2_H
5
6    #include <iostream>
7
8    using std::ostream;
9
10   #include "circle2.h"
11
12   class Cylinder : public Circle {
13      friend ostream &operator<<( ostream &, const Cylinder & );
14
15   public:
16      // default constructor
17      Cylinder( double h = 0.0, double r = 0.0,
18                int x = 0, int y = 0 );
19
20      void setHeight( double );   // set height
21      double getHeight() const;   // return height
22      double area() const;        // calculate and return area
23      double volume() const;      // calculate and return volume
24
25   protected:
26      double height;              // height of the Cylinder
27   }; // end class Cylinder
28
29   #endif
```

Outline

cylindr2.h

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



cylindr2.cpp (1 of 2)

```
30  // Fig. 19.10: cylindr2.cpp
31  // Member and friend function definitions
32  // for class Cylinder.
33  #include "cylindr2.h"
34
35  // Cylinder constructor calls Circle constructor
36  Cylinder::Cylinder( double h, double r, int x, int y )
37     : Circle( r, x, y )   // call base-class constructor
38  { setHeight( h ); }
39
40  // Set height of Cylinder
41  void Cylinder::setHeight( double h )
42     { height = ( h >= 0 ? h : 0 ); }
43
44  // Get height of Cylinder
45  double Cylinder::getHeight() const { return height; }
46
47  // Calculate area of Cylinder (i.e., surface area)
48  double Cylinder::area() const
49  {
50     return 2 * Circle::area() +
51            2 * 3.14159 * radius * height;
52  } // end function area
53
54  // Calculate volume of Cylinder
55  double Cylinder::volume() const
56     { return Circle::area() * height; }
57
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Outline

cylindr2.cpp (2 of 2)

```
58   // Output Cylinder dimensions
59   ostream &operator<<( ostream &output, const Cylinder &c )
60   {
61      output << static_cast< Circle >( c )
62             << "; Height = " << c.height;
63
64      return output;   // enables cascaded calls
65   } // end operator<< function
```

fig19_10.cpp (1 of 3)

```
66   // Fig. 19.10: fig19_10.cpp
67   // Driver for class Cylinder
68   #include <iostream>
69
70   using std::cout;
71   using std::endl;
72
73   #include "point2.h"
74   #include "circle2.h"
75   #include "cylindr2.h"
76
77   int main()
78   {
79      // create Cylinder object
80      Cylinder cyl( 5.7, 2.5, 12, 23 );
81
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Outline

fig19_10.cpp (2 of 3)

```
82     // use get functions to display the Cylinder
83     cout << "X coordinate is " << cyl.getX()
84          << "\nY coordinate is " << cyl.getY()
85          << "\nRadius is " << cyl.getRadius()
86          << "\nHeight is " << cyl.getHeight() << "\n\n";
87
88     // use set functions to change the Cylinder's attributes
89     cyl.setHeight( 10 );
90     cyl.setRadius( 4.25 );
91     cyl.setPoint( 2, 2 );
92     cout << "The new location, radius, and height of cyl are:\n"
93          << cyl << '\n';
94
95     cout << "The area of cyl is:\n"
96          << cyl.area() << '\n';
97
98     // display the Cylinder as a Point
99     Point &pRef = cyl;   // pRef "thinks" it is a Point
100    cout << "\nCylinder printed as a Point is: "
101         << pRef << "\n\n";
102
103    // display the Cylinder as a Circle
104    Circle &circleRef = cyl;  // circleRef thinks it is a Circle
105    cout << "Cylinder printed as a Circle is:\n" << circleRef
106         << "\nArea: " << circleRef.area() << endl;
107
108    return 0;
109 } // end function main
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
X coordinate is 12
Y coordinate is 23
Radius is 2.5
Height is 5.7

The new location, radius, and height of cyl are:
Center = [2, 2]; Radius = 4.25; Height = 10.00
The area of cyl is:
380.53

Cylinder printed as a Point is: [2, 2]

Cylinder printed as a Circle is:
Center = [2, 2]; Radius = 4.25
Area: 56.74
```

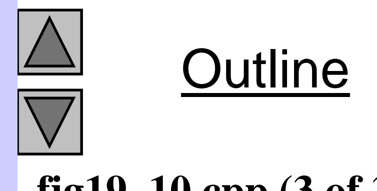


fig19_10.cpp (3 of 3)

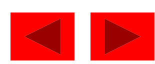
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Chapter 20 - C++ Virtual Functions and Polymorphism

Outline

20.1 Introduction
20.2 Type Fields and `switch` Statements
20.3 Virtual Functions
20.4 Abstract Base Classes and Concrete Classes
20.5 Polymorphism
20.6 New Classes and Dynamic Binding
20.7 Virtual Destructors
20.8 Case Study: Inheriting Interface and Implementation
20.9 Polymorphism, `virtual` Functions and Dynamic Binding "Under the Hood"

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Objectives

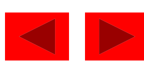
- In this chapter, you will learn:
  - To understand the notion of polymorphism.
  - To understand how to define and use `virtual` functions to effect polymorphism.
  - To understand the distinction between abstract classes and concrete classes.
  - To learn how to define pure `virtual` functions to create abstract classes.
  - To appreciate how polymorphism makes systems extensible and maintainable.
  - To understand how C++ implements `virtual` functions and dynamic binding "under the hood."

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

20.1 Introduction

- `virtual` functions and polymorphism
  - Design and implement systems that are more easily extensible
  - Programs written to generically process objects of all existing classes in a hierarchy

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

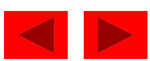


20.2 Type Fields and `switch` Statements

- `switch` statement
  - Take an action on a object based on its type
  - A `switch` structure could determine which `print` function to call based on which type in a hierarchy of `shapes`

- Problems with `switch`
  - Programmer may forget to test all possible cases in a switch.
    - Tracking this down can be time consuming and error prone
    - `virtual` functions and polymorphic programming can eliminate the need for `switch`

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



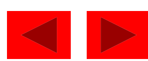
20.3 Virtual Functions

- `virtual` functions
  - Used instead of `switch` statements
  - Definition:
    - Keyword `virtual` before function prototype in base class

```
virtual void draw() const;
```

  - A base-class pointer to a derived class object will call the correct `draw` function
  - If a derived class does not define a `virtual` function it is inherited from the base class

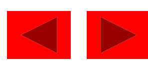
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



20.3 Virtual Functions

- `ShapePtr->Draw();`
  - Compiler implements dynamic binding
  - Function determined during execution time
- `ShapeObject.Draw();`
  - Compiler implements static binding
  - Function determined during compile-time

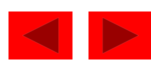
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



20.4 Abstract and Concrete Classes

- Abstract classes
  - Sole purpose is to provide a base class for other classes
  - No objects of an abstract base class can be instantiated
    - Too generic to define real objects, i.e. `TwoDimensionalShape`
    - Can have pointers and references
  - Concrete classes - classes that can instantiate objects
    - Provide specifics to make real objects, i.e. `Square`, `Circle`

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



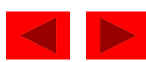
20.4 Abstract and Concrete Classes

- Making abstract classes
  - Define one or more `virtual` functions as “pure” by initializing the function to zero

```
virtual double earnings() const = 0;
```

    - Pure `virtual` function

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

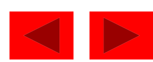


20.5 Polymorphism

- Polymorphism:
  - Ability for objects of different classes to respond differently to the same function call
  - Base-class pointer (or reference) calls a `virtual` function
    - C++ chooses the correct overridden function in object
  - Suppose `print` not a `virtual` function

```
Employee e, *ePtr = &e;
HourlyWorker h, *hPtr = &h;
ePtr->print();    //call base-class print function
hPtr->print();    //call derived-class print function
ePtr=&h;          //allowable implicit conversion
ePtr->print();    // still calls base-class print
```

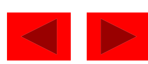
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



20.6 New Classes and Dynamic Binding

- Dynamic binding (late binding)
  - Object's type not needed when compiling `virtual` functions
  - Accommodate new classes that have been added after compilation
  - Important for ISV’s (Independent Software Vendors) who do not wish to reveal source code to their customers

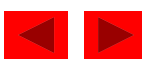
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



20.7 Virtual Destructors

- Problem:
  - If base-class pointer to a derived object is `deleted`, the base-class destructor will act on the object
- Solution:
  - Define a `virtual` base-class destructor
  - Now, the appropriate destructor will be called

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

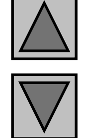


20.8 Case Study: Inheriting Interface and Implementation

- Re-examine the Point, Circle, Cylinder hierarchy
  - Use the abstract base class Shape to head the hierarchy

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
1   // Fig. 20.1: shape.h
2   // Definition of abstract base class Shape
3   #ifndef SHAPE_H
4   #define SHAPE_H
5
6   class Shape {
7   public:
8      virtual double area() const { return 0.0; }
9      virtual double volume() const { return 0.0; }
10
11     // pure virtual functions overridden in derived classes
12     virtual void printShapeName() const = 0;
13     virtual void print() const = 0;
14  }; // end class Shape
15
16  #endif
17 // Fig. 20.1: point1.h
18 // Definition of class Point
19 #ifndef POINT1_H
20 #define POINT1_H
21
22 #include <iostream>
23
24 using std::cout;
25
```



<u>Outline</u>

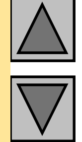
1. Shape **Definition (abstract base** class**)**

----

1. Point **Definition (derived** class**)**

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
26  #include "shape.h"
27
28  class Point : public Shape {
29  public:
30     Point( int = 0, int = 0 );  // default constructor
31     void setPoint( int, int );
32     int getX() const { return x; }
33     int getY() const { return y; }
34     virtual void printShapeName() const { cout << "Point: "; }
35     virtual void print() const;
36  private:
37     int x, y;   // x and y coordinates of Point
38  }; // end class Point
39
40  #endif
41 // Fig. 20.1: point1.cpp
42 // Member function definitions for class Point
43 #include "point1.h"
44
45 Point::Point( int a, int b ) { setPoint( a, b ); }
46
47 void Point::setPoint( int a, int b )
48 {
49    x = a;
50    y = b;
51 } // end function setPoint
52
```



1. `Point` **Definition (derived** `class`**)**

1.1 Function Definitions

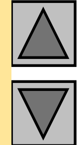
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
53 void Point::print() const
54    { cout << '[' << x << ", " << y << ']'; }
55 // Fig. 20.1: circle1.h
56 // Definition of class Circle
57 #ifndef CIRCLE1_H
58 #define CIRCLE1_H
59 #include "point1.h"
60
61 class Circle : public Point {
62 public:
63    // default constructor
64    Circle( double r = 0.0, int x = 0, int y = 0 );
65
66    void setRadius( double );
67    double getRadius() const;
68    virtual double area() const;
69    virtual void printShapeName() const { cout << "Circle: "; }
70    virtual void print() const;
71 private:
72    double radius;   // radius of Circle
73 }; // end class Circle
74
75 #endif
```

1. Circle Definition (derived class)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
76   // Fig. 20.1: circle1.cpp
77   // Member function definitions for class Circle
78   #include <iostream>
79
80   using std::cout;
81
82   #include "circle1.h"
83
84   Circle::Circle( double r, int a, int b )
85      : Point( a, b )  // call base-class constructor
86   { setRadius( r ); }
87
88   void Circle::setRadius( double r ) { radius = r > 0 ? r : 0; }
89
90   double Circle::getRadius() const { return radius; }
91
92   double Circle::area() const
93      { return 3.14159 * radius * radius; }
94
95   void Circle::print() const
96   {
97      Point::print();
98      cout << "; Radius = " << radius;
99   } // end function print
```

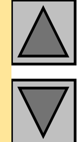


Outline

1.1 Function Definitions

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
100 // Fig. 20.1: cylindr1.h
101 // Definition of class Cylinder
102 #ifndef CYLINDR1_H
103 #define CYLINDR1_H
104 #include "circle1.h"
105
106 class Cylinder : public Circle {
107 public:
108    // default constructor
109    Cylinder( double h = 0.0, double r = 0.0,
110              int x = 0, int y = 0 );
111
112    void setHeight( double );
113    double getHeight();
114    virtual double area() const;
115    virtual double volume() const;
116    virtual void printShapeName() const { cout << "Cylinder: "; }
117    virtual void print() const;
118 private:
119    double height;   // height of Cylinder
120 }; // end class Cylinder
121
122 #endif
```

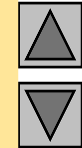


Outline

1. Cylinder **Definition (derived** class**)**

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
123 // Fig. 20.1: cylindr1.cpp
124 // Member and friend function definitions for class Cylinder
125 #include <iostream>
126
127 using std::cout;
128
129 #include "cylindr1.h"
130
131 Cylinder::Cylinder( double h, double r, int x, int y )
132    : Circle( r, x, y )  // call base-class constructor
133 { setHeight( h ); }
134
135 void Cylinder::setHeight( double h )
136    { height = h > 0 ? h : 0; }
137
138 double Cylinder::getHeight() { return height; }
139
140 double Cylinder::area() const
141 {
142    // surface area of Cylinder
143    return 2 * Circle::area() +
144           2 * 3.14159 * getRadius() * height;
145 } // end function area
146
```



1.1 Function Definitions

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
147 double Cylinder::volume() const
148    { return Circle::area() * height; }
149
150 void Cylinder::print() const
151 {
152    Circle::print();
153    cout << "; Height = " << height;
154 } // end function print
155 // Fig. 20.1: fig20_01.cpp
156 // Driver for shape, point, circle, cylinder hierarchy
157 #include <iostream>
158
159 using std::cout;
160 using std::endl;
161
162 #include <iomanip>
163
164 using std::ios;
165 using std::setiosflags;
166 using std::setprecision;
167
168 #include "shape.h"
169 #include "point1.h"
170 #include "circle1.h"
171 #include "cylindr1.h"
172
```

Outline

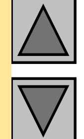
Driver

1. Load headers

1.1 Function prototypes

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
173 void virtualViaPointer( const Shape * );
174 void virtualViaReference( const Shape & );
175
176 int main()
177 {
178    cout << setiosflags( ios::fixed | ios::showpoint )
179         << setprecision( 2 );
180
181    Point point( 7, 11 );                  // create a Point
182    Circle circle( 3.5, 22, 8 );           // create a Circle
183    Cylinder cylinder( 10, 3.3, 10, 10 );  // create a Cylinder
184
185    point.printShapeName();    // static binding
186    point.print();             // static binding
187    cout << '\n';
188
189    circle.printShapeName();   // static binding
190    circle.print();            // static binding
191    cout << '\n';
192
193    cylinder.printShapeName(); // static binding
194    cylinder.print();          // static binding
195    cout << "\n\n";
196
197    Shape *arrayOfShapes[ 3 ];  // array of base-class pointers
198
```



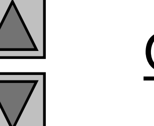
Outline

1.2 Initialize objects

2. Function calls

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
199    // aim arrayOfShapes[0] at derived-class Point object
200    arrayOfShapes[ 0 ] = &point;
201
202    // aim arrayOfShapes[1] at derived-class Circle object
203    arrayOfShapes[ 1 ] = &circle;
204
205    // aim arrayOfShapes[2] at derived-class Cylinder object
206    arrayOfShapes[ 2 ] = &cylinder;
207
208    // Loop through arrayOfShapes and call virtualViaPointer
209    // to print the shape name, attributes, area, and volume
210    // of each object using dynamic binding.
211    cout << "Virtual function calls made off "
212         << "base-class pointers\n";
213
214    for ( int i = 0; i < 3; i++ )
215       virtualViaPointer( arrayOfShapes[ i ] );
216
217    // Loop through arrayOfShapes and call virtualViaReference
218    // to print the shape name, attributes, area, and volume
219    // of each object using dynamic binding.
220    cout << "Virtual function calls made off "
221         << "base-class references\n";
222
```



Outline

2. Function calls

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
223    for ( int j = 0; j < 3; j++ )
224       virtualViaReference( *arrayOfShapes[ j ] );
225
226    return 0;
227 } // end function main
228
229 // Make virtual function calls off a base-class pointer
230 // using dynamic binding.
231 void virtualViaPointer( const Shape *baseClassPtr )
232 {
233    baseClassPtr->printShapeName();
234    baseClassPtr->print();
235    cout << "\nArea = " << baseClassPtr->area()
236         << "\nVolume = " << baseClassPtr->volume() << "\n\n";
237 } // end function virtualViaPointer
238
239 // Make virtual function calls off a base-class reference
240 // using dynamic binding.
241 void virtualViaReference( const Shape &baseClassRef )
242 {
243    baseClassRef.printShapeName();
244    baseClassRef.print();
245    cout << "\nArea = " << baseClassRef.area()
246         << "\nVolume = " << baseClassRef.volume() << "\n\n";
247 } // end function virtualViaReference
```

3. Function Definitions

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
Point: [7, 11]
Circle: [22, 8]; Radius = 3.50
Cylinder: [10, 10]; Radius = 3.30; Height = 10.00

Virtual function calls made off base-class pointers
Point: [7, 11]
Area = 0.00
Volume = 0.00

Circle: [22, 8]; Radius = 3.50
Area = 38.48
Volume = 0.00

Cylinder: [10, 10]; Radius = 3.30; Height = 10.00
Area = 275.77
Volume = 342.12

Virtual function calls made off base-class references
Point: [7, 11]
Area = 0.00
Volume = 0.00

Circle: [22, 8]; Radius = 3.50
Area = 38.48
Volume = 0.00

Cylinder: [10, 10]; Radius = 3.30; Height = 10.00
Area = 275.77
Volume = 342.12
```


Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

20.9 Polymorphism, `virtual` Functions and Dynamic Binding "Under the Hood"

- When to use polymorphism
  - Polymorphism has a lot of overhead
- `virtual` function table (vtable)
  - Every class with a `virtual` function has a vtable
  - For every `virtual` function, vtable has a pointer to the proper function
    - If a derived class has the same function as a base class, then the function pointer points to the base-class function
  - Detailed explanation in Fig. 20.2

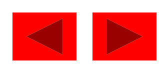
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Chapter 21 - C++ Stream Input/Output

Outline

21.1 Introduction
21.2 Streams
21.2.1 Iostream Library Header Files
21.2.2 Stream Input/Output Classes and Objects
21.3 Stream Output
21.3.1 Stream-Insertion Operator
21.3.2 Cascading Stream-Insertion/Extraction Operators
21.3.3 Output of `char *` Variables
21.3.4 Character Output with Member Function `put`; Cascading `puts`
21.4 Stream Input
21.4.1 Stream-Extraction Operator
21.4.2 `get` and `getline` Member Functions
21.4.3 `istream` Member Functions `peek`, `putback` and `ignore`
21.4.4 Type-Safe I/O
21.5 Unformatted I/O with `read`, `gcount` and `write`

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

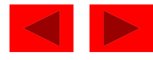


Chapter 21 - C++ Stream Input/Output

Outline (continued)

21.6 Stream Manipulators
21.6.1 Integral Stream Base: `dec`, `oct`, `hex` **and** `setbase`
21.6.2 Floating-Point Precision (`precision`, `setprecision`**)**
21.6.3 Field Width (`setw`, `width`**)**
21.6.4 User-Defined Manipulators
21.7 Stream Format States
21.7.1 Format State Flags
21.7.2 Trailing Zeros and Decimal Points (`ios::showpoint`**)**
21.7.3 Justification (`ios::left`, `ios::right`, `ios::internal`**)**
21.7.4 Padding (`fill`, `setfill`**)**
21.7.5 Integral Stream Base (`ios::dec`, `ios::oct`, `ios::hex`, `ios::showbase`**)**
21.7.6 Floating-Point Numbers; Scientific Notation (`ios::scientific`, `ios::fixed`**)**
21.7.7 Uppercase/Lowercase Control (`ios::uppercase`**)**
21.7.8 Setting and Resetting the Format Flags (`flags`, `setiosflags`, `resetiosflags`**)**
21.8 Stream Error States
21.9 Tying an Output Stream to an Input Stream

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Objectives

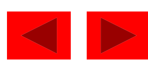
- In this chapter, you will learn:
  - To understand how to use C++ object-oriented stream input/output.
  - To be able to format inputs and outputs.
  - To understand the stream I/O class hierarchy.
  - To understand how to input/output objects of user-defined types.
  - To be able to create user-defined stream manipulators.
  - To be able to determine the success or failure of input/output operations.
  - To be able to tie output streams to input streams.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

21.1 Introduction

- Many C++ I/O features are object-oriented
  - Use references, function overloading and operator overloading
- C++ uses type safe I/O
  - Each I/O operation is automatically performed in a manner sensitive to the data type
- Extensibility
  - Users may specify I/O of user-defined types as well as standard types

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

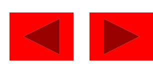


21.2 Streams

- Stream
  - A transfer of information in the form of a sequence of bytes

- I/O Operations:
  - Input: A stream that flows from an input device (i.e.: keyboard, disk drive, network connection) to main memory
  - Output: A stream that flows from main memory to an output device (i.e.: screen, printer, disk drive, network connection)

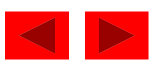
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



21.2 Streams

- I/O operations are a bottleneck
  - The time for a stream to flow is many times larger than the time it takes the CPU to process the data in the stream
- Low-level I/O
  - Unformatted
  - Individual byte unit of interest
  - High speed, high volume, but inconvenient for people
- High-level I/O
  - Formatted
  - Bytes grouped into meaningful units: integers, characters, etc.
  - Good for all I/O except high-volume file processing

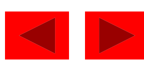
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



21.2.1 Iostream Library Header Files

- `iostream` library:
  - `<iostream.h>`: Contains `cin`, `cout`, `cerr` and `clog` objects
  - `<iomanip.h>`: Contains *parameterized stream manipulators*

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



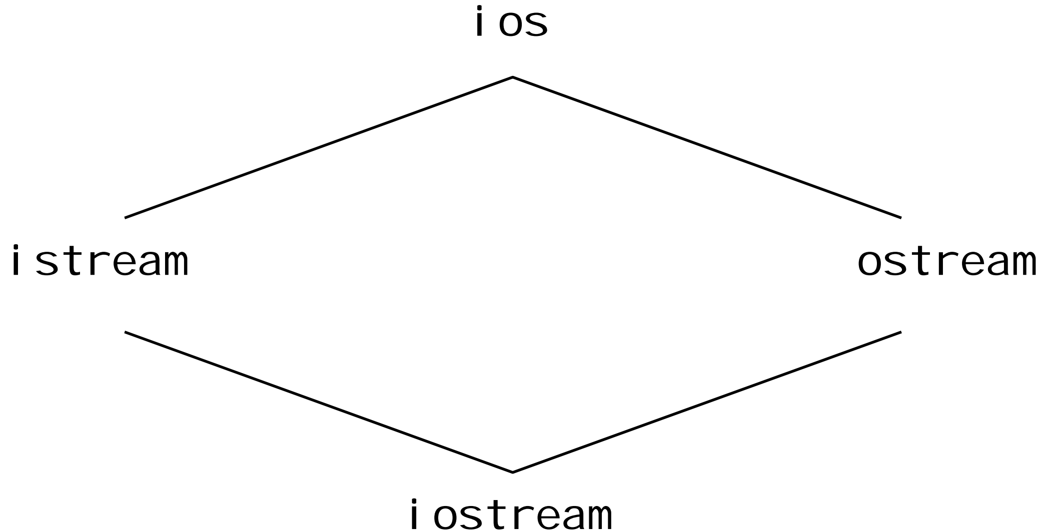
21.2.2 Stream Input/Output Classes and Objects

- `ios`:
  - `istream` and `ostream` inherit from `ios`
    - `iostream` inherits from `istream` and `ostream`.
- `<<` (left-shift operator)
  - Overloaded as *stream insertion operator*
- `>>` (right-shift operator)
  - Overloaded as *stream extraction operator*
  - Both operators used with `cin`, `cout`, `cerr`, `clog`, and with user-defined stream objects

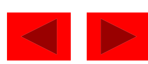
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

21.2.2 Stream Input/Output Classes and Objects

Figure 21.1 Portion of the stream I/O class hierarchy.



© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



21.2.2 Stream Input/Output Classes and Objects

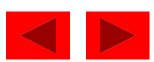
- `istream`: input streams

  `cin >> grade;`

  - `cin` knows what type of data is to be assigned to `grade` (based on the type of `grade`).

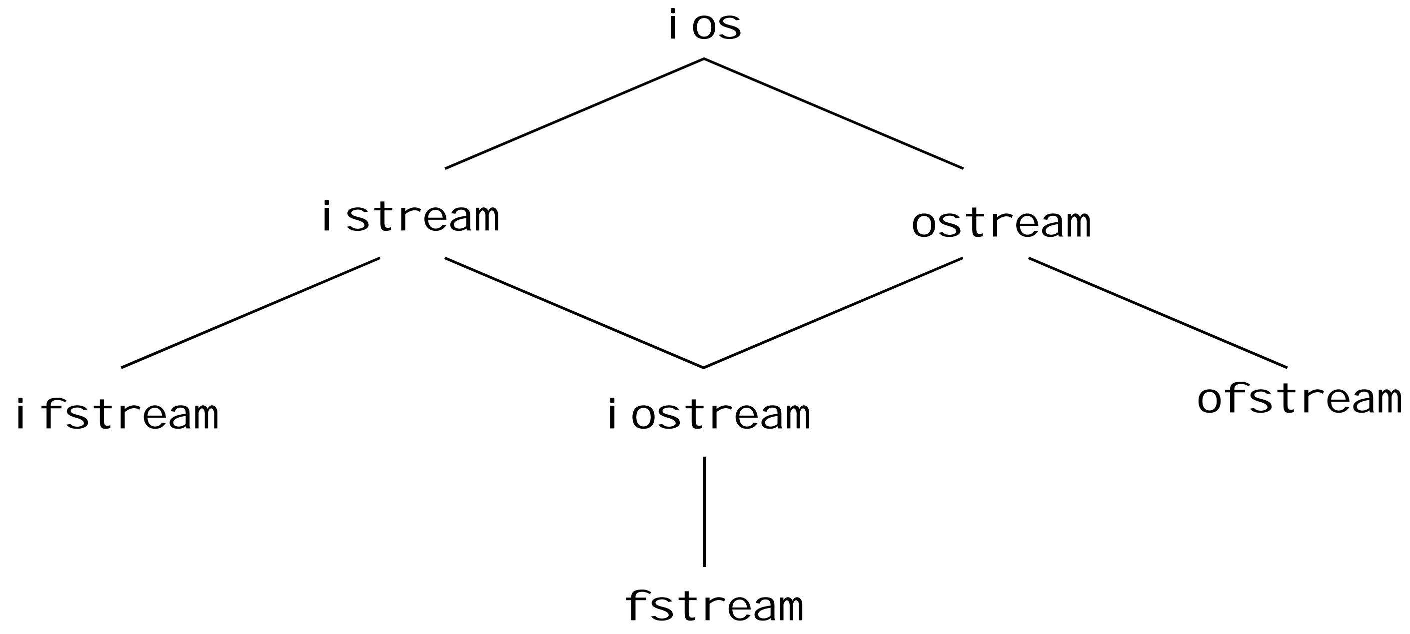
- `ostream`: output streams
  - `cout << grade;`
    - **`cout`** knows the type of data to output
  - `cerr << errorMessage;`
    - Unbuffered - prints `errorMessage` immediately.
  - `clog << errorMessage;`
    - Buffered - prints `errorMessage` as soon as output buffer is full or flushed

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

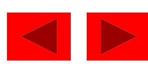


21.2.2 Stream Input/Output Classes and Objects

Figure 21.2 Portion of stream-I/O class hierarchy with key file-processing classes.



© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



21.3 Stream Output

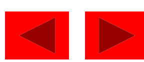
- `ostream`: performs formatted and unformatted output
  - Uses `put` for characters and `write` for unformatted output
  - Output of integers in decimal, octal and hexadecimal
  - Varying precision for floating points
  - Formatted text outputs

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

21.3.1 Stream-Insertion Operator

- << is overloaded to output built-in types
  - Can also be used to output user-defined types
  - `cout << '\n';`
    - Prints newline character
  - `cout << endl;`
    - `endl` is a stream manipulator that issues a newline character and flushes the output buffer
  - `cout << flush;`
    - `flush` flushes the output buffer

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



```
1  // Fig. 21.3: fig21_03.cpp
2  // Outputting a string using stream insertion.
3  #include <iostream>
4
5  using std::cout;
6
7  int main()
8  {
9     cout << "Welcome to C++!\n";
10
11     return 0;
12 } // end function main
```

```
Welcome to C++!
```

Outline

fig21_03.cpp

Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
1  // Fig. 21.4: fig21_04.cpp
2  // Outputting a string using two stream insertions.
3  #include <iostream>
4
5  using std::cout;
6
7  int main()
8  {
9     cout << "Welcome to ";
10    cout << "C++!\n";
11
12    return 0;
13 } // end function main
```

```
Welcome to C++!
```

Outline

fig21_04.cpp

Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
1  // Fig. 21.5: fig21_05.cpp
2  // Using the endl stream manipulator.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  int main()
9  {
10    cout << "Welcome to ";
11    cout << "C++!";
12    cout << endl;   // end line stream manipulator
13
14    return 0;
15 } // end function main
```

```
Welcome to C++!
```

Outline

fig21_05.cpp

Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
1  // Fig. 21.6: fig21_06.cpp
2  // Outputting expression values.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  int main()
9  {
10    cout << "47 plus 53 is ";
11
12    // parentheses not needed; used for clarity
13    cout << ( 47 + 53 );       // expression
14    cout << endl;
15
16    return 0;
17 } // end function main
```

```
47 plus 53 is 100
```

Outline

fig21_06.cpp

Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

21.3.2 Cascading Stream-Insertion/Extraction Operators

- `<<` : Associates from left to right, and returns a reference to its left-operand object (i.e. `cout`).
  - This enables cascading

```
cout << "How" << " are" << " you?";
```

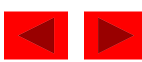
Make sure to use parenthesis:

```
cout << "1 + 2 = " << (1 + 2);
```

NOT

```
cout << "1 + 2 = " << 1 + 2;
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Outline

fig21_07.cpp

```
1  // Fig. 21.7: fig21_07.cpp
2  // Cascading the overloaded << operator.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  int main()
9  {
10    cout << "47 plus 53 is " << ( 47 + 53 ) << endl;
11
12    return 0;
13 } // end function main
```

Program Output

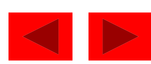
```
47 plus 53 is 100
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

21.3.3 Output of `char *` Variables

- `<<` will output a variable of type `char *` as a string
- To output the address of the first character of that string, cast the variable as type `void *`

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



```
1  // Fig. 21.8: fig21_08.cpp
2  // Printing the address stored in a char* variable
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  int main()
9  {
10    const char *string = "test";
11
12    cout << "Value of string is: " << string
13         << "\nValue of static_cast< void * >( string ) is: "
14         << static_cast< void * >( string ) << endl;
15    return 0;
16 } // end function main
```

```
Value of string is: test
Value of static_cast< void *>( string ) is: 0046C070
```

Outline

fig21_08.cpp

Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

21.3.4 Character Output with Member Function `put`; Cascading `put`s

- `put` member function
  - Outputs one character to specified stream

    ```
    cout.put( 'A');
    ```
  - Returns a reference to the object that called it, so may be cascaded

    ```
    cout.put( 'A' ).put( '\n' );
    ```
  - May be called with an ASCII-valued expression

    ```
    cout.put( 65 );
    ```
    - Outputs `A`

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

21.4 Stream Input

- >> (stream-extraction)
  - Used to perform stream input
  - Normally ignores whitespaces (spaces, tabs, newlines)
  - Returns zero (`false`) when `EOF` is encountered, otherwise returns reference to the object from which it was invoked (i.e. `cin`)

- >> controls the state bits of the stream
  - `failbit` set if wrong type of data input
  - `badbit` set if the operation fails

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

21.4.1 Stream-Extraction Operator

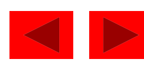
- `>>` and `<<` have relatively high precedence
  - Conditional and arithmetic expressions must be contained in parentheses

- Popular way to perform loops

```
while (cin >> grade)
```

  - Extraction returns `0` (`false`) when `EOF` encountered, and loop ends

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Outline

fig21_09.cpp

```
1  // Fig. 21.9: fig21_09.cpp
2  // Calculating the sum of two integers input from the keyboard
3  // with cin and the stream-extraction operator.
4  #include <iostream>
5
6  using std::cout;
7  using std::cin;
8  using std::endl;
9
10 int main()
11 {
12    int x, y;
13
14    cout << "Enter two integers: ";
15    cin >> x >> y;
16    cout << "Sum of " << x << " and " << y << " is: "
17         << ( x + y ) << endl;
18
19    return 0;
20 } // end function main
```

```
Enter two integers: 30 92
Sum of 30 and 92 is: 122
```

Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Outline

fig21_10.cpp

```
1  // Fig. 21.10: fig21_10.cpp
2  // Avoiding a precedence problem between the stream-insertion
3  // operator and the conditional operator.
4  // Need parentheses around the conditional expression.
5  #include <iostream>
6
7  using std::cout;
8  using std::cin;
9  using std::endl;
10
11 int main()
12 {
13    int x, y;
14
15    cout << "Enter two integers: ";
16    cin >> x >> y;
17    cout << x << ( x == y ? " is" : " is not" )
18         << " equal to " << y << endl;
19
20    return 0;
21 } // end function main
```

Program Output

```
Enter two integers: 7 5
7 is not equal to 5
```

```
Enter two integers: 8 8
8 is equal to 8
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

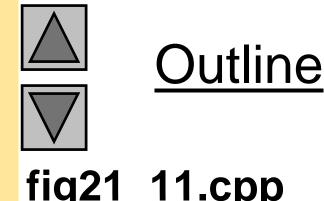
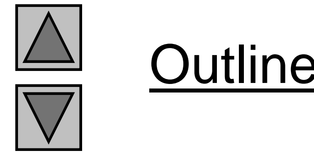


fig21_11.cpp

```
1  // Fig. 21.11: fig21_11.cpp
2  // Stream-extraction operator returning false on end-of-file.
3  #include <iostream>
4
5  using std::cout;
6  using std::cin;
7  using std::endl;
8
9  int main()
10 {
11    int grade, highestGrade = -1;
12
13    cout << "Enter grade (enter end-of-file to end): ";
14    while ( cin >> grade ) {
15       if ( grade > highestGrade )
16          highestGrade = grade;
17
18       cout << "Enter grade (enter end-of-file to end): ";
19    } // end while
20
21    cout << "\n\nHighest grade is: " << highestGrade << endl;
22    return 0;
23 } // end function main
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
Enter grade (enter end-of-file to end): 67
Enter grade (enter end-of-file to end): 87
Enter grade (enter end-of-file to end): 73
Enter grade (enter end-of-file to end): 95
Enter grade (enter end-of-file to end): 34
Enter grade (enter end-of-file to end): 99
Enter grade (enter end-of-file to end): ^Z
Highest grade is: 99
```



Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

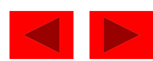
21.4.2 `get` and `getline` Member Functions

- `cin.eof()`: returns `true` if end-of-file has occurred on `cin`

- `cin.get()`: inputs a character from stream (even white spaces) and returns it

- `cin.get( c )`: inputs a character from stream and stores it in `c`

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



```
1  // Fig. 21.12: fig21_12.cpp
2  // Using member functions get, put and eof.
3  #include <iostream>
4
5  using std::cout;
6  using std::cin;
7  using std::endl;
8
9  int main()
10 {
11    char c;
12
13    cout << "Before input, cin.eof() is " << cin.eof()
14         << "\nEnter a sentence followed by end-of-file:\n";
15
16    while ( ( c = cin.get() ) != EOF )
17       cout.put( c );
18
19    cout << "\nEOF in this system is: " << c;
20    cout << "\nAfter input, cin.eof() is " << cin.eof() << endl;
21    return 0;
22 } // end function main
```

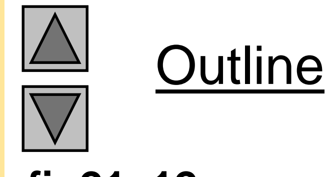


fig21_12.cpp

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
Before input, cin.eof() is 0
Enter a sentence followed by end-of-file:
Testing the get and put member functions
Testing the get and put member functions
^Z

EOF in this system is: -1
After input cin.eof() is 1
```

Program Output

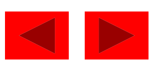
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

21.4.2 `get` and `getline` Member Functions

- `cin.get(array, size)`:
  - Accepts 3 arguments: array of characters, the size limit, and a delimiter (default of `'\n'`).
  - Uses the array as a buffer
  - When the delimiter is encountered, it remains in the input stream
  - Null character is inserted in the array
  - Unless delimiter flushed from stream, it will stay there

- `cin.getline(array, size)`
  - Operates like `cin.get(buffer, size)` but it discards the delimiter from the stream and does not store it in array
  - Null character inserted into array

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



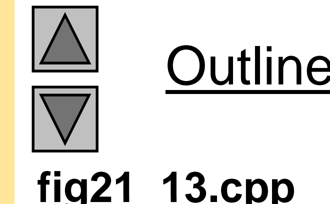
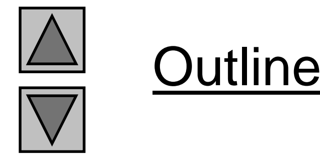


fig21_13.cpp

```
1 // Fig. 21.13: fig21_13.cpp
2 // Contrasting input of a string with cin and cin.get.
3 #include <iostream>
4
5 using std::cout;
6 using std::cin;
7 using std::endl;
8
9 int main()
10 {
11    const int SIZE = 80;
12    char buffer1[ SIZE ], buffer2[ SIZE ];
13
14    cout << "Enter a sentence:\n";
15    cin >> buffer1;
16    cout << "\nThe string read with cin was:\n"
17         << buffer1 << "\n\n";
18
19    cin.get( buffer2, SIZE );
20    cout << "The string read with cin.get was:\n"
21         << buffer2 << endl;
22
23    return 0;
24 } // end function main
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
Enter a sentence:
Contrasting string input with cin and cin.get

The string read with cin was:
Contrasting

The string read with cin.get was:
 string input with cin and cin.get
```



Outline

Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Outline

fig21_14.cpp

```
1  // Fig. 21.14: fig21_14.cpp
2  // Character input with member function getline.
3  #include <iostream>
4
5  using std::cout;
6  using std::cin;
7  using std::endl;
8
9  int main()
10 {
11    const SIZE = 80;
12    char buffer[ SIZE ];
13
14    cout << "Enter a sentence:\n";
15    cin.getline( buffer, SIZE );
16
17    cout << "\nThe sentence entered is:\n" << buffer << endl;
18    return 0;
19 } // end function main
```

Program Output

```
Enter a sentence:
Using the getline member function

The sentence entered is:
Using the getline member function
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

21.4.3 `istream` Member Functions `peek`, `putback` and `ignore`

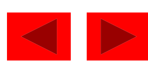
- `ignore` member function
  - Skips over a designated number of characters (default of one)
  - Terminates upon encountering a designated delimiter (default is `EOF`, skips to the end of the file)
- `putback` member function
  - Places the previous character obtained by `get` back in to the stream.
- `peek`
  - Returns the next character from the stream without removing it

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

21.4.4 Type-Safe I/O

- << and >> operators
  - Overloaded to accept data of different types
  - When unexpected data encountered, error flags set
  - Program stays in control

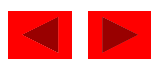
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



21.5 Unformatted I/O with read, gcount and write

- read and write member functions
  - Unformatted I/O
  - Input/output raw bytes to or from a character array in memory
  - Since the data is unformatted, the functions will not terminate at a newline character for example
    - Instead, like getline, they continue to process a designated number of characters
  - If fewer than the designated number of characters are read, then the failbit is set
- gcount:
  - Returns the total number of characters read in the last input operation

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Outline

fig21_15.cpp

```
1  // Fig. 21.15: fig21_15.cpp
2  // Unformatted I/O with read, gcount and write.
3  #include <iostream>
4
5  using std::cout;
6  using std::cin;
7  using std::endl;
8
9  int main()
10 {
11    const int SIZE = 80;
12    char buffer[ SIZE ];
13
14    cout << "Enter a sentence:\n";
15    cin.read( buffer, 20 );
16    cout << "\nThe sentence entered was:\n";
17    cout.write( buffer, cin.gcount() );
18    cout << endl;
19    return 0;
20 } // end function main
```

Program Output

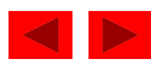
```
Enter a sentence:
Using the read, write and gcount member functions
The sentence entered was:
Using the read, writ
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

21.6 Stream Manipulators

- Stream manipulator capabilities
  - Setting field widths
  - Setting precisions
  - Setting and unsetting format flags
  - Setting the fill character in fields
  - Flushing streams
  - Inserting a newline in the output stream and flushing the stream
  - Inserting a null character in the output stream and skipping whitespace in the input stream

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



21.6.1 Integral Stream Base: dec, oct, hex and setbase

- oct, hex or dec:
  - Change base of which integers are interpreted from the stream.

  Example:

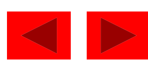
  ```
  int n = 15;
   cout << hex << n;
  ```

  - Prints "F"
- setbase:
  - Changes base of integer output
  - Load <iomanip>
  - Accepts an integer argument (10, 8, or 16)

    ```
    cout << setbase(16) << n;
    ```

  - Parameterized stream manipulator - takes an argument

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



```
1  // Fig. 21.16: fig21_16.cpp
2  // Using hex, oct, dec and setbase stream manipulators.
3  #include <iostream>
4
5  using std::cout;
6  using std::cin;
7  using std::endl;
8
9  #include <iomanip>
10
11 using std::hex;
12 using std::dec;
13 using std::oct;
14 using std::setbase;
15
16 int main()
17 {
18    int n;
19
20    cout << "Enter a decimal number: ";
21    cin >> n;
22
```

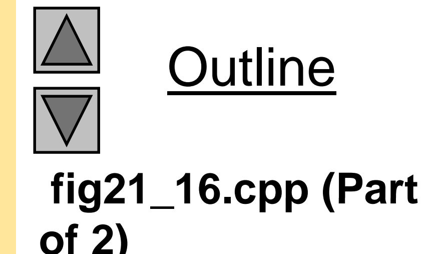


fig21_16.cpp (Part 1 of 2)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
23    cout << n << " in hexadecimal is: "
24         << hex << n << '\n'
25         << dec << n << " in octal is: "
26         << oct << n << '\n'
27         << setbase( 10 ) << n << " in decimal is: "
28         << n << endl;
29
30    return 0;
31 } // end function main
```

```
Enter a decimal number: 20
20 in hexadecimal is: 14
20 in octal is: 24
20 in decimal is: 20
```


fig21_16.cpp (Part 2 of 2)

Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

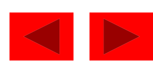
21.6.2 Floating-Point Precision (precision, setprecision)

- `precision`
  - Member function
  - Sets number of digits to the right of decimal point
    `cout.precision(2);`
  - `cout.precision()` returns current precision setting
- setprecision
  - Parameterized stream manipulator
  - Like all parameterized stream manipulators, `<iomanip>` required
  - Specify precision:

```
cout << setprecision(2) << x;
```

- For both methods, changes last until a different value is set

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



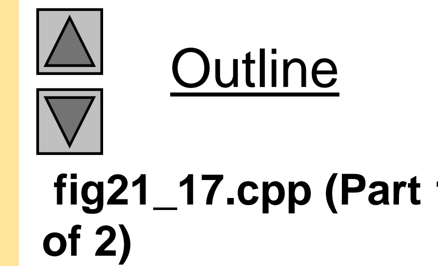


fig21_17.cpp (Part 1 of 2)

```
1  // Fig. 21.17: fig21_17.cpp
2  // Controlling precision of floating-point values
3  #include <iostream>
4
5  using std::cout;
6  using std::cin;
7  using std::endl;
8
9  #include <iomanip>
10
11 using std::ios;
12 using std::setiosflags;
13 using std::setprecision;
14
15 #include <cmath>
16
17 int main()
18 {
19    double root2 = sqrt( 2.0 );
20    int places;
21
22    cout << setiosflags( ios::fixed )
23         << "Square root of 2 with precisions 0-9.\n"
24         << "Precision set by the "
25         << "precision member function:" << endl;
26
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
27    for ( places = 0; places <= 9; places++ ) {
28       cout.precision( places );
29       cout << root2 << '\n';
30    } // end for
31
32    cout << "\nPrecision set by the "
33         << "setprecision manipulator:\n";
34
35    for ( places = 0; places <= 9; places++ )
36       cout << setprecision( places ) << root2 << '\n';
37
38    return 0;
39 } // end function main
```

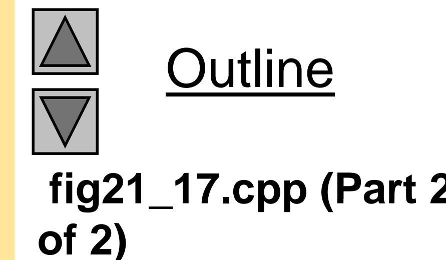


fig21_17.cpp (Part 2 of 2)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
Square root of 2 with precisions 0-9.
Precision set by the precision member function:
1
1.4
1.41
1.414
1.4142
1.41421
1.414214
1.4142136
1.41421356
1.414213562

Precision set by the setprecision manipulator:
1
1.4
1.41
1.414
1.4142
1.41421
1.414214
1.4142136
1.41421356
1.414213562
```

Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

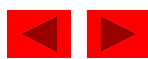
21.6.3 Field Width(`setw`, `width`)

- `ios width` member function
  - Sets field width (number of character positions a value should be output or number of characters that should be input)
  - Returns previous width
  - If values processed are smaller than width, fill characters inserted as padding
  - Values are not truncated - full number printed
  - `cin.width(5);`
- `setw` stream manipulator

```
cin >> setw(5) >> string;
```

- Remember to reserve one space for the null character

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



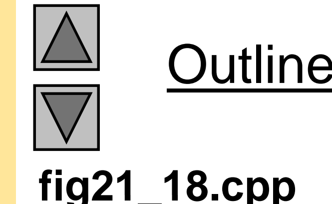
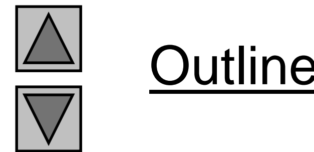


fig21_18.cpp

```
1  // fig21_18.cpp
2  // Demonstrating the width member function
3  #include <iostream>
4
5  using std::cout;
6  using std::cin;
7  using std::endl;
8
9  int main()
10 {
11    int w = 4;
12    char string[ 10 ];
13
14    cout << "Enter a sentence:\n";
15    cin.width( 5 );
16
17    while ( cin >> string ) {
18       cout.width( w++ );
19       cout << string << endl;
20       cin.width( 5 );
21    } // end while
22
23    return 0;
24 } // end function main
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Program Output

```
Enter a sentence:
This is a test of the width member function
This
   is
     a
   test
      of
      the
      widt
          h
        memb
           er
          func
           tion
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

21.6.4 User-Defined Manipulators

- We can create our own stream manipulators
  - `bell`
  - `ret` (carriage return)
  - `tab`
  - `endLine`

- Parameterized stream manipulators
  - Consult installation manuals

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
1  // Fig. 21.19: fig21_19.cpp
2  // Creating and testing user-defined, nonparameterized
3  // stream manipulators.
4  #include <iostream>
5
6  using std::ostream;
7  using std::cout;
8  using std::flush;
9
10 // bell manipulator (using escape sequence \a)
11 ostream& bell( ostream& output ) { return output << '\a'; }
12
13 // ret manipulator (using escape sequence \r)
14 ostream& ret( ostream& output ) { return output << '\r'; }
15
16 // tab manipulator (using escape sequence \t)
17 ostream& tab( ostream& output ) { return output << '\t'; }
18
19 // endLine manipulator (using escape sequence \n
20 // and the flush member function)
21 ostream& endLine( ostream& output )
22 {
23    return output << '\n' << flush;
24 } // end function endLine
25
```

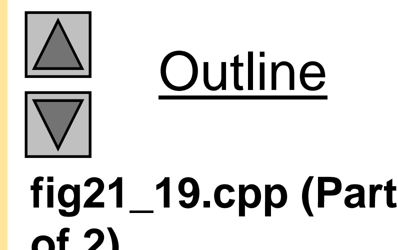


fig21_19.cpp (Part 1 of 2)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
26 int main()
27 {
28    cout << "Testing the tab manipulator:" << endLine
29         << 'a' << tab << 'b' << tab << 'c' << endLine
30         << "Testing the ret and bell manipulators:"
31         << endLine << "..........";
32    cout << bell;
33    cout << ret << "-----" << endLine;
34    return 0;
35 } // end function main
```

```
Testing the tab manipulator:
a       b       c
Testing the ret and bell manipulators:
-----.....
```

Outline

fig21_19.cpp (Part 2 of 2)

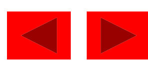
Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

21.7 Stream Format States

- Format flags
  - Specify formatting to be performed during stream I/O operations
- `setf`, `unsetf` and `flags`
  - Member functions that control the flag settings

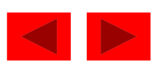
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



21.7.1 Format State Flags

- Format State Flags
  - Defined as an enumeration in class `ios`
  - Can be controlled by member functions
  - `flags` - specifies a value representing the settings of all the flags
    - Returns `long` value containing prior options
  - `setf` - one argument, "ors" flags with existing flags
  - `unsetf` - unsets flags
  - `setiosflags` - parameterized stream manipulator used to set flags
  - `resetiosflags` - parameterized stream manipulator, has same functions as `unsetf`
- Flags can be combined using bitwise OR (|)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

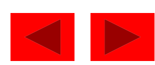


21.7.1 Format State Flags

Format state flag	Description
`ios::skipws`	Skip whitespace characters on an input stream.
`ios::left`	Left-justify output in a field. Padding characters appear to the right if necessary.
`ios::right`	Right-justify output in a field. Padding characters appear to the left if necessary.
`ios::internal`	Indicate that a number's sign should be left-justified in a field and a number's magnitude should be right-justified in that same field (i.e., padding characters appear between the sign and the number).
`ios::dec`	Specify that integers should be treated as decimal (base 10) values.
`ios::oct`	Specify that integers should be treated as octal (base 8) values.
`ios::hex`	Specify that integers should be treated as hexadecimal (base 16) values.
`ios::showbase`	Specify that the base of a number is to be output ahead of the number (a leading `0` for octals; a leading `0x` or `0X` for hexadecimals).
`ios::showpoint`	Specify that floating-point numbers should be output with a decimal point. This is normally used with `ios::fixed` to guarantee a certain number of digits to the right of the decimal point.
`ios::uppercase`	Specify that uppercase letters (i.e., `X` and `A` through `F`) should be used in the hexadecimal integer and that uppercase `E` should be used when representing a floating-point value in scientific notation.
`ios::showpos`	Specify that positive and negative numbers should be preceded by a `+` or `–` sign, respectively.
`ios::scientific`	Specify output of a floating-point value in scientific notation.
`ios::fixed`	Specify output of a floating-point value in fixed-point notation with a specific number of digits to the right of the decimal point.

Fig. 21.20 Format state flags.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



21.7.2 Trailing Zeros and Decimal Points (ios::showpoint)

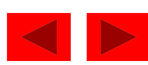
- ios::showpoint
  - Forces a float with an integer value to be printed with its decimal point and trailing zeros

```
cout.setf(ios::showpoint)
cout << 79;
```

79 will print as 79.00000

- Number of zeros determined by precision settings

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Outline

fig21_21.cpp

```
1  // Fig. 21.21: fig21_21.cpp
2  // Controlling the printing of trailing zeros and decimal
3  // points for floating-point values.
4  #include <iostream>
5
6  using std::cout;
7  using std::endl;
8
9  #include <iomanip>
10
11 using std::ios;
12
13 #include <cmath>
14
15 int main()
16 {
17    cout << "Before setting the ios::showpoint flag\n"
18         << "9.9900 prints as: " << 9.9900
19         << "\n9.9000 prints as: " << 9.9000
20         << "\n9.0000 prints as: " << 9.0000
21         << "\n\nAfter setting the ios::showpoint flag\n";
22    cout.setf( ios::showpoint );
23    cout << "9.9900 prints as: " << 9.9900
24         << "\n9.9000 prints as: " << 9.9000
25         << "\n9.0000 prints as: " << 9.0000 << endl;
26    return 0;
27 } // end function main
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
Before setting the ios::showpoint flag
9.9900 prints as: 9.99
9.9000 prints as: 9.9
9.0000 prints as: 9

After setting the ios::showpoint flag
9.9900 prints as: 9.99000
9.9000 prints as: 9.90000
9.0000 prints as: 9.00000
```

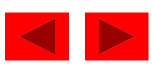

Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

21.7.3 Justification (`ios::left`, `ios::right`, `ios::internal`)

- `ios::left`
  - Fields to left-justified with padding characters to the right
- `ios::right`
  - Default setting
  - Fields right-justified with padding characters to the left
- Character used for padding set by
  - `fill` member function
  - `setfill` parameterized stream manipulator
  - Default character is space

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

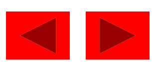


21.7.3 Justification (`ios::left`, `ios::right`, `ios::internal`)

- `internal` flag
  - Number's sign left-justified
  - Number's magnitude right-justified
  - Intervening spaces padded with the fill character
- `static` data member `ios::adjustfield`
  - Contains `left`, `right` and `internal` flags
  - `ios::adjustfield` must be the second argument to `setf` when setting the `left`, `right` or `internal` justification flags

```
cout.setf( ios::left, ios::adjustfield);
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



```
1  // Fig. 21.22: fig21_22.cpp
2  // Left-justification and right-justification.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  #include <iomanip>
9
10 using std::ios;
11 using std::setw;
12 using std::setiosflags;
13 using std::resetiosflags;
14
15 int main()
16 {
17    int x = 12345;
18
19    cout << "Default is right justified:\n"
20         << setw(10) << x << "\n\nUSING MEMBER FUNCTIONS"
21         << "\nUse setf to set ios::left:\n" << setw(10);
22
```

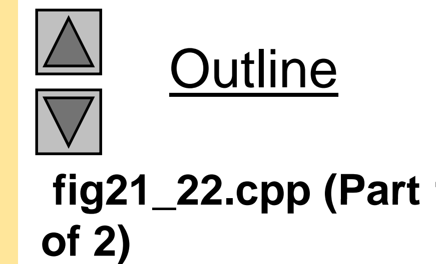


fig21_22.cpp (Part 1 of 2)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
23    cout.setf( ios::left, ios::adjustfield );
24    cout << x << "\nUse unsetf to restore default:\n";
25    cout.unsetf( ios::left );
26    cout << setw( 10 ) << x
27         << "\n\nUSING PARAMETERIZED STREAM MANIPULATORS"
28         << "\nUse setiosflags to set ios::left:\n"
29         << setw( 10 ) << setiosflags( ios::left ) << x
30         << "\nUse resetiosflags to restore default:\n"
31         << setw( 10 ) << resetiosflags( ios::left )
32         << x << endl;
33    return 0;
34 } // end function main
```

```
Default is right justified:
     12345

USING MEMBER FUNCTIONS
Use setf to set ios::left:
12345
Use unsetf to restore default:
     12345

USING PARAMETERIZED STREAM MANIPULATORS
Use setiosflags to set ios::left:
12345
Use resetiosflags to restore default:
     12345
```

Outline

fig21_22.cpp (Part 2 of 2)

Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
1  // Fig. 21.23: fig21_23.cpp
2  // Printing an integer with internal spacing and
3  // forcing the plus sign.
4  #include <iostream>
5
6  using std::cout;
7  using std::endl;
8
9  #include <iomanip>
10
11 using std::ios;
12 using std::setiosflags;
13 using std::setw;
14
15 int main()
16 {
17    cout << setiosflags( ios::internal | ios::showpos )
18         << setw( 10 ) << 123 << endl;
19    return 0;
20 } // end function main
```

```
+      123
```

Outline

fig21_23.cpp

Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

21.7.4 Padding (`fill`, `setfill`)

- `fill` member function
  - Specifies the fill character
  - Space is default
  - Returns the prior padding character

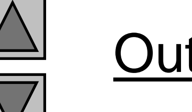
```
cout.fill('*');
```

- `setfill` manipulator
  - Also sets fill character

```
cout << setfill ('*');
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
1  // Fig. 21.24: fig21_24.cpp
2  // Using the fill member function and the setfill
3  // manipulator to change the padding character for
4  // fields larger than the values being printed.
5  #include <iostream>
6
7  using std::cout;
8  using std::endl;
9
10 #include <iomanip>
11
12 using std::ios;
13 using std::setw;
14 using std::hex;
15 using std::dec;
16 using std::setfill;
17
18 int main()
19 {
20    int x = 10000;
21
```



Outline

fig21_24.cpp (Part 1 of 2)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
22     cout << x << " printed as int right and left justified\n"
23          << "and as hex with internal justification.\n"
24          << "Using the default pad character (space):\n";
25     cout.setf( ios::showbase );
26     cout << setw( 10 ) << x << '\n';
27     cout.setf( ios::left, ios::adjustfield );
28     cout << setw( 10 ) << x << '\n';
29     cout.setf( ios::internal, ios::adjustfield );
30     cout << setw( 10 ) << hex << x;
31
32     cout << "\n\nUsing various padding characters:\n";
33     cout.setf( ios::right, ios::adjustfield );
34     cout.fill( '*' );
35     cout << setw( 10 ) << dec << x << '\n';
36     cout.setf( ios::left, ios::adjustfield );
37     cout << setw( 10 ) << setfill( '%' ) << x << '\n';
38     cout.setf( ios::internal, ios::adjustfield );
39     cout << setw( 10 ) << setfill( '^' ) << hex << x << endl;
40     return 0;
41  } // end function main
```

fig21_24.cpp (Part 1 of 2)

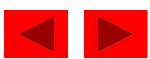
```
10000 printed as int right and left justified
and as hex with internal justification.
Using the default pad character (space):
     10000
10000
0x    2710

Using various padding characters:
*****10000
10000%%%%%
0x^^^^2710
```

Program Output

21.7.5- Integral Stream Base (`ios::dec`, `ios::oct`, `ios::hex`, `ios::showbase`)

- `ios::basefield` static member
  - Used similarly to `ios::adjustfield` with `setf`
  - Includes the `ios::oct`, `ios::hex` and `ios::dec` flag bits
  - Specify that integers are to be treated as octal, hexadecimal and decimal values
  - Default is decimal
  - Default for stream extractions depends on form inputted
    - Integers starting with `0` are treated as octal
    - Integers starting with `0x` or `0X` are treated as hexadecimal
  - Once a base specified, settings stay until changed

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Outline

fig21_25.cpp

```
1 // Fig. 21.25: fig21_25.cpp
2 // Using the ios::showbase flag
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
7
8 #include <iomanip>
9
10 using std::ios;
11 using std::setiosflags;
12 using std::oct;
13 using std::hex;
14
15 int main()
16 {
17    int x = 100;
18
19    cout << setiosflags( ios::showbase )
20         << "Printing integers preceded by their base:\n"
21         << x << '\n'
22         << oct << x << '\n'
23         << hex << x << endl;
24    return 0;
25 } // end function main
```

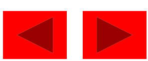
Program Output

```
Printing integers preceded by their base:
100
0144
0x64
```

21.7.6 Floating-Point Numbers; Scientific Notation (`ios::scientific`, `ios::fixed`)

- `ios::scientific`
  - Forces output of a floating point number in scientific notation:
    - `1.946000e+009`
- `ios::fixed`
  - Forces floating point numbers to display a specific number of digits to the right of the decimal (specified with `precision`)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



21.7.6 Floating-Point Numbers; Scientific Notation (ios::scientific, ios::fixed)

- static data member ios::floatfield
  - Contains ios::scientific and ios::fixed
  - Used similarly to ios::adjustfield and ios::basefield in setf
    - cout.setf(ios::scientific, ios::floatfield);
  - cout.setf(0, ios::floatfield) restores default format for outputting floating-point numbers

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

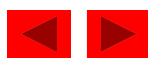
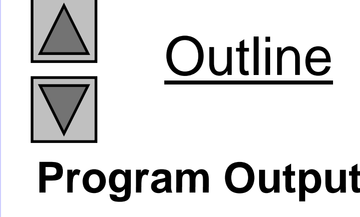


fig21_26.cpp

```
1  // Fig. 21.26: fig21_26.cpp
2  // Displaying floating-point values in system default,
3  // scientific, and fixed formats.
4  #include <iostream>
5
6  using std::cout;
7  using std::endl;
8  using std::ios;
9
10 int main()
11 {
12    double x = .001234567, y = 1.946e9;
13
14    cout << "Displayed in default format:\n"
15         << x << '\t' << y << '\n';
16    cout.setf( ios::scientific, ios::floatfield );
17    cout << "Displayed in scientific format:\n"
18         << x << '\t' << y << '\n';
19    cout.unsetf( ios::scientific );
20    cout << "Displayed in default format after unsetf:\n"
21         << x << '\t' << y << '\n';
22    cout.setf( ios::fixed, ios::floatfield );
23    cout << "Displayed in fixed format:\n"
24         << x << '\t' << y << endl;
25    return 0;
26 } // end function main
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
Displayed in default format:
0.00123457      1.946e+009
Displayed in scientific format:
1.234567e-003   1.946000e+009
Displayed in default format after unsetf:
0.00123457      1.946e+009
Displayed in fixed format:
0.001235        1946000000.000000
```



Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

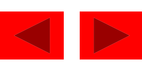
21.7.7 Uppercase/Lowercase Control (`ios::uppercase`)

- `ios::uppercase`
  - Forces uppercase `E` to be output with scientific notation

    `4.32E+010`
  - Forces uppercase `X` to be output with hexadecimal numbers, and causes all letters to be uppercase

    `75BDE`

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Outline

fig21_27.cpp

```
1  // Fig. 21.27: fig21_27.cpp
2  // Using the ios::uppercase flag
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  #include <iomanip>
9
10 using std::setiosflags;
11 using std::ios;
12 using std::hex;
13
14 int main()
15 {
16    cout << setiosflags( ios::uppercase )
17         << "Printing uppercase letters in scientific\n"
18         << "notation exponents and hexadecimal values:\n"
19         << 4.345e10 << '\n' << hex << 123456789 << endl;
20    return 0;
21 } // end function main
```

Program Output

```
Printing uppercase letters in scientific
notation exponents and hexadecimal values:
4.345E+010
75BCD15
```

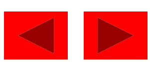
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

21.7.8 Setting and Resetting the Format Flags (`flags`, `setiosflags`, `resetiosflags`)

- `flags` member function
  - Without argument, returns the current settings of the format flags (as a `long` value)
  - With a `long` argument, sets the format flags as specified
    - Returns prior settings
- `setf` member function
  - Sets the format flags provided in its argument
  - Returns the previous flag settings as a `long` value
  - Unset the format using `unsetf` member function

```
long previousFlagSettings =
    cout.setf( ios::showpoint | ios::showpos );
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



21.7.8 Setting and Resetting the Format Flags (flags, setiosflags, resetiosflags)

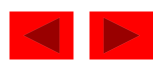
- `setf` with two `long` arguments

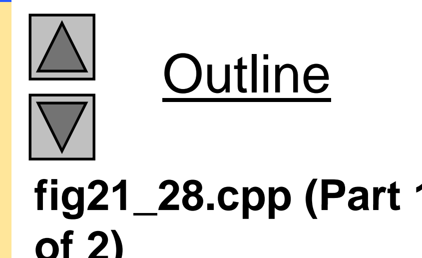
  `cout.setf( ios::left, ios::adjustfield );`

  clears the bits of `ios::adjustfield` then sets `ios::left`

  - This version of setf can be used with
  - `ios::basefield (ios::dec, ios::oct, ios::hex)`
  - `ios::floatfield (ios::scientific, ios::fixed)`
  - `ios::adjustfield (ios::left, ios::right, ios::internal )`
- `unsetf`
  - Resets specified flags
  - Returns previous settings

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.





Outline

fig21_28.cpp (Part 1 of 2)

```
1  // Fig. 21.28: fig21_28.cpp
2  // Demonstrating the flags member function.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7  using std::ios;
8
9
10 int main()
11 {
12    int i = 1000;
13    double d = 0.0947628;
14
15    cout << "The value of the flags variable is: "
16         << cout.flags()
17         << "\nPrint int and double in original format:\n"
18         << i << '\t' << d << "\n\n";
19    long originalFormat =
20            cout.flags( ios::oct | ios::scientific );
21    cout << "The value of the flags variable is: "
22         << cout.flags()
23         << "\nPrint int and double in a new format\n"
24         << "specified using the flags member function:\n"
25         << i << '\t' << d << "\n\n";
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
26    cout.flags( originalFormat );
27    cout << "The value of the flags variable is: "
28         << cout.flags()
29         << "\nPrint values in original format again:\n"
30         << i << '\t' << d << endl;
31    return 0;
32 } // end function main
```

```
The value of the flags variable is: 513
Print int and double in original format:
1000    0.0947628

The value of the flags variable is: 12000
Print int and double in a new format
specified using the flags member function:
1750    9.476280e-002

The value of the flags variable is: 513
Print values in original format again:
1000    0.0947628
```

Outline

fig21_28.cpp (Part 2 of 2)

Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

21.8 Stream Error States

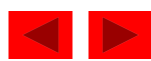
- `eofbit`
  - Set for an input stream after end-of-file encountered
  - `cin.eof()` returns `true` if end-of-file has been encountered on `cin`
- `failbit`
  - Set for a stream when a format error occurs
  - `cin.fail()` - returns `true` if a stream operation has failed
  - Normally possible to recover from these errors

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

21.8 Stream Error States

- `badbit`
  - Set when an error occurs that results in data loss
  - `cin.bad()` returns `true` if stream operation failed
  - normally nonrecoverable
- `goodbit`
  - Set for a stream if neither `eofbit`, `failbit` or `badbit` are set
  - `cin.good()` returns `true` if the `bad`, `fail` and `eof` functions would all return false.
  - I/O operations should only be performed on "good" streams
- `rdstate`
  - Returns the state of the stream
  - Stream can be tested with a `switch` statement that examines all of the state bits
  - Easier to use `eof`, `bad`, `fail`, and `good` to determine state

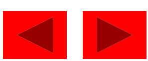
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



21.8 Stream Error States

- `clear`
  - Used to restore a stream's state to "good"
  - `cin.clear()` clears `cin` and sets `goodbit` for the stream
  - `cin.clear( ios::failbit )` actually sets the `failbit`
    - Might do this when encountering a problem with a user-defined type
- Other operators
  - `operator!`
    - Returns `true` if `badbit` or `failbit` set
  - `operator void*`
    - Returns `false` if `badbit` or `failbit` set
  - Useful for file processing

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Outline

fig21_29.cpp (Part 1 of 2)

```
1  // Fig. 21.29: fig21_29.cpp
2  // Testing error states.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7  using std::cin;
8
9  int main()
10 {
11    int x;
12    cout << "Before a bad input operation:"
13         << "\ncin.rdstate(): " << cin.rdstate()
14         << "\n    cin.eof(): " << cin.eof()
15         << "\n   cin.fail(): " << cin.fail()
16         << "\n    cin.bad(): " << cin.bad()
17         << "\n   cin.good(): " << cin.good()
18         << "\n\nExpects an integer, but enter a character: ";
19    cin >> x;
20
21    cout << "\nAfter a bad input operation:"
22         << "\ncin.rdstate(): " << cin.rdstate()
23         << "\n    cin.eof(): " << cin.eof()
24         << "\n   cin.fail(): " << cin.fail()
25         << "\n    cin.bad(): " << cin.bad()
26         << "\n   cin.good(): " << cin.good() << "\n\n";
27
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
28    cin.clear();
29
30    cout << "After cin.clear()"
31         << "\ncin.fail(): " << cin.fail()
32         << "\ncin.good(): " << cin.good() << endl;
33    return 0;
34 } // end function main
```

```
Before a bad input operation:
cin.rdstate(): 0
    cin.eof(): 0
   cin.fail(): 0
    cin.bad(): 0
   cin.good(): 1

Expects an integer, but enter a character: A

After a bad input operation:
cin.rdstate(): 2
    cin.eof(): 0
   cin.fail(): 1
    cin.bad(): 0
   cin.good(): 0

After cin.clear()
cin.fail(): 0
cin.good(): 1
```

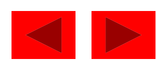

Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

21.9 Tying an Output Stream to an Input Stream

- `tie` member function
  - Synchronize operation of an `istream` and an `ostream`
  - Outputs appear before subsequent inputs
  - Automatically done for `cin` and `cout`
- `inputStream.tie( &outputStream );`
  - Ties `inputStream` to `outputStream`
  - `cin.tie( &cout)` done automatically
- `inputStream.tie( 0 );`
  - Unties `inputStream` from an output stream

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

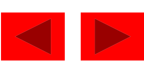


Chapter 22 - C++ Templates

Outline

22.1 Introduction
22.2 Class Templates
22.3 Class Templates and Non-type Parameters
22.4 Templates and Inheritance
22.5 Templates and friends
22.6 Templates and static Members

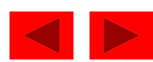
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Objectives

- In this chapter, you will learn:
  - To be able to use class templates to create a group of related types.
  - To be able to distinguish between class templates and template classes.
  - To understand how to overload template functions.
  - To understand the relationships among templates, `friend`s, inheritance and `static` members.

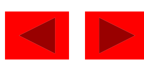
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



22.1 Introduction

- Templates
  - Easily create a large range of related functions or classes
  - Function template - the blueprint of the related functions
  - Template function - a specific function *made* from a function template

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



22.2 Class Templates

- Class templates
  - Allow type-specific versions of generic classes
- Format:

```
template <class T>
class ClassName{
 definition
 }
```

  - Need not use "T", any identifier will work
  - To create an object of the class, type

```
ClassName< type > myObject;
```

 Example: `Stack< double > doubleStack;`

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

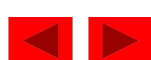
22.2 Class Templates (II)

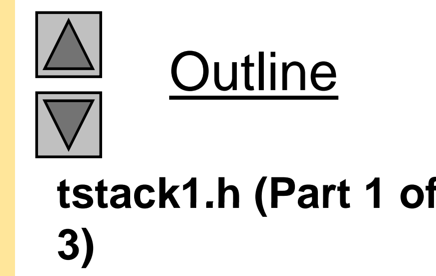
- Template class functions
  - Defined normally, but preceded by `template<class T>`
    - Generic data in class listed as type `T`
  - Binary scope resolution operator used
  - Template class function definition:

```
template<class T>
MyClass< T >::MyClass(int size)
{
   myArray = new T[size];
}
```

    - Constructor definition - creates an array of type `T`

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

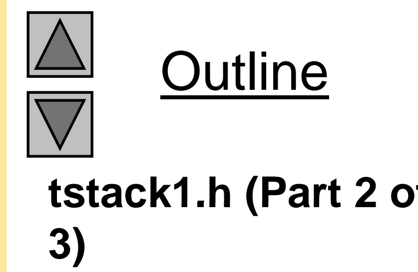




tstack1.h (Part 1 of 3)

```
1  // Fig. 22.1: tstack1.h
2  // Class template Stack
3  #ifndef TSTACK1_H
4  #define TSTACK1_H
5
6  template< class T >
7  class Stack {
8  public:
9     Stack( int = 10 );    // default constructor (stack size 10)
10    ~Stack() { delete [] stackPtr; } // destructor
11    bool push( const T& ); // push an element onto the stack
12    bool pop( T& );        // pop an element off the stack
13 private:
14    int size;             // # of elements in the stack
15    int top;              // location of the top element
16    T *stackPtr;          // pointer to the stack
17
18    bool isEmpty() const { return top == -1; }      // utility
19    bool isFull() const { return top == size - 1; } // functions
20 }; // end class template Stack
21
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



tstack1.h (Part 2 of 3)

```
22 // Constructor with default size 10
23 template< class T >
24 Stack< T >::Stack( int s )
25 {
26    size = s > 0 ? s : 10;
27    top = -1;                // Stack is initially empty
28    stackPtr = new T[ size ]; // allocate space for elements
29 } // end Stack constructor
30
31 // Push an element onto the stack
32 // return 1 if successful, 0 otherwise
33 template< class T >
34 bool Stack< T >::push( const T &pushValue )
35 {
36    if ( !isFull() ) {
37       stackPtr[ ++top ] = pushValue; // place item in Stack
38       return true;  // push successful
39    } // end if
40    return false;     // push unsuccessful
41 } // end function template push
42
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
43 // Pop an element off the stack
44 template< class T >
45 bool Stack< T >::pop( T &popValue )
46 {
47    if ( !isEmpty() ) {
48       popValue = stackPtr[ top-- ];  // remove item from Stack
49       return true;  // pop successful
50    } // end if
51    return false;     // pop unsuccessful
52 } // end function template pop
53
54 #endif
```

```
55 // Fig. 22.1: fig22_01.cpp
56 // Test driver for Stack template
57 #include <iostream>
58
59 using std::cout;
60 using std::cin;
61 using std::endl;
62
63 #include "tstack1.h"
64
```

Outline

tstack1.h (Part 3 of 3)

fig22_01.cpp (Part 1 of 3)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
65 int main()
66 {
67    Stack< double > doubleStack( 5 );
68    double f = 1.1;
69    cout << "Pushing elements onto doubleStack\n";
70
71    while ( doubleStack.push( f ) ) { // success true returned
72       cout << f << ' ';
73       f += 1.1;
74    } // end while
75
76    cout << "\nStack is full. Cannot push " << f
77         << "\n\nPopping elements from doubleStack\n";
78
79    while ( doubleStack.pop( f ) )  // success true returned
80       cout << f << ' ';
81
82    cout << "\nStack is empty. Cannot pop\n";
83
84    Stack< int > intStack;
85    int i = 1;
86    cout << "\nPushing elements onto intStack\n";
87
88    while ( intStack.push( i ) ) { // success true returned
89       cout << i << ' ';
90       ++i;
91    } // end while
92
```

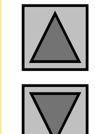


fig22_01.cpp (Part 2 of 3)

```
93    cout << "\nStack is full. Cannot push " << i
94         << "\n\nPopping elements from intStack\n";
95
96    while ( intStack.pop( i ) )  // success true returned
97       cout << i << ' ';
98
99    cout << "\nStack is empty. Cannot pop\n";
100    return 0;
101 } // end function main
```

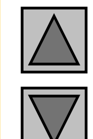
```
Pushing elements onto doubleStack
1.1 2.2 3.3 4.4 5.5
Stack is full. Cannot push 6.6

Popping elements from doubleStack
5.5 4.4 3.3 2.2 1.1
Stack is empty. Cannot pop

Pushing elements onto intStack
1 2 3 4 5 6 7 8 9 10
Stack is full. Cannot push 11

Popping elements from intStack
10 9 8 7 6 5 4 3 2 1
Stack is empty. Cannot pop
```

fig22_01.cpp (Part 3 of 3)

Program Output

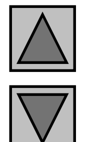
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
1  // Fig. 22.2: fig22_02.cpp
2  // Test driver for Stack template.
3  // Function main uses a function template to manipulate
4  // objects of type Stack< T >.
5  #include <iostream>
6
7  using std::cout;
8  using std::cin;
9  using std::endl;
10
11 #include "tstack1.h"
12
13 // Function template to manipulate Stack< T >
14 template< class T >
15 void testStack(
16    Stack< T > &theStack,    // reference to the Stack< T >
17    T value,                 // initial value to be pushed
18    T increment,             // increment for subsequent values
19    const char *stackName ) // name of the Stack < T > object
20 {
21    cout << "\nPushing elements onto " << stackName << '\n';
22
23    while ( theStack.push( value ) ) { // success true returned
24       cout << value << ' ';
25       value += increment;
26    } // end while
27
```



© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
28    cout << "\nStack is full. Cannot push " << value
29         << "\n\nPopping elements from " << stackName << '\n';
30
31    while ( theStack.pop( value ) )  // success true returned
32       cout << value << ' ';
33
34    cout << "\nStack is empty. Cannot pop\n";
35 } // end function template testStack
36
37 int main()
38 {
39    Stack< double > doubleStack( 5 );
40    Stack< int > intStack;
41
42    testStack( doubleStack, 1.1, 1.1, "doubleStack" );
43    testStack( intStack, 1, 1, "intStack" );
44
45    return 0;
46 } // end function main
```

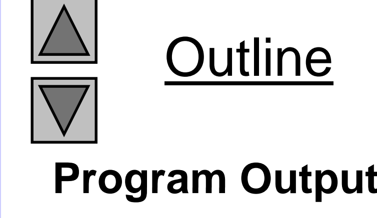


Outline

fig22_02.cpp (Part 2 of 2)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
Pushing elements onto doubleStack
1.1 2.2 3.3 4.4 5.5
Stack is full. Cannot push 6.6

Popping elements from doubleStack
5.5 4.4 3.3 2.2 1.1
Stack is empty. Cannot pop

Pushing elements onto intStack
1 2 3 4 5 6 7 8 9 10
Stack is full. Cannot push 11

Popping elements from intStack
10 9 8 7 6 5 4 3 2 1
Stack is empty. Cannot pop
```



Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

22.3 Class Templates and Non-type Parameters

- Can use non-type parameters in templates
  - Default argument
  - Treated as `const`
- Example:

```
template< class T, int elements >
Stack< double, 100 > mostRecentSalesFigures;
```

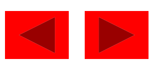
  - Defines object of type `Stack< double, 100>`

  - This may appear in the class definition:

```
T stackHolder[ elements ]; //array to hold stack
```

  - Creates array at compile time, rather than dynamic allocation at execution time

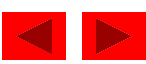
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



22.3 Class Templates and Non-type Parameters (II)

- Classes can be overridden
  - For template class `Array`, define a class named `Array<myCreatedType>`
  - This new class overrides then class template for `myCreatedType`
  - The template remains for unoverriden types

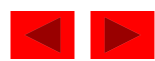
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



22.4 Templates and Inheritance

- A class template can be derived from a template class
- A class template can be derived from a non-template class
- A template class can be derived from a class template
- A non-template class can be derived from a class template

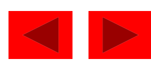
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



22.5 Templates and friends

- Friendships allowed between a class template and
  - Global function
  - Member function of another class
  - Entire class
- friend functions
  - Inside definition of class template **X**:
  - friend void f1();
    - f1() a friend of all template classes
  - friend void f2(X< T > &);
    - f2(X< int > &) is a friend of X< int > only. The same applies for float, double, etc.
  - friend void A::f3();
    - Member function f3 of class A is a friend of all template classes

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

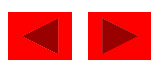


22.5 Templates and `friends` (II)

- `friend void C< T >::f4( X< T > & );`
  - `C<float>::f4( X< float> & )` is a `friend` of `class` `X<float>` only

- `friend` classes
  - `friend class Y;`
    - Every member function of `Y` a friend with every template class made from `X`
  - `friend class Z<T>;`
    - Class `Z<float>` a `friend` of class `X<float>`, etc.

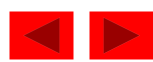
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



22.6 Templates and `static` Members

- Non-template class
  - `static` data members shared between all objects
- Template classes
  - Each class (`int`, `float`, etc.) has its own copy of `static` data members
  - `static` variables initialized at file scope
  - Each template class gets its own copy of `static` member functions

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

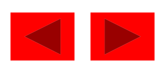


Chapter 23 - Exception Handling

Outline

23.1 Introduction
23.2 When Exception Handling Should Be Used
23.3 Other Error-Handling Techniques
23.4 Basics of C++ Exception Handling: `try`, `throw`, `catch`
23.5 A Simple Exception-Handling Example: Divide by Zero
23.6 Throwing an Exception
23.7 Catching an Exception
23.8 Rethrowing an Exception
23.9 Exception Specifications
23.10 Processing Unexpected Exceptions
23.11 Stack Unwinding
23.12 Constructors, Destructors and Exception Handling
23.13 Exceptions and Inheritance
23.14 Processing `new` Failures
23.15 Class `auto_ptr` and Dynamic Memory Allocation
23.16 Standard Library Exception Hierarchy

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Objectives

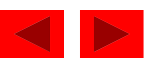
- In this chapter, you will learn:
  - To use `try`, `throw` and `catch` to watch for, indicate and handle exceptions, respectively.
  - To process uncaught and unexpected exceptions.
  - To be able to process `new` failures.
  - To use `auto_ptr` to prevent memory leaks.
  - To understand the standard exception hierarchy.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

23.1 Introduction

- Errors can be dealt with at place error occurs
  - Easy to see if proper error checking implemented
  - Harder to read application itself and see how code works
- Exception handling
  - Makes clear, robust, fault-tolerant programs
  - C++ removes error handling code from "main line" of program
- Common failures
  - new not allocating memory
  - Out of bounds array subscript
  - Division by zero
  - Invalid function parameters

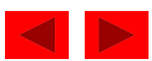
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



23.1 Introduction (II)

- Exception handling - catch errors before they occur
  - Deals with synchronous errors (i.e., Divide by zero)
  - Does not deal with asynchronous errors - disk I/O completions, mouse clicks - use interrupt processing
  - Used when system can recover from error
    - Exception handler - recovery procedure
  - Typically used when error dealt with in different place than where it occurred
  - Useful when program cannot recover but must shut down cleanly
- Exception handling should not be used for program control
  - Not optimized, can harm program performance

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



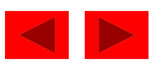
23.1 Introduction (III)

- Exception handling improves fault-tolerance
  - Easier to write error-processing code
  - Specify what type of exceptions are to be caught

- Most programs support only single threads
  - Techniques in this chapter apply for multithreaded OS as well (windows NT, OS/2, some UNIX)

- Exception handling another way to return control from a function or block of code

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



23.2 When Exception Handling Should Be Used

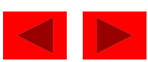
- Error handling should be used for
  - Processing exceptional situations
  - Processing exceptions for components that cannot handle them directly
  - Processing exceptions for widely used components (libraries, classes, functions) that should not process their own exceptions
  - Large projects that require uniform error processing

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

23.3 Other Error-Handling Techniques

- Use `assert`
  - If assertion `false`, the program terminates
- Ignore exceptions
  - Use this "technique" on casual, personal programs - not commercial!
- Abort the program
  - Appropriate for nonfatal errors give appearance that program functioned correctly
  - Inappropriate for mission-critical programs, can cause resource leaks
- Set some error indicator
  - Program may not check indicator at all points where error could occur

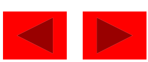
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



23.3 Other Error-Handling Techniques (II)

- Test for the error condition
  - Issue an error message and call `exit`
  - Pass error code to environment
- `setjump` and `longjump`
  - In `<csetjmp>`
  - Jump out of deeply nested function calls back to an error handler.
  - Dangerous - unwinds the stack without calling destructors for automatic objects (more later)
- Specific errors
  - Some have dedicated capabilities for handling them
  - If `new` fails to allocate memory `new_handler` function executes to deal with problem

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



23.4 Basics of C++ Exception Handling: `try`, `throw`, `catch`

- A function can `throw` an exception object if it detects an error
  - Object typically a character string (error message) or class object
  - If exception handler exists, exception caught and handled
  - Otherwise, program terminates

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

23.4 Basics of C++ Exception Handling: `try`, `throw`, `catch` (II)

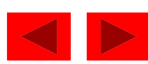
- Format
  - Enclose code that may have an error in `try` block
  - Follow with one or more `catch` blocks
    - Each `catch` block has an exception handler
  - If exception occurs and matches parameter in `catch` block, code in catch block executed
  - If no exception thrown, exception handlers skipped and control resumes after catch blocks
  - `throw` point - place where exception occurred
    - Control cannot return to `throw` point

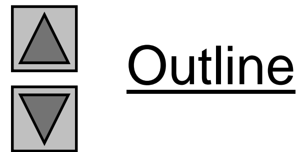
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

23.5 A Simple Exception-Handling Example: Divide by Zero

- Look at the format of `try` and `catch` blocks
- Afterwards, we will cover specifics

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.





Outline

fig23_01.cpp (Part 1 of 3)

```
1  // Fig. 23.1: fig23_01.cpp
2  // A simple exception handling example.
3  // Checking for a divide-by-zero exception.
4  #include <iostream>
5
6  using std::cout;
7  using std::cin;
8  using std::endl;
9
10 // Class DivideByZeroException to be used in exception
11 // handling for throwing an exception on a division by zero.
12 class DivideByZeroException {
13 public:
14    DivideByZeroException()
15       : message( "attempted to divide by zero" ) { }
16    const char *what() const { return message; }
17 private:
18    const char *message;
19 }; // end class DivideByZeroException
20
21 // Definition of function quotient. Demonstrates throwing
22 // an exception when a divide-by-zero exception is encountered.
23 double quotient( int numerator, int denominator )
24 {
25    if ( denominator == 0 )
26       throw DivideByZeroException();
27
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
28    return static_cast< double > ( numerator ) / denominator;
29 } // end function quotient
30
31 // Driver program
32 int main()
33 {
34    int number1, number2;
35    double result;
36
37    cout << "Enter two integers (end-of-file to end): ";
38
39    while ( cin >> number1 >> number2 ) {
40
41       // the try block wraps the code that may throw an
42       // exception and the code that should not execute
43       // if an exception occurs
44       try {
45          result = quotient( number1, number2 );
46          cout << "The quotient is: " << result << endl;
47       } // end try
```

fig23_01.cpp (Part 2 of 3)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

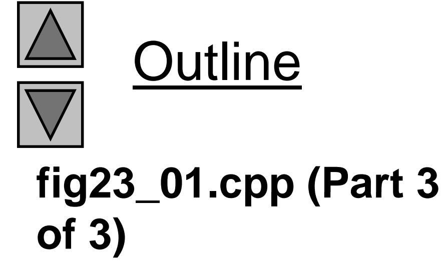


fig23_01.cpp (Part 3 of 3)

```
48       catch ( DivideByZeroException ex ) { // exception handler
49          cout << "Exception occurred: " << ex.what() << '\n';
50       } // end catch
51
52       cout << "\nEnter two integers (end-of-file to end): ";
53    } // end while
54
55    cout << endl;
56    return 0;      // terminate normally
57 } // end function main
```

Program Output

```
Enter two integers (end-of-file to end): 100 7
The quotient is: 14.2857

Enter two integers (end-of-file to end): 100 0
Exception occurred: attempted to divide by zero

Enter two integers (end-of-file to end): 33 9
The quotient is: 3.66667

Enter two integers (end-of-file to end):^Z
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

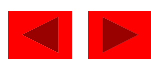
23.6 Throwing an Exception

- `throw` - indicates an exception has occurred
  - Usually has one operand (sometimes zero) of any type
    - If operand an object, called an exception object
    - Conditional expression can be thrown
  - Code referenced in a `try` block can throw an exception
  - Exception caught by closest exception handler
  - Control exits current try block and goes to `catch` handler (if it exists)
  - Example (inside function definition)

```
if ( denominator == 0 )
   throw DivideByZeroException();
```

    - Throws a `dividebyzeroexception` object

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



23.6 Throwing an Exception (II)

- Exception not required to terminate program
  - However, terminates block where exception occurred

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

23.7 Catching an Exception

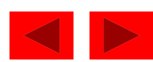
- Exception handlers are in `catch` blocks
  - Format: `catch(` *exceptionType parameterName*`){`

    ```
    exception handling code
    }
    ```
  - Caught if argument type matches `throw` type
  - If not caught then `terminate` called which (by default) calls `abort`
  - Example:

```
catch ( DivideByZeroException ex) {
   cout << "Exception occurred: " << ex.what()
   <<'\n'
}
```

  - Catches exceptions of type `DivideByZeroException`

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



23.7 Catching an Exception (II)

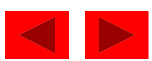
- Catch all exceptions

  `catch(...)` - catches all exceptions

  - You do not know what type of exception occurred
  - There is no parameter name - cannot reference the object

- If no handler matches thrown object
  - Searches next enclosing `try` block
    - If none found, `terminate` called
  - If found, control resumes after last `catch` block
  - If several handlers match thrown object, first one found is executed

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



23.7 Catching an Exception (III)

- `catch` parameter matches thrown object when
  - They are of the same type
    - Exact match required - no promotions/conversions allowed
  - The `catch` parameter is a `public` base class of the thrown object
  - The `catch` parameter is a base-class pointer/ reference type and the thrown object is a derived-class pointer/ reference type
  - The `catch` handler is `catch( ... )`
  - Thrown `const` objects have `const` in the parameter type

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

23.7 Catching an Exception (IV)

- Unreleased resources
  - Resources may have been allocated when exception thrown
  - `catch` handler should `delete` space allocated by `new` and close any opened files
- `catch` handlers can throw exceptions
  - Exceptions can only be processed by outer `try` blocks

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

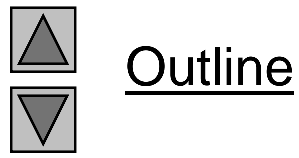
23.8 Rethrowing an Exception

- Rethrowing exceptions
  - Used when an exception handler cannot process an exception
  - Rethrow exception with the statement:

    `throw;`

    - No arguments
    - If no exception thrown in first place, calls `terminate`
  - Handler can always rethrow exception, even if it performed some processing
  - Rethrown exception detected by next enclosing `try` block

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Outline

fig23_02.cpp (Part 1 of 2)

```
1  // Fig. 23.2: fig23_02.cpp
2  // Demonstration of rethrowing an exception.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  #include <exception>
9
10 using std::exception;
11
12 void throwException()
13 {
14    // Throw an exception and immediately catch it.
15    try {
16       cout << "Function throwException\n";
17       throw exception();  // generate exception
18    } // end try
19    catch( exception e )
20    {
21       cout << "Exception handled in function throwException\n";
22       throw;  // rethrow exception for further processing
23    } // end catch
24
25    cout << "This also should not print\n";
26 } // end function throwException
27
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

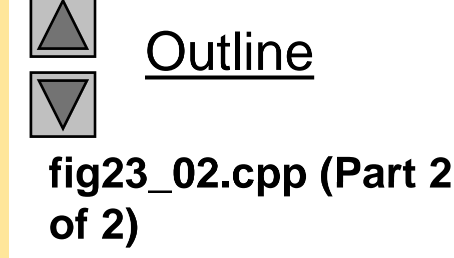


fig23_02.cpp (Part 2 of 2)

```
28 int main()
29 {
30    try {
31       throwException();
32       cout << "This should not print\n";
33    } // end try
34    catch ( exception e )
35    {
36       cout << "Exception handled in main\n";
37    } // end catch
38
39    cout << "Program control continues after catch in main"
40         << endl;
41    return 0;
42 } // end function main
```

Program Output

```
Function throwException
Exception handled in function throwException
Exception handled in main
Program control continues after catch in main
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

23.9 Exception Specifications

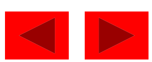
- Exception specification (`throw` list)
  - Lists exceptions that can be thrown by a function

Example:

```
int g( double h ) throw ( a, b, c )
  {
     // function body
  }
```

  - Function can throw listed exceptions or derived types
  - If other type thrown, function `unexpected` called
  - `throw()` (i.e., no `throw` list) states that function will not throw any exceptions
    - In reality, function can still throw exceptions, but calls `unexpected` (more later)
  - If no `throw` list specified, function can `throw` any exception

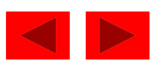
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



23.10 Processing Unexpected Exceptions

- Function unexpected
  - Calls the function specified with set_unexpected
    - Default: terminate
- Function terminate
  - Calls function specified with set_terminate
    - Default: abort
- set_terminate and set_unexpected
  - Prototypes in <exception>
  - Take pointers to functions (i.e., Function name)
    - Function must return void and take no arguments
  - Returns pointer to last function called by terminate or unexpected

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



23.11 Stack Unwinding

- Function-call stack unwound when exception thrown and not caught in a particular scope
  - Tries to catch exception in next outer `try/catch` block
  - Function in which exception was not caught terminates
    - Local variables destroyed
    - Control returns to place where function was called
  - If control returns to a `try` block, attempt made to `catch` exception
    - Otherwise, further unwinds stack
  - If exception not caught, `terminate` called

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
1  // Fig. 23.3: fig23_03.cpp
2  // Demonstrating stack unwinding.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  #include <stdexcept>
9
10 using std::runtime_error;
11
12 void function3() throw ( runtime_error )
13 {
14    throw runtime_error( "runtime_error in function3" );
15 } // end function function3
16
17 void function2() throw ( runtime_error )
18 {
19    function3();
20 } // end function function2
21
22 void function1() throw ( runtime_error )
23 {
24    function2();
25 } // end function function1
26
```

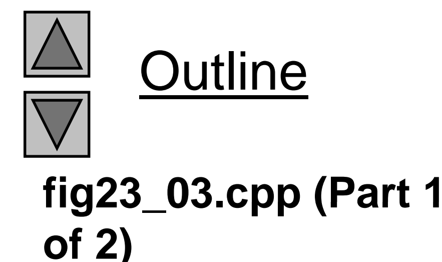


fig23_03.cpp (Part 1 of 2)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
27 int main()
28 {
29    try {
30       function1();
31    } // end try
32    catch ( runtime_error e )
33    {
34       cout << "Exception occurred: " << e.what() << endl;
35    } // end catch
36
37    return 0;
38 } // end function main
```

```
Exception occurred: runtime_error in function3
```


fig23_03.cpp (Part 2 of 2)

Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

23.12 Constructors, Destructors and Exception Handling

- What to do with an error in a constructor?
  - A constructor cannot return a value - how do we let the outside world know of an error?
    - Keep defective object and hope someone tests it
    - Set some variable outside constructor
  - A thrown exception can tell outside world about a failed constructor
  - `catch` handler must have a copy constructor for thrown object

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

23.12 Constructors, Destructors and Exception Handling (II)

- Thrown exceptions in constructors
  - Destructors called for all completed base-class objects and member objects before exception thrown
  - If the destructor that is originally called due to stack unwinding ends up throwing an exception, `terminate` called
  - If object has partially completed member objects when exception thrown, destructors called for completed objects

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

23.12 Constructors, Destructors and Exception Handling (II)

- Resource leak
  - Exception comes before code that releases a resource
  - One solution: initialize local object when resource acquired
    - Destructor will be called before exception occurs

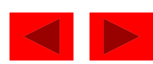
- `catch` exceptions from destructors
  - Enclose code that calls them in `try` block followed by appropriate `catch` block

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

23.13 Exceptions and Inheritance

- Exception classes can be derived from base classes
- If `catch` can get a pointer/reference to a base class, can also `catch` pointers/references to derived classes

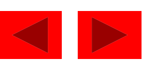
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



23.14 Processing `new` Failures

- If `new` could not allocate memory
  - Old method - use `assert` function
    - If new returns `0`, `abort`
    - Does not allow program to recover
  - Modern method (header `<new>`)
    - new throws `bad_alloc` exception
  - Method used depends on compiler
  - On some compilers: use `new(nothrow)` instead of `new` to have `new` return `0` when it fails
    - Function `set_new_handler(`*functionName*`)` **-** sets which function is called when `new` fails.
    - Function can return no value and take no arguments
    - `new` will not throw `bad_alloc`

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

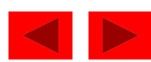


23.14 Processing `new` Failures (II)

- `new`
  - Loop that tries to acquire memory

- A `new` handler function should either:
  - Make more memory available by deleting other dynamically allocated memory and return to the loop in operator `new`
  - Throw an exception of type `bad_alloc`
  - Call function `abort` or `exit` (header `<cstdlib>`) to terminate the program

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



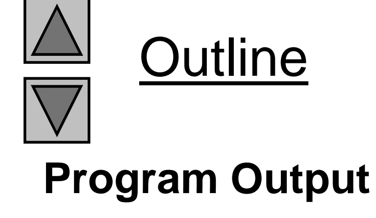
Outline

fig23_04.cpp

```
1  // Fig. 23.4: fig23_04.cpp
2  // Demonstrating new returning 0
3  // when memory is not allocated
4  #include <iostream>
5
6  using std::cout;
7
8  int main()
9  {
10    double *ptr[ 50 ];
11
12    for ( int i = 0; i < 50; i++ ) {
13       ptr[ i ] = new double[ 5000000 ];
14
15       if ( ptr[ i ] == 0 ) { // new failed to allocate memory
16          cout << "Memory allocation failed for ptr[ "
17               << i << " ]\n";
18          break;
19       } // end if
20       else
21          cout << "Allocated 5000000 doubles in ptr[ "
22               << i << " ]\n";
23    } // end for
24
25    return 0;
26 } // end function main
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
Allocated 5000000 doubles in ptr[ 0 ]
Allocated 5000000 doubles in ptr[ 1 ]
Allocated 5000000 doubles in ptr[ 2 ]
Allocated 5000000 doubles in ptr[ 3 ]
Memory allocation failed for ptr[ 4 ]
```



Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

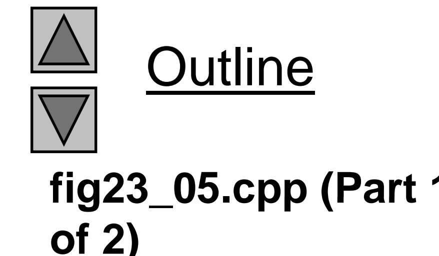


fig23_05.cpp (Part 1 of 2)

```
1  // Fig. 23.5: fig23_05.cpp
2  // Demonstrating new throwing bad_alloc
3  // when memory is not allocated
4  #include <iostream>
5
6  using std::cout;
7  using std::endl;
8
9  #include <new>
10
11 using std::bad_alloc;
12
13 int main()
14 {
15    double *ptr[ 50 ];
16
17    try {
18       for ( int i = 0; i < 50; i++ ) {
19          ptr[ i ] = new double[ 5000000 ];
20          cout << "Allocated 5000000 doubles in ptr[ "
21               << i << " ]\n";
22       } // end for
23    } // end try
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
24    catch ( bad_alloc exception ) {
25       cout << "Exception occurred: "
26            << exception.what() << endl;
27    } // end catch
28
29    return 0;
30 } // end function main
```

fig23_05.cpp (Part 2 of 2)

Program Output

```
Allocated 5000000 doubles in ptr[ 0 ]
Allocated 5000000 doubles in ptr[ 1 ]
Allocated 5000000 doubles in ptr[ 2 ]
Allocated 5000000 doubles in ptr[ 3 ]
Exception occurred: Allocation Failure
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
1  // Fig. 23.6: fig23_06.cpp
2  // Demonstrating set_new_handler
3  #include <iostream>
4
5  using std::cout;
6  using std::cerr;
7
8  #include <new>
9  #include <cstdlib>
10
11 using std::set_new_handler;
12
13 void customNewHandler()
14 {
15    cerr << "customNewHandler was called";
16    abort();
17 } // end function customNewHandler
18
19 int main()
20 {
21    double *ptr[ 50 ];
22    set_new_handler( customNewHandler );
23
24    for ( int i = 0; i < 50; i++ ) {
25       ptr[ i ] = new double[ 5000000 ];
26
```

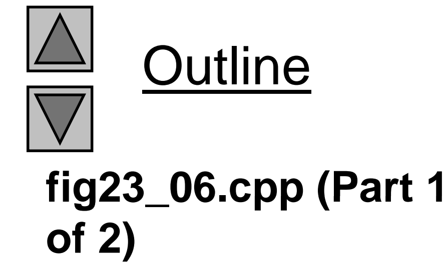


fig23_06.cpp (Part 1 of 2)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
27       cout << "Allocated 5000000 doubles in ptr[ "
28            << i << " ]\n";
29    } // end for
30
31    return 0;
32 } // end function main
```

```
Allocated 5000000 doubles in ptr[ 0 ]
Allocated 5000000 doubles in ptr[ 1 ]
Allocated 5000000 doubles in ptr[ 2 ]
Allocated 5000000 doubles in ptr[ 3 ]
customNewHandler was called
```

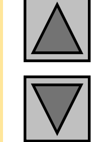


fig23_06.cpp (Part 2 of 2)

Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

23.15 Class `auto_ptr` and Dynamic Memory Allocation

- Pointers to dynamic memory
  - Memory leak can occur if exceptions happens before `delete` command
  - Use class template `auto_ptr` (header `<memory>`) to resolve this
  - `auto_ptr` objects act just like pointers
    - Automatically deletes what it points to when it is destroyed (leaves scope)
    - Can use `*` and `->` like normal pointers

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

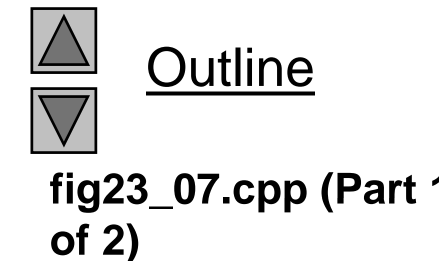


fig23_07.cpp (Part 1 of 2)

```
1  // Fig. 23.7: fig23_07.cpp
2  // Demonstrating auto_ptr
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  #include <memory>
9
10 using std::auto_ptr;
11
12 class Integer {
13 public:
14    Integer( int i = 0 ) : value( i )
15       { cout << "Constructor for Integer " << value << endl; }
16    ~Integer()
17       { cout << "Destructor for Integer " << value << endl; }
18    void setInteger( int i ) { value = i; }
19    int getInteger() const { return value; }
20 private:
21    int value;
22 }; // end class Integer
23
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

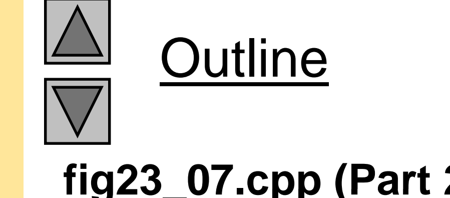


fig23_07.cpp (Part 2 of 2)

```
24 int main()
25 {
26    cout << "Creating an auto_ptr object that points "
27         << "to an Integer\n";
28
29    auto_ptr< Integer > ptrToInteger( new Integer( 7 ) );
30
31    cout << "Using the auto_ptr to manipulate the Integer\n";
32    ptrToInteger->setInteger( 99 );
33    cout << "Integer after setInteger: "
34         << ( *ptrToInteger ).getInteger()
35         << "\nTerminating program" << endl;
36
37    return 0;
38 } // end function main
```

Program Output

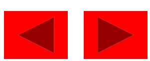
```
Creating an auto_ptr object that points to an Integer
Constructor for Integer 7
Using the auto_ptr to manipulate the Integer
Integer after setInteger: 99
Terminating program
Destructor for Integer 99
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

23.16 Standard Library Exception Hierarchy

- Exceptions fall into categories
  - Hierarchy of exception classes
  - Base class `exception` (header `<exception>`)
    - Function `what()` issues appropriate error message
  - Derived classes: `runtime_error` and `logic_error` (header `<stdexcept>`)
- Class `logic_error`
  - Errors in program logic, can be prevented by writing proper code
  - Derived classes:
    - `invalid_argument` - invalid argument passed to function
    - `length_error` - length larger than maximum size allowed was used
    - `out_of_range` - out of range subscript

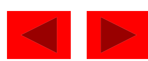
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



23.16 Standard Library Exception Hierarchy (II)

- Class runtime_error
  - Errors detected at execution time
  - Derived classes:
    - overflow_error - arithmetic overflow
    - underflow_error - arithmetic underflow
- Other classes derived from exception
  - Exceptions thrown by C++ language features
    - new - bad_alloc
    - dynamic_cast - bad_cast
    - typeid - bad_typeid
  - Put std::bad_exception in throw list
    - unexpected() will throw bad_exception instead of calling function set by set_unexpected

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

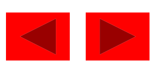


Chapter 24 - Introduction to Java Applications and Applets

Outline

24.1 Introduction
24.2 Basics of a Typical Java Environment
24.3 General Notes about Java and This Book
24.4 A Simple Program: Printing a Line of Text
24.5 Another Java Application: Adding Integers
24.6 Sample Applets from the Java 2 Software Development Kit
24.7 A Simple Java Applet: Drawing a String
24.8 Two More Simple Applets: Drawing Strings and Lines
24.9 Another Java Applet: Adding Integers

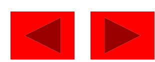
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Objectives

- In this chapter, you will learn:
  - To be able to write simple Java applications.
  - To be able to use input and output statements.
  - To observe some of Java’s exciting capabilities through several demonstration applets provided with the Java 2 Software Development Kit.
  - To understand the difference between an applet and an application.
  - To be able to write simple Java applets.
  - To be able to write simple Hypertext Markup Language (HTML) files to load an applet into the `appletviewer` or a World Wide Web browser.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



24.1 Introduction

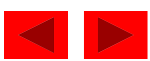
- Java
  - Powerful, object-oriented language
  - Fun to use for beginners, appropriate for experience programmers
  - Language of choice for Internet and network communications
- In the Java chapters, we discuss
  - Graphics (and graphical user interfaces [GUI])
  - Multimedia
  - Event-driven programming
  - Free implementation at http://java.sun.com

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

24.2 Basics of a Typical Java Environment

- Java Systems
  - Consist of environment, language, Java Applications Programming Interface (API), Class libraries
- Java programs have five phases
  - Edit
    - Use an editor to type Java program
    - `vi` or `emacs`, notepad, Jbuilder, Visual J++
    - `.java` extension
  - Compile
    - Translates program into bytecodes, understood by Java interpreter
    - `javac` command: `javac myProgram.java`
    - Creates `.class` file, containing bytecodes (`myProgram.class`)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



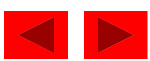
24.2 Basics of a Typical Java Environment (II)

- Java programs have five phases (continued)
  - Loading
    - Class loader transfers `.class` file into memory
      - Applications - run on user's machine
      - Applets - loaded into Web browser, temporary
    - Classes loaded and executed by interpreter with `java` command

      `java Welcome`
    - HTML documents can refer to Java Applets, which are loaded into web browsers. To load,

      `appletviewer Welcome.html`
      - `appletviewer` is a minimal browser, can only interpret applets

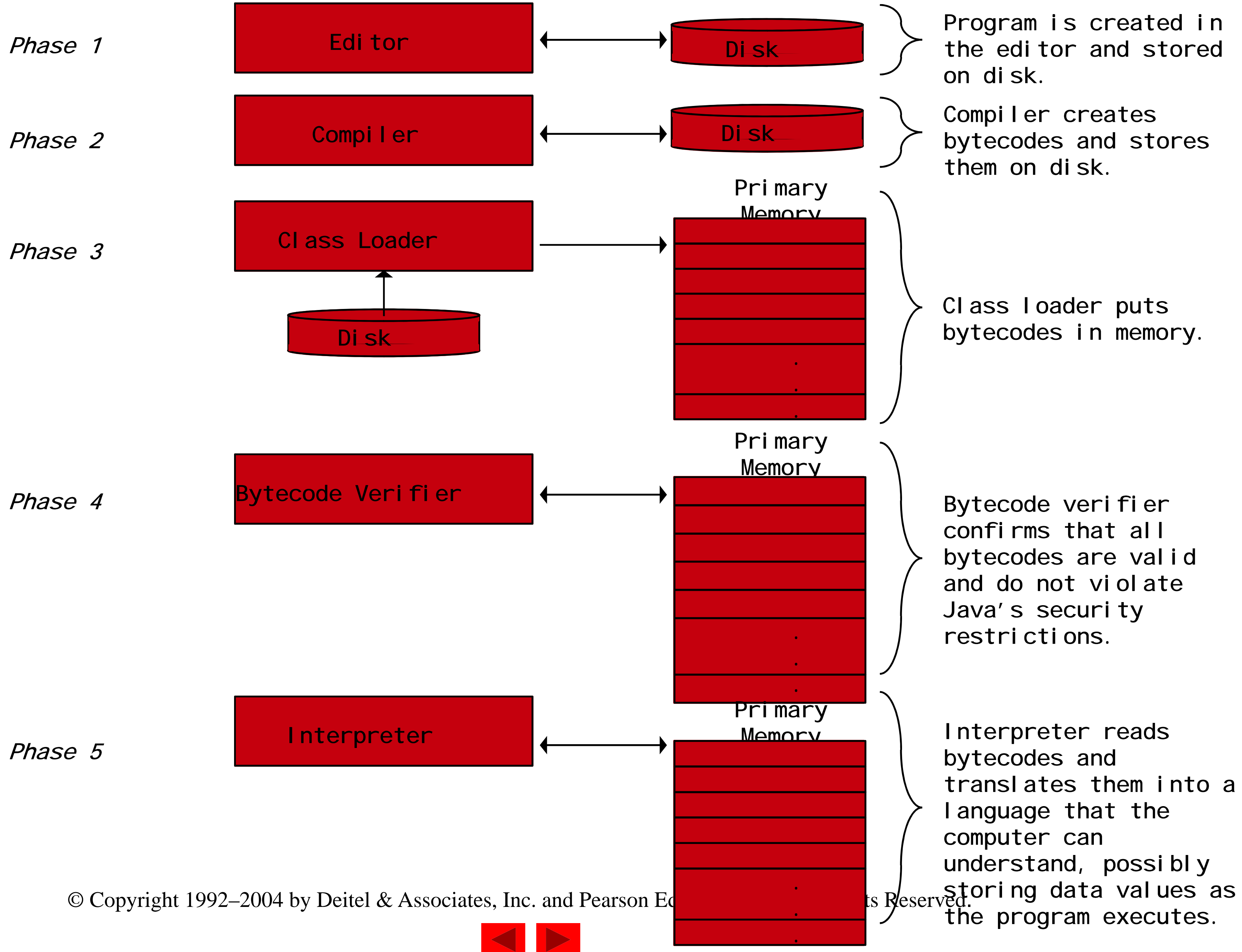
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



24.2 Basics of a Typical Java Environment (II)

- Java programs have five phases (continued)
  - Verify
    - Bytecode verifier makes sure bytecodes are valid and do not violate security
    - Java must be secure - Java programs transferred over networks, possible to damage files (viruses)
  - Execute
    - Computer (controlled by CPU) interprets program one bytecode at a time
    - Performs actions specified in program
  - Program may not work on first try
    - Make changes in edit phase and repeat

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

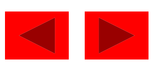


© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Ed... ts Reserved.

24.3 General Notes about Java and This Book

- Java
  - Powerful language
  - Programming
    - Clarity - Keep it Simple
    - Portability - Java portable, but it is an elusive goal
  - Some details of Java not covered
    - `http://java.sun.com` for documentation
  - Performance
    - Interpreted programs run slower than compiled ones
      - Compiling has delayed execution, interpreting executes immediately
    - Can compile Java programs into machine code
      - Runs faster, comparable to C / C++

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



24.3 General Notes about Java and This Book (II)

- Just-in-time compiler
  - Midway between compiling and interpreting
    - As interpreter runs, compiles code and executes it
    - Not as efficient as full compilers
      - Being developed for Java
  - Integrated Development Environment (IDE)
    - Tools to support software development
    - Several Java IDE's are as powerful as C / C++ IDE's

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

24.4 A Simple Program: Printing a Line of Text

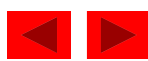
- Application
  - Program that runs using Java interpreter (discussed later)

```
1  // Fig. 24.2: Welcome1.java
2  // A first program in Java
3
4  public class Welcome1 {
5     public static void main( String args[] )
6     {
7        System.out.println( "Welcome to Java Programming!" );
8     } // end main
9  } // end class Welcome1.java
```

```
Welcome to Java Programming!
```

  - Comments
    - Java uses C-style `//` (preferred by Java programmers)
    - Can also use `/* ... */`

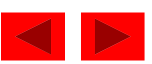
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



24.4 A Simple Program: Printing a Line of Text (II)

- `public class Welcome1 {`
  - Begins class definition
  - Every Java program has a user-defined class
  - Use keyword (reserved word) `class` followed by `ClassName`
    - Name format - `MyClassName`
    - Identifier - letters, digits, underscores, dollar signs, does not begin with a digit, contains no spaces
    - Java case sensitive
  - `public` - For Chapters 24 and 25, every class will be `public`
    - Later, discuss classes that are not (Chapter 26)
    - Programmers initially learn by mimicking features. Explanations come later.
  - When saving a file, class name must be part of file name
    - Save file as `Welcome1.java`

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



24.4 A Simple Program: Printing a Line of Text (III)

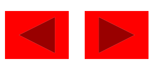
- Braces
  - Body - delineated by left and right braces
    - Class definitions
- `public static void main( String args[] )`
  - Part of every Java application
    - Program begins executing at `main`
    - Must be defined in every Java application
  - `main` is a method (a function)
  - `void` means method returns nothing
    - Many methods can return information
  - Braces used for method body
  - For now, mimic `main`'s first line

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

24.4 A Simple Program: Printing a Line of Text (IV)

- `System.out.println( "Welcome to Java Programming!");`
  - Prints string
    - String - called character string, message string, string literal
    - Characters between quotes a generic string
  - `System.out` - standard output object
    - Displays information in command window
  - Method `System.out.println`
    - Prints a line of text in command window
    - When finished, positions cursor on next line
  - Method `System.out.print`
    - As above, except cursor stays on line
    - `\n` - newline
  - Statements must end with `;`

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



24.4 A Simple Program: Printing a Line of Text (V)

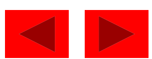
- Executing the program
  - `javac Welcome1`
    - Creates `Welcome1.class` (containing bytecodes)
  - `java Welcome1`
    - Interprets bytecodes in `Welcome1.class` (`.class` left out in `java` command)
    - Automatically calls `main`
- Output types
  - Command window
  - Dialog box / Windows

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

24.4 A Simple Program: Printing a Line of Text (VI)

- Packages
  - Predefined, related classes grouped by directories on disk
    - All in directory `java` or `javax`, or subdirectories
  - Referred to collectively as the Java class library or the Java applications programming interface (Java API)
  - `import` - locates classes needed to compile program
- Class `JOptionPane`
  - Defined in package called `javax.swing`
    - Contains classes used for a graphical user interface (GUI)
      - Facilitates data entry and data output
    - `import javax.swing.JOptionPane;`

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



24.4 A Simple Program: Printing a Line of Text (VII)

- Class `JOptionPane`
  - Contains methods that display a dialog box
    - `static` method `showMessageDialog`
    - First argument - `null` (more Chapter 29)
    - Second argument - string to display
- `static` methods
  - Called using dot operator (`.`) then method name

```
JOptionPane.showMessageDialog(arguments);
```

  - `exit` - method of class `System`
    - Terminates application, required in programs with GUIs

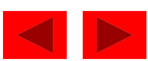
```
System.exit( 0 );
```

`0` - normal exit

`non-zero` - signals that error occurred

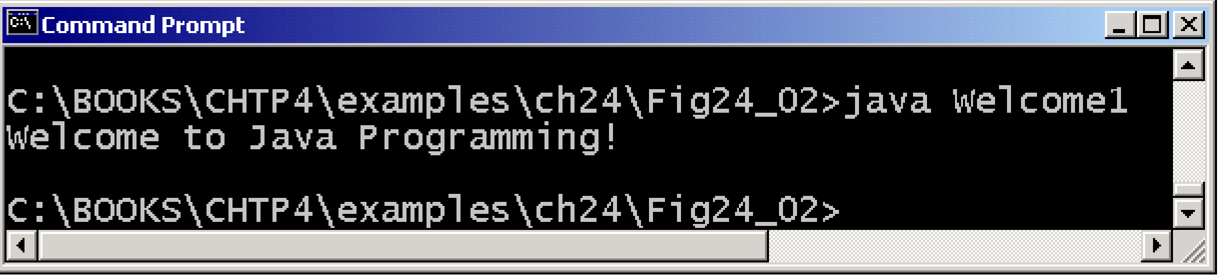
  - Class `System` in package `java.lang`
    - Automatically imported in every Java program

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

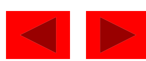


24.4 A Simple Program: Printing a Line of Text (VIII)

Figure 24.3 Executing the Welcome1 application in a Microsoft Windows MS-DOS Prompt



© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



```java
1  // Fig. 24.4: Welcome2.java
2  // Printing multiple lines in a dialog box
3  import javax.swing.JOptionPane;  // import class JOptionPane
4
5  public class Welcome2 {
6     public static void main( String args[] )
7     {
8        JOptionPane.showMessageDialog(
9           null, "Welcome\nto\nJava\nProgramming!" );
10
11        System.exit( 0 );  // terminate the program
12     } // end main
13 } // end class Welcome2
```

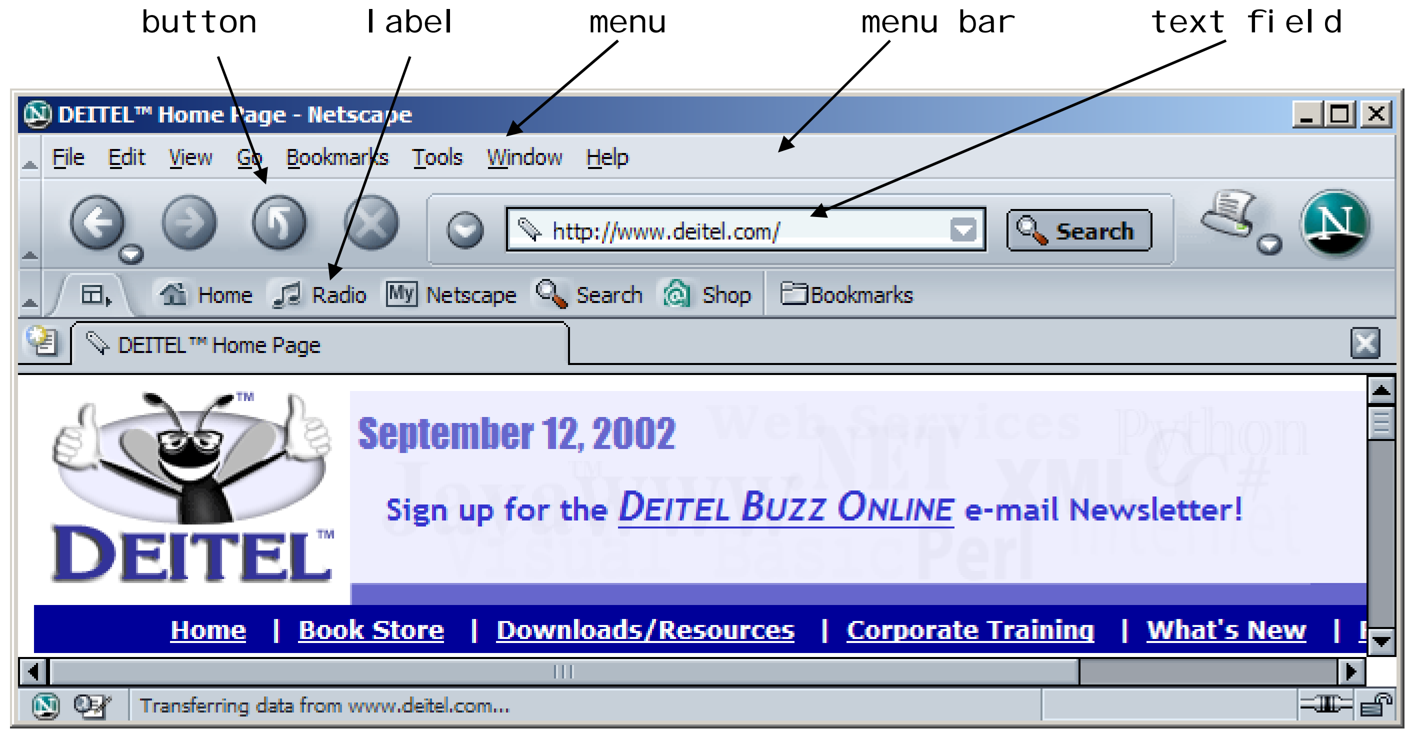
Welcome2.java

Program Output

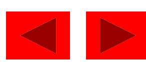
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

24.4 A Simple Program: Printing a Line of Text (IX)

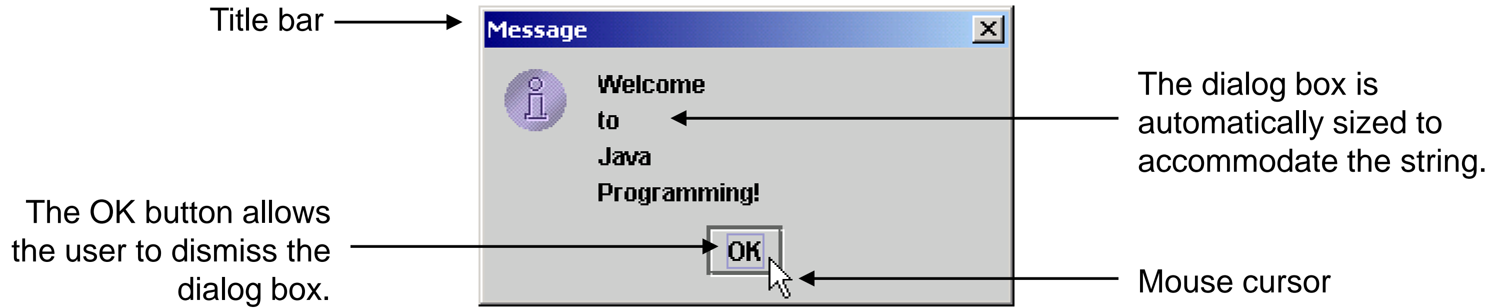
Figure 24.5 A sample Netscape Navigator window with GUI components.



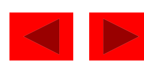
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



24.4 A Simple Program: Printing a Line of Text (X)



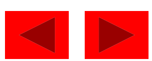
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



24.5 Another Java Application: Adding Integers

- Variables
  - Locations in memory that hold data
  - Must be defined with name and data type before use
    - Primitive data types (keywords): `boolean`, `char`, `byte`, `short`, `int`, `long`, `float`, `double` (details in Chapter 25)
    - `String` (java.lang) - hold strings: `"Hi" "37"`
    - `int` - holds integers: `-1`, `0`, `15`
  - Name format - first letter lowercase, new word capitalized
    - `myVariable`, `myOtherVariable`
  - Definitions: specify name and type
    - Can have multiple variables per definition
    - `int myInt, myInt2, myInt3;`
    - `String myString, myString2;`

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



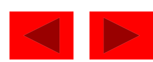
24.5 Another Java Application: Adding Integers (II)

- Method `showInputDialog`
  - Of class `JOptionPane`
  - Displays prompt (gets user input)
    - Argument - Text to display in prompt
  - Java does not have a simple form of input
    - Nothing analogous to `System.out.print`
  - Returns what user input
    - Assign input to a variable using assignment operator =

```
myString = JOptionPane.showInputDialog( "Enter an
  integer" );
```

- = has two operands (binary operator)
  - Expression on right evaluated, assigned to variable on left

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



24.5 Another Java Application: Adding Integers (III)

- `Integer.parseInt`
  - `static` method of class `Integer`
  - Input from `showInputDialog` a `String`
    - Want to convert it into an integer
    - `parseInt` takes a `String`, returns an integer

```
myInt = Integer.parseInt( myString );
```

    - Note assignment operator

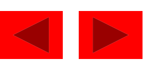
- The + operator
  - String concatenation - "adding" strings

    `"Hello" + " there "` same as `"Hello there"`
  - Print variables

```
"myInt has a value of: " + myInt
```

  - Used for addition, as in C / C++:
    - `sum = int1 + int2;`

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

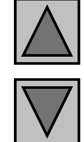


24.5 Another Java Application: Adding Integers (III)

- `showOptionDialog`
  - Another version
  - First argument: `null`
  - Second: message to display
  - Third: string to display in title bar
  - Fourth: type of message to display
    - `JOptionPane.PLAIN_MESSAGE`
    - Other types in Fig. 24.7

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
1  // Fig. 24.6: Addition.java
2  // An addition program
3
4  import javax.swing.JOptionPane;   // import class JOptionPane
5
6  public class Addition {
7     public static void main( String args[] )
8     {
9        String firstNumber,    // first string entered by user
10              secondNumber;   // second string entered by user
11        int number1,           // first number to add
12            number2,           // second number to add
13            sum;               // sum of number1 and number2
14
15        // read in first number from user as a string
16        firstNumber =
17           JOptionPane.showInputDialog( "Enter first integer" );
18
19        // read in second number from user as a string
20        secondNumber =
21           JOptionPane.showInputDialog( "Enter second integer" );
22
23        // convert numbers from type String to type int
24        number1 = Integer.parseInt( firstNumber );
25        number2 = Integer.parseInt( secondNumber );
26
```

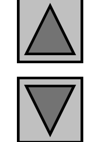


Outline

Addition.java (Part 1 of 2)

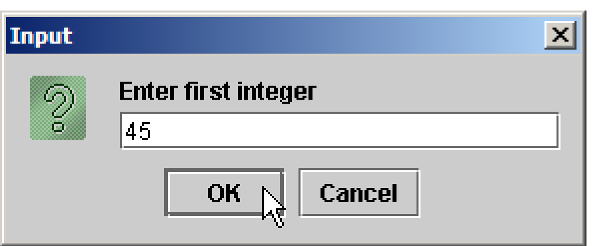
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

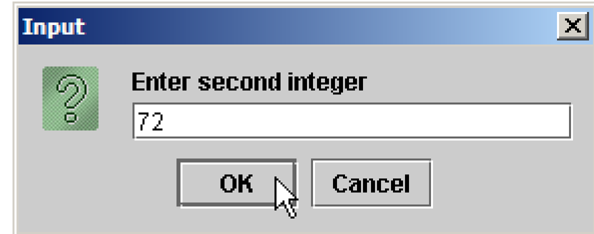
```
27        // add the numbers
28        sum = number1 + number2;
29
30        // display the results
31        JOptionPane.showMessageDialog(
32           null, "The sum is " + sum, "Results",
33           JOptionPane.PLAIN_MESSAGE );
34
35        System.exit( 0 );   // terminate the program
36     } // end main
37  } // end class Addition
```

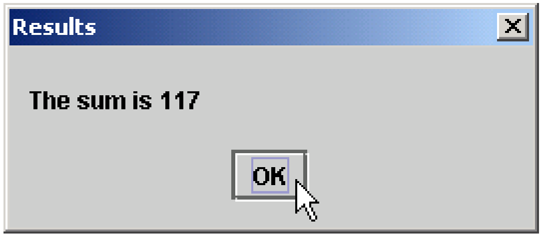


Outline

Addition.java (Part 2 of 2)

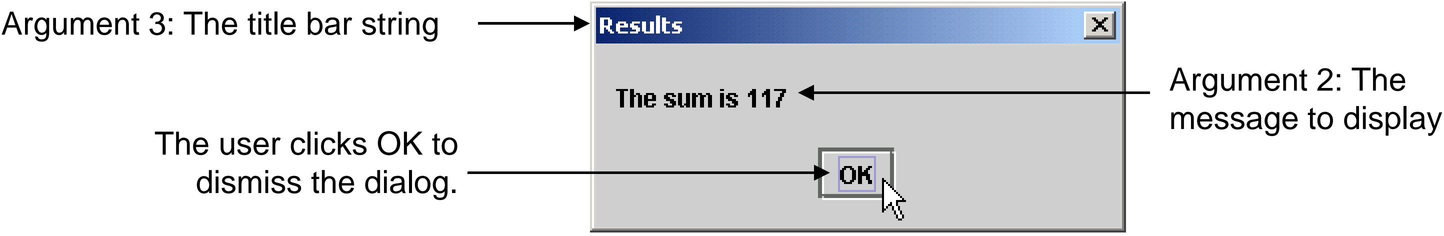




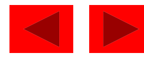


© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

24.5 Another Java Application: Adding Integers (IV)



© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

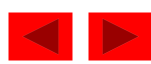


24.5 Another Java Application: Adding Integers (V)

Message dialog type	Icon	Description
JOptionPane.ERROR_MESSAGE		Displays a dialog that indicates an error to the application user.
JOptionPane.INFORMATION_MESSAGE		Displays a dialog with an informational message to the application user—the user can simply dismiss the dialog.
JOptionPane.WARNING_MESSAGE		Displays a dialog that warns the application user of a potential problem.
JOptionPane.QUESTION_MESSAGE		Displays a dialog that poses a question to the application user. This normally requires a response such as clicking a Yes or No button.
JOptionPane.PLAIN_MESSAGE	no icon	Displays a dialog that simply contains a message with no icon.

Fig. 24.7 JOptionPane constants for message dialogs.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



24.6 Sample Applets from the Java 2 Software Development Kit

- Applet
  - Program that runs in
    - `appletviewer` (test utility for applets)
    - Web browser (IE, Communicator)
  - Executes when HTML document containing applet is opened
- Sample Applets
  - Provided in Java 2 Software Development Kit (J2SDK)
  - Source code included (`.java` files)
  - Located in `demo` directory of J2SDK install

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

24.6 Sample Applets from the Java 2 Software Development Kit

- Running applets
  - In command prompt, change to subdirectory of applet
    `cd directoryName`
  - There will be an HTML file used to execute applet
  - type `appletviewer example1.html`
  - Applet will run, `Reload` and `Quit` commands under `Applet` menu
- Example applets
  - Tic-Tac-Toe
  - Drawing programs
  - Animations
  - See Fig. 24.8

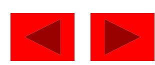
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

24.6 Sample Applets from the Java 2 Software Development Kit

Example	Description
Animator	Performs one of four separate animations.
ArcTest	Demonstrates drawing arcs. You can interact with the applet to change attributes of the arc that is displayed.
BarChart	Draws a simple bar chart.
Blink	Displays blinking text in different colors.
CardTest	Demonstrates several GUI components and a variety of ways in which GUI components can be arranged on the screen. (The arrangement of GUI components is also known as the *layout* of the GUI components.)
Clock	Draws a clock with rotating "hands," the current date and the current time. The clock is updated once per second.
DitherTest	Demonstrates drawing with a graphics technique known as dithering that allows gradual transformation from one color to another.
DrawTest	Allows the user to drag the mouse to draw lines and points on the applet in different colors.
Fractal	Draws a fractal. Fractals typically require complex calculations to determine how they are displayed.
GraphicsTest	Draws a variety of shapes to illustrate graphics capabilities.
GraphLayout	Draws a graph consisting of many nodes (represented as rectangles) connected by lines. Drag a node to see the other nodes in the graph adjust on the screen and demonstrate complex graphical interactions.
ImageMap	Demonstrates an image with *hot spots*. Positioning the mouse pointer over certain areas of the image highlights the area and a message is displayed in the lower-left corner of the `appletviewer` window. Position over the mouth in the image to hear the applet say "hi."
JumpingBox	Moves a rectangle randomly around the screen. Try to catch it by clicking it with the mouse!

Fig. 24.8 The examples from the `applets` directory. (Part 1 of 2)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

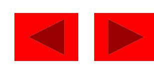


24.6 Sample Applets from the Java 2 Software Development Kit

Example	Description
`MoleculeViewer`	Presents a three-dimensional view of several different chemical molecules. Drag the mouse to view the molecule from different angles.
`NervousText`	Draws text that jumps around the screen.
`SimpleGraph`	Draws a complex curve.
`SortDemo`	Compares three sorting techniques. Sorting (described in Chapter 7) arranges information in order—like alphabetizing words. When you execute the applet, three `appletviewer` windows appear. Click in each one to start the sort. Notice that the sorts all operate at different speeds.
`SpreadSheet`	Demonstrates a simple spreadsheet of rows and columns.
`SymbolTest`	Draws characters from the Java character set.
`TicTacToe`	Allows the user to play Tic-Tac-Toe against the computer.
`WireFrame`	Draws a three-dimensional shape as a wire frame. Drag the mouse to view the shape from different angles.

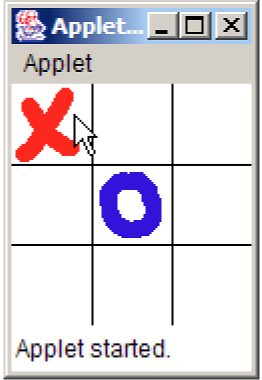
Fig. 24.8 The examples from the `applets` directory. (Part 2 of 2)

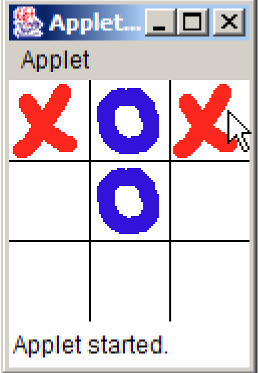
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

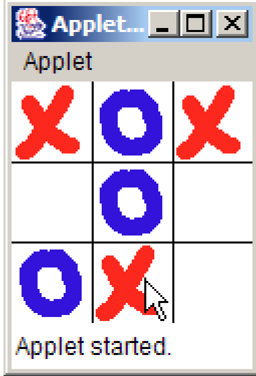


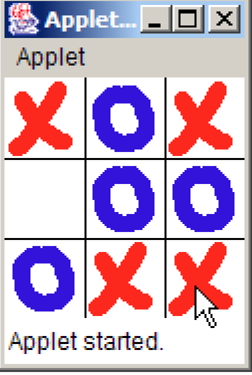
24.6 Sample Applets from the Java 2 Software Development Kit

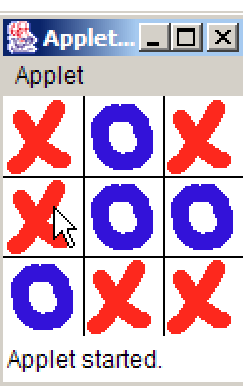
Figure 24.9 Sample execution of the TicTacToe applet.







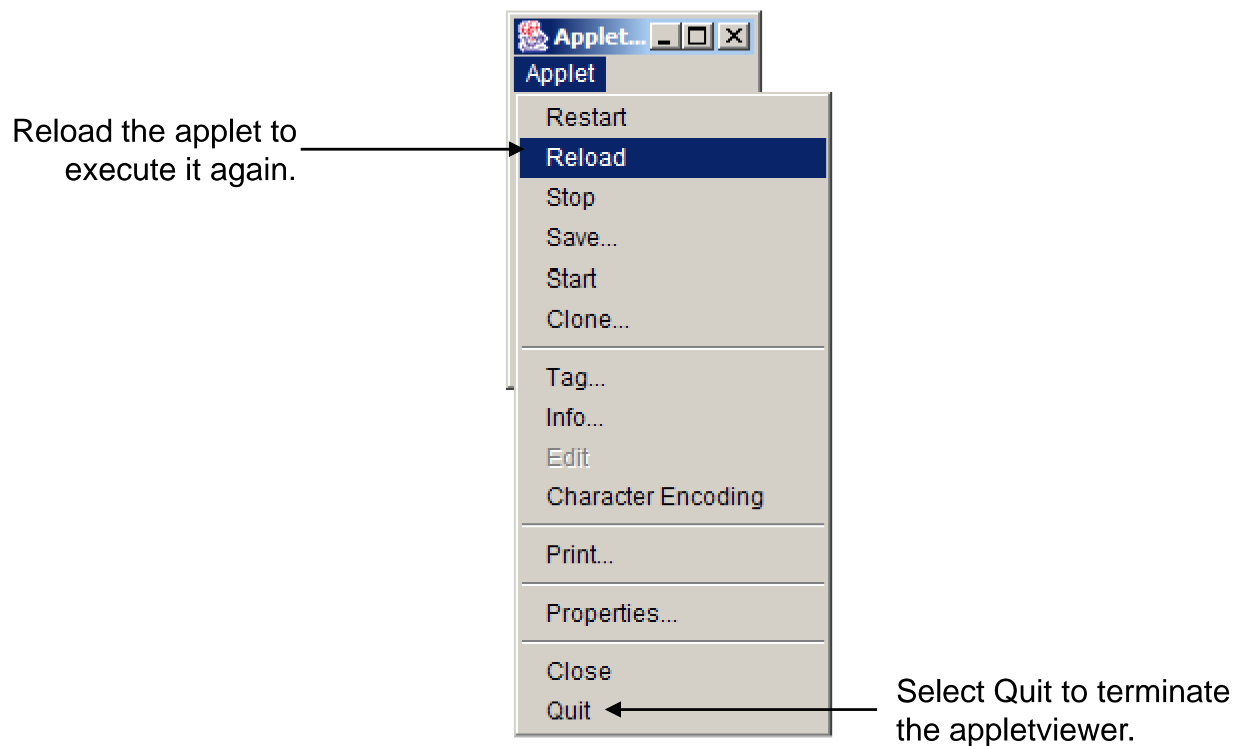




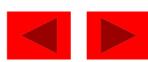
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

24.6 Sample Applets from the Java 2 Software Development Kit

Figure 24.10 Selecting Reload from the appletviewer's Applet menu.



© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



24.6 Sample Applets from the Java 2 Software Development Kit

Figure 24.11 Sample execution of the DrawTest applet.

Applet Viewer: DrawTest.class

Applet

Points

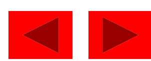
Applet started.

Drag the mouse here to draw.

Select the drawing color by clicking the circle for the color you want. These GUI components are commonly known as *radio buttons*.

Select the shape to draw by clicking the down arrow, then clicking **Lines** or **Points**. This GUI component is commonly known as a *combo box*, *choice* or *drop-down list*.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



24.6 Sample Applets from the Java 2 Software Development Kit

Figure 24.12 Sample execution of the Java2D applet.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

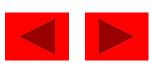
24.7 A Simple Java Applet: Drawing a String

- Create our own applet
  - Print `"Welcome to Java Programming!"`
  - `import javax.swing.JApplet`
    - Needed for all applets
  - `import java.awt.Graphics`
    - Allows program to draw graphics (lines, ovals, text) on an applet
  - Like applications, applets have at least one class definition
- Rarely create applets from scratch
  - Use pieces of class existing definitions

```
public class WelcomeApplet extends JApplet {
```

  - `extends ClassName` - class to inherit from
    - In this case, inherit from class `JApplet`

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



24.7 A Simple Java Applet: Drawing a String (II)

- Inheritance
  - `JApplet` is superclass (base class)
  - `WelcomeApplet` is subclass (derived class)
  - Derived class inherits data and methods of base class
    - Can add new features to derived class
  - Benefits
    - Someone else has already defined what an applet is
      - Applets require over 200 methods to be defined!
      - By using inheritance, all those methods are now ours
    - We do not need to know all the details of `JApplet`

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

24.7 A Simple Java Applet: Drawing a String (III)

- Classes
  - Templates/blueprints create or instantiate objects
    - Objects - locations in memory to store data
    - Implies that data and methods associated with object
- Methods
  - `paint`, `init`, and `start` called automatically for all applets
    - Get "free" version when you inherit from JApplet
    - By default, have empty bodies
    - Must override them and define yourself

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

24.7 A Simple Java Applet: Drawing a String (IV)

- Method `paint`
  - Used to draw graphics, define:

```
public void paint( Graphics g )
```

    - Takes a `Graphics` object `g` as a parameter
    - For now, all method definitions begin with public
  - Call methods of object `g` to draw on applet

```
drawString("String to draw", x, y);
```

    - Draws `"String to draw"` at location (`x`,`y`)
      - Coordinates specify bottom left corner of string
      - (`0`, `0`) is upper left corner of screen
      - Measured in pixels (picture elements)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

24.7 A Simple Java Applet: Drawing a String (IV)

- Create the HTML file (`.html` or `.htm`)
  - Many HTML codes (tags) come in pairs

    ```
    <myTag> ... </myTag>
    ```

  - Create `<HTML>` tags with `<applet>` tags inside
  - `appletviewer` only understands `<applet>` tags
    - Minimal browser
    - Specify complied `.class` file, width, and height of applet (in pixels)

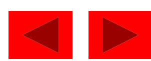
    ```
    <applet code = "WelcomeApplet.class" width = 300
       height = 30>
    ```

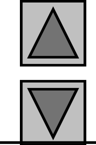
    - Close tag with `</applet>`

- Running the applet

  ```
  appletviewer WelcomeApplet.html
  ```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



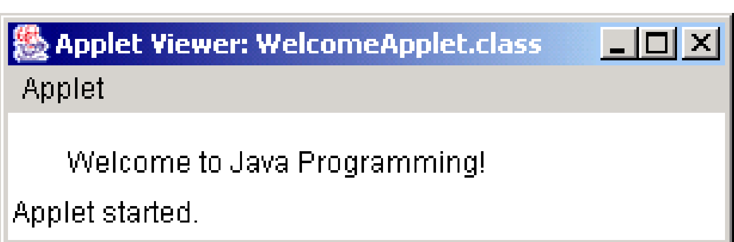


Outline

WelcomeApplet.java

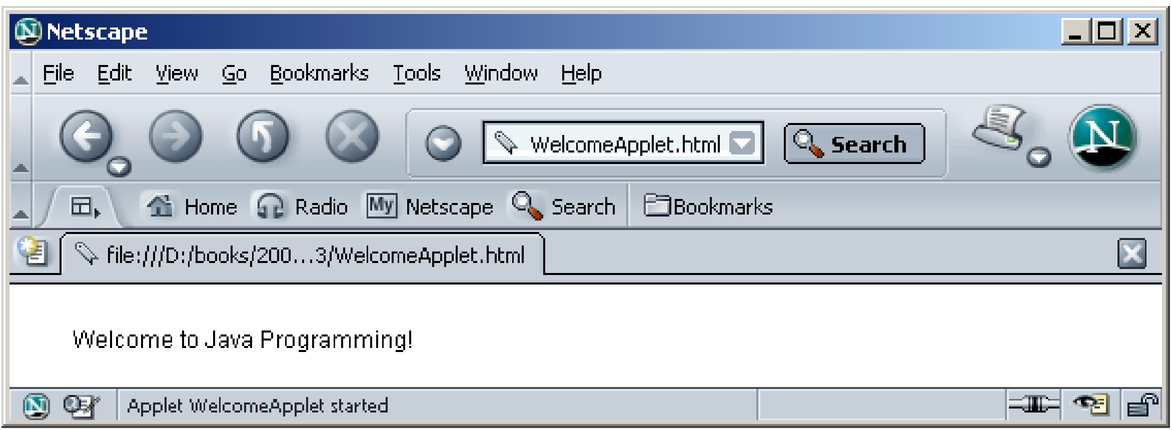
```
1  // Fig. 24.13: WelcomeApplet.java
2  // A first applet in Java
3  import javax.swing.JApplet;   // import class JApplet
4  import java.awt.Graphics;     // import class Graphics
5
6  public class WelcomeApplet extends JApplet {
7     public void paint( Graphics g )
8     {
9        g.drawString( "Welcome to Java Programming!", 25, 25 );
10    } // end method paint
11 } // end class WelcomeApplet
```

```
1  <html>
2  <applet code="WelcomeApplet.class" width=300 height=30>
3  </applet>
4  </html>
```

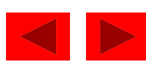


© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

24.8 Two More Simple Applets: Drawing Strings and Lines



© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



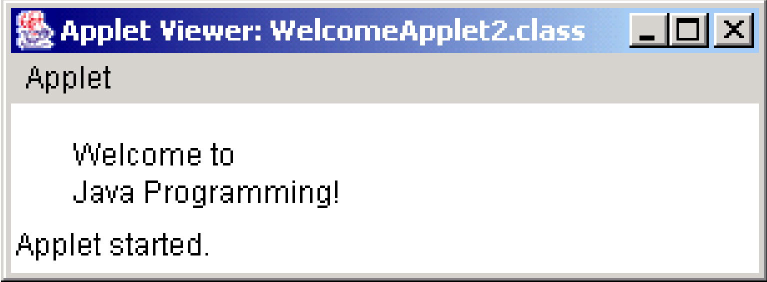
24.8 Two More Simple Applets: Drawing Strings and Lines

- Other methods of class `Graphics`
  - No concept of lines of text, as in `System.out.println` when drawing graphics
  - To print multiple lines, use multiple `drawString` calls
  - `drawLine( x1, y2, x2, y2 ) ;`
    - Draws a line from (`x1`, `y1`) to (`x2`, `y2`)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```java
1  // Fig. 24.15: WelcomeApplet2.java
2  // Displaying multiple strings
3  import javax.swing.JApplet;   // import class JApplet
4  import java.awt.Graphics;     // import class Graphics
5
6  public class WelcomeApplet2 extends JApplet {
7     public void paint( Graphics g )
8     {
9        g.drawString( "Welcome to", 25, 25 );
10        g.drawString( "Java Programming!", 25, 40 );
11     } // end method paint
12 } // end class WelcomeApplet2
```

```html
1  <html>
2  <applet code="WelcomeApplet2.class" width=300 height=45>
3  </applet>
4  </html>
```



Outline

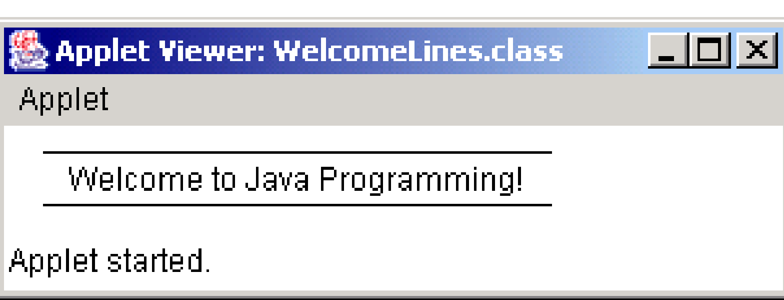
WelcomeApplet2.java

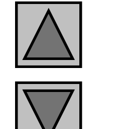
Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```java
1  // Fig. 24.17: WelcomeLines.java
2  // Displaying text and lines
3  import javax.swing.JApplet;   // import class JApplet
4  import java.awt.Graphics;     // import class Graphics
5
6  public class WelcomeLines extends JApplet {
7     public void paint( Graphics g )
8     {
9        g.drawLine( 15, 10, 210, 10 );
10        g.drawLine( 15, 30, 210, 30 );
11        g.drawString( "Welcome to Java Programming!", 25, 25 );
12     } // end method paint
13 } // end class WelcomeLines
```

```html
1  <html>
2  <applet code="WelcomeLines.class" width=300 height=40>
3  </applet>
4  </html>
```





Outline

WelcomeLines.java

Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

24.9 Another Java Applet: Adding Integers

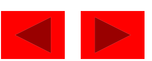
- Next applet mimics program to add two integers
  - This time, use floating point numbers
    - Can have decimal point, `6.7602`
    - `float` - single precision floating point number (7 significant digits)
    - `double` - approximately double precision floating point number (15 significant digits)
      - Uses more memory
  - Use `showInputDialog` to get input, as before
  - Use `Double.parseDouble( String )`
    - Converts a `String` to a `double`

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

24.9 Another Java Applet: Adding Integers (II)

- `import` statements
  - Not necessary if specify full class name every time needed
  `public void paint( java.awt.Graphics g )`
  - `*` - indicates all classes in package should be available
    - `import java.swing.*;`
      - Recall that this contains `JApplet` and `JOptionPane`
    - Does not import subdirectories
- Instance variables
  - Variables defined in body of a class (not in a method)
    - Each object of class gets its own copy
    - Can be used inside any method of the class
  - Before, variables defined in `main`
    - Local variables, known only in body of method defined

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

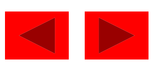


24.9 Another Java Applet: Adding Integers (III)

- Instance variables
  - Have default values
    - Local variables do not, and require initialization before use
    - Good practice to initialize instance variables anyway
- Method `init`
  - Called automatically in all applets
  - Commonly used to initialize variables

    `public void init()`
- References
  - Identifiers (such as `myString`) refer to objects
    - Contain locations in memory
  - References used to call methods, i.e. `g.drawString`

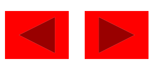
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



24.9 Another Java Applet: Adding Integers (III)

- Variables vs. Objects
  - Variables
    - Defined by a primitive data type
    - `char`, `byte`, `short`, `int`, `long`, `float`, `double`, `boolean`
    - Store one value at a time
    - Variable `myInt`
  - Objects defined in classes
    - Can contain primitive (built-in) data types
    - Can contain methods
    - `Graphics` object `g`
  - If data type a class name, then identifier is a reference
    - Otherwise, identifier is a variable

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

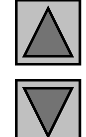


24.9 Another Java Applet: Adding Integers (IV)

- Other methods of class `Graphics`
  - `drawRect( x1, y1, x2, y2 );`
  - Draws a rectangle with upper-left corner (`x1`, `y1`), and lower right corner (`x2, y2`)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
1  // Fig. 24.19: AdditionApplet.java
2  // Adding two floating-point numbers
3  import java.awt.Graphics;   // import class Graphics
4  import javax.swing.*;       // import package javax.swing
5
6  public class AdditionApplet extends JApplet {
7     double sum;  // sum of the values entered by the user
8
9     public void init()
10    {
11       String firstNumber,   // first string entered by user
12              secondNumber;  // second string entered by user
13       double number1,       // first number to add
14              number2;       // second number to add
15
16       // read in first number from user
17       firstNumber =
18          JOptionPane.showInputDialog(
19             "Enter first floating-point value" );
20
21       // read in second number from user
22       secondNumber =
23          JOptionPane.showInputDialog(
24             "Enter second floating-point value" );
25
```



Outline

AdditionApplet.java (Part 1 of 2)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
26        // convert numbers from type String to type double
27        number1 = Double.parseDouble( firstNumber );
28        number2 = Double.parseDouble( secondNumber );
29
30        // add the numbers
31        sum = number1 + number2;
32     } // end method init
33
34     public void paint( Graphics g )
35     {
36        // draw the results with g.drawString
37        g.drawRect( 15, 10, 270, 20 );
38        g.drawString( "The sum is " + sum, 25, 25 );
39     } // end method paint
40  } // end class AdditionApplet
```

```
1  <html>
2  <applet code="AdditionApplet.class" width=300 height=50>
3  </applet>
4  </html>
```

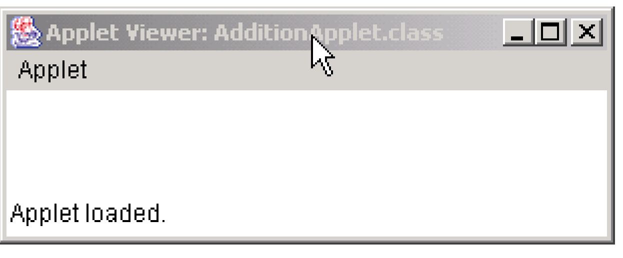
Outline

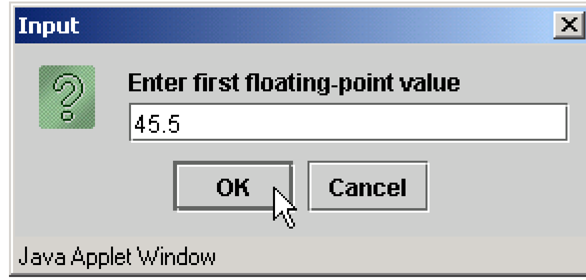
AdditionApplet.java (Part 2 of 2)

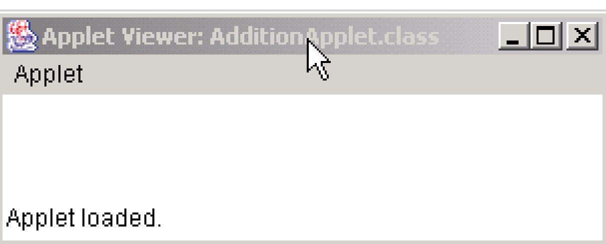
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

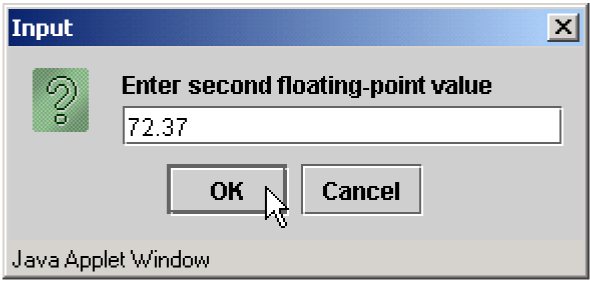
Outline

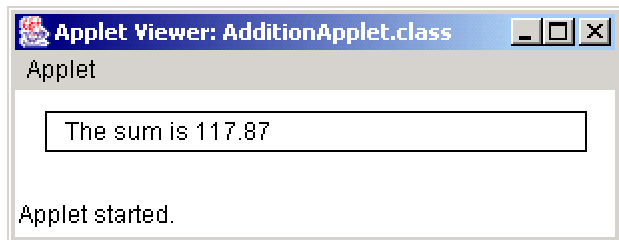
Program Output











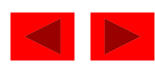
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Chapter 25 - Beyond C & C++: Operators, Methods, and Arrays in Java

Outline

25.1 Introduction
25.2 Primitive Data Types and Keywords
25.3 Logical Operators
25.4 Method Definitions
25.5 Java API Packages
25.6 Random Number Generation
25.7 Example: A Game of Chance
25.8 Methods of Class JApplet
25.9 Defining and Allocating Arrays
25.10 Examples Using Arrays
25.11 References and Reference Parameters
25.12 Multiple-Subscripted Arrays

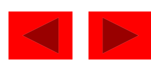
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Objectives

- In this chapter, you will learn:
  - To understand primitive types and logical operators as they are used in Java.
  - To introduce the common math methods available in the Java API.
  - To be able to create new methods.
  - To understand the mechanisms used to pass information between methods.
  - To introduce simulation techniques using random number generation.
  - To understand array objects in Java.
  - To understand how to write and use methods that call themselves.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



25.1 Introduction

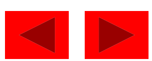
- In this chapter
  - Differences between C, C++, and Java
  - Java's logical operators and methods
  - Packages that comprise Applications Programming Interface (API)
  - Craps simulator
  - Random numbers in Java
  - Arrays in Java

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

25.2 Primitive Data Types and Keywords

- Primitive data types
  - `char`, `byte`, `short`, `int`, `long`, `float`, `double`, `boolean`
  - Building blocks for more complicated types
    - All variables must have a type before being used
    - Strongly typed language
  - Primitive types portable, unlike C and C++
    - In C/C++, write different versions of programs
      - Data types not guaranteed to be identical
      - `int`s may be 2 or 4 bytes, depending on system
    - WORA - Write once, run anywhere
  - Default values
    - `boolean` gets `false`, all other types are `0`

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

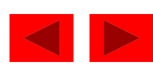


25.2 Primitive Data Types and Keywords (II)

Type	Size in bits	Values	Standard
boolean	8	true or false	
char	16	'\u0000' to '\uFFFF'	(ISO Unicode character set)
byte	8	–128 to +127	
short	16	–32,768 to +32,767	
int	32	–2,147,483,648 to +2,147,483,647	
long	64	–9,223,372,036,854,775,808 to +9,223,372,036,854,775,807	
float	32	–3.40292347E+38 to +3.40292347E+38	(IEEE 754 floating point)
double	64	–1.79769313486231570E+308 to +1.79769313486231570E+308	(IEEE 754 floating point)

Fig. 25.1 The Java primitive data types.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



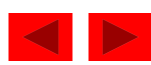
25.2 Primitive Data Types and Keywords (III)

- Keywords
  - Reserved names, cannot be used as identifiers
  - Used to implement features

Java Keywords				
abstract	boolean	break	Byte	case
catch	char	class	continue	default
do	double	else	Extends	false
final	finally	float	for	if
implements	import	instanceof	int	interface
long	native	new	null	package
private	protected	public	return	short
static	super	switch	synchronized	this
throw	throws	transient	true	try
void	volatile	while		
Keywords that are reserved but not used by Java				
const	goto			

Fig. 25.2 Java keywords.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

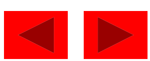


25.3 Logical Operators

- Logical operators
  - Form complex conditions and control structures
  - Logical AND (&&)
    - `true` if both conditions `true`
  - Logical OR (||)
    - `true` if either condition `true`
    - `true` if both conditions `true` (inclusive)
    - If left condition `true`, skips right condition
  - Boolean logical AND (**&**) , boolean logical inclusive OR (**|**)
    - Act like counterparts, but always evaluate both expressions
    - Useful if expression performs action:

```
birthday == true | ++age >= 65
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



25.3 Logical Operators (II)

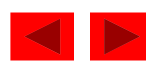
expression1	expression2	expression1 && expression2
fal se	fal se	Fal se
fal se	true	Fal se
true	fal se	Fal se
true	true	true

Fig. 25.3 Truth table for the && (logical AND) operator.

expression1	expression2	expression1 || expression2
fal se	fal se	fal se
fal se	true	true
true	fal se	true
true	true	true

Fig. 25.4 Truth table for the || (logical OR) operator.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

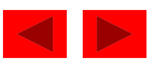


25.3 Logical Operators (III)

- Logical Operators (continued)
  - Boolean logical exclusive OR (^)
    - `true` if exactly one condition `true`
    - `false` if both conditions `true`
  - Logical NOT (negation)
    - Unary operator (one operand)
      - All other logical operators binary (two operands)
    - Reverses condition
    - If `true`, returns `false`
    - If `false`, returns `true`
    - `!=` - "does not equal"

```
if (grade != sentinelValue)
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



25.3 Logical Operators (IV)

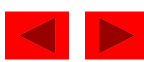
expression1	expression2	expression1 ^ expression2
false	false	false
false	true	true
true	false	true
true	true	false

Fig. 25.5 Truth table for the boolean logical exclusive OR (^) operator.

expression	! expression
false	true
true	false

Fig. 25.6 Truth table for operator ! (logical NOT).

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



25.3 Logical Operators (V)

- More GUI Classes (`javax.swing`)
  - `JTextArea`
    - Create an area where text can be displayed
    - Provide `(rows, columns)` to constructor to specify size

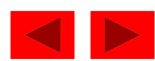
```
JTextArea myArea;   //declares object type
myArea = new JTextArea( 17, 20 ); //initialize
```

  - `myArea.setText( myString );`
    - Sets the text of `myArea` to `myString`
  - `JScrollPane`
    - Creates a window that can scroll

```
JScrollPane myScroller =
        new JScrollPane ( myArea );
```

    - Declaration and initialization, allows `myArea` to have scrolling

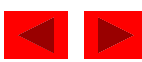
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

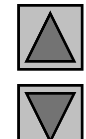


25.3 Logical Operators (VI)

- More GUI classes
  - `showMessageDialog(null, myScroller, titleString, type);`
    - Second argument indicates that `myScroller` (and attached `myArea`) should be displayed in message dialog

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.





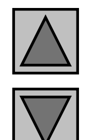
Outline

Logical-Operators.java (Part 1 of 2)

```
1  // Fig. 25.7: LogicalOperators.java
2  // Demonstrating the logical operators
3  import javax.swing.*;
4
5  public class LogicalOperators {
6     public static void main( String args[] )
7     {
8        JTextArea outputArea = new JTextArea( 17, 20 );
9        JScrollPane scroller = new JScrollPane( outputArea );
10        String output = "";
11
12        output += "Logical AND (&&)" +
13                  "\nfalse && false: " + ( false && false ) +
14                  "\nfalse && true: " + ( false && true ) +
15                  "\ntrue && false: " + ( true && false ) +
16                  "\ntrue && true: " + ( true && true );
17
18        output += "\n\nLogical OR (||)" +
19                  "\nfalse || false: " + ( false || false ) +
20                  "\nfalse || true: " + ( false || true ) +
21                  "\ntrue || false: " + ( true || false ) +
22                  "\ntrue || true: " + ( true || true );
23
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

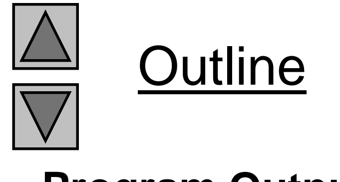
```
24        output += "\n\nBoolean logical AND (&)" +
25                  "\nfalse & false: " + ( false & false ) +
26                  "\nfalse & true: " + ( false & true ) +
27                  "\ntrue & false: " + ( true & false ) +
28                  "\ntrue & true: " + ( true & true );
29
30        output += "\n\nBoolean logical inclusive OR (|)" +
31                  "\nfalse | false: " + ( false | false ) +
32                  "\nfalse | true: " + ( false | true ) +
33                  "\ntrue | false: " + ( true | false ) +
34                  "\ntrue | true: " + ( true | true );
35
36        output += "\n\nBoolean logical exclusive OR (^)" +
37                  "\nfalse ^ false: " + ( false ^ false ) +
38                  "\nfalse ^ true: " + ( false ^ true ) +
39                  "\ntrue ^ false: " + ( true ^ false ) +
40                  "\ntrue ^ true: " + ( true ^ true );
41
42        output += "\n\nLogical NOT (!)" +
43                  "\n!false: " + ( !false ) +
44                  "\n!true: " + ( !true );
45
46        outputArea.setText( output );
47        JOptionPane.showMessageDialog( null, scroller,
48           "Truth Tables", JOptionPane.INFORMATION_MESSAGE );
49        System.exit( 0 );
50     } // end main
51  } // end class LogicalOperators
```



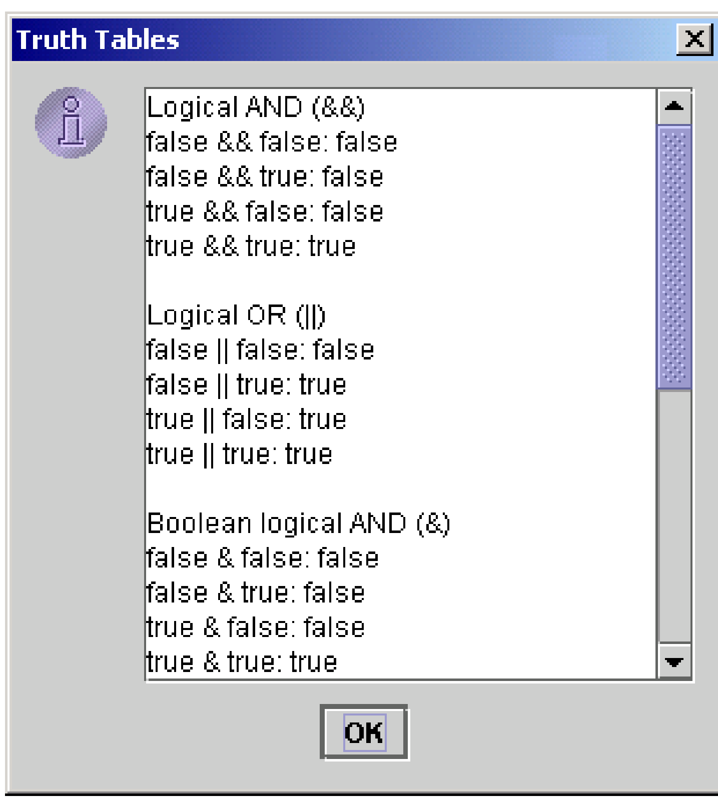
Outline

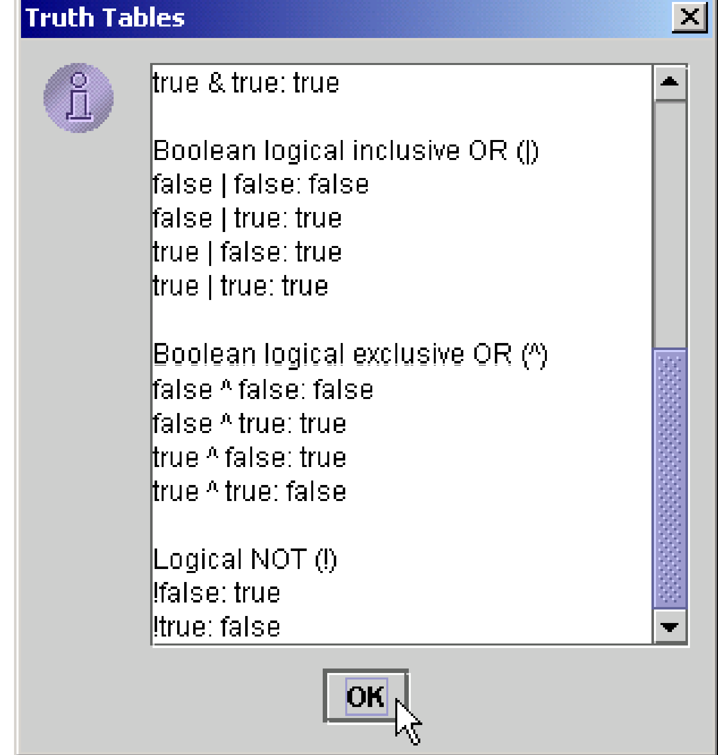
Logical-Operators.java (Part 2 of 2)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Program Output





© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

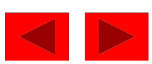
25.4 Method Definitions

- Method definition format

  return-value-type method-name(*parameter-list*)
  {
  declarations and statements
  }

  - Method-name: any valid identifier
  - Return-value-type: data type of the result
    - `void` - method returns nothing
    - Can return at most one value
  - Parameter-list: comma separated list, defines parameters
    - Method call must have proper number and type of parameters
  - Definitions and statements: method body (block)
    - Variables can be defined inside blocks (can be nested)
    - Method cannot be defined inside another function

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

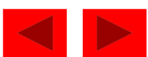


25.4 Method Definitions (II)

- Program control
  - When method call encountered
    - Control transferred from point of invocation to method
  - Returning control
    - If nothing returned: `return;`
      - Or until reaches right brace
    - If value returned: `return` *expression*;
      - Returns the value of *expression*
  - Example user-defined method:

```
public int square( int y )
 {
 return y * y
 }
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



25.4 Method Definitions (III)

- Calling methods
  - Three ways
    - Method name and arguments
      - Can be used by methods of same class
      - `square( 2 );`
    - Dot operator - used with objects
      - `g.drawLine( x1, y1, x2, y2 );`
    - Dot operator - used with `static` methods of classes
      - `Integer.parseInt( myString );`
      - More Chapter 26

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

25.4 Method Definitions (IV)

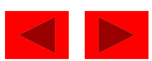
- More GUI components
  - Content Pane - on-screen display area
    - Attach GUI components to it to be displayed
    - Object of class `Container` (`java.awt`)
  - `getContentPane`
    - Method inherited from `JApplet`
    - Returns reference to Content Pane

```
Container c = getContentPane();
```

  - `Container` method `add`
    - Attaches GUI components to content pane, so they can be displayed
    - For now, only attach one component (occupies entire area)
    - Later, learn how to add and layout multiple components

```
c.add( myTextArea );
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Outline

SquareInt.java (Part 1 of 2)

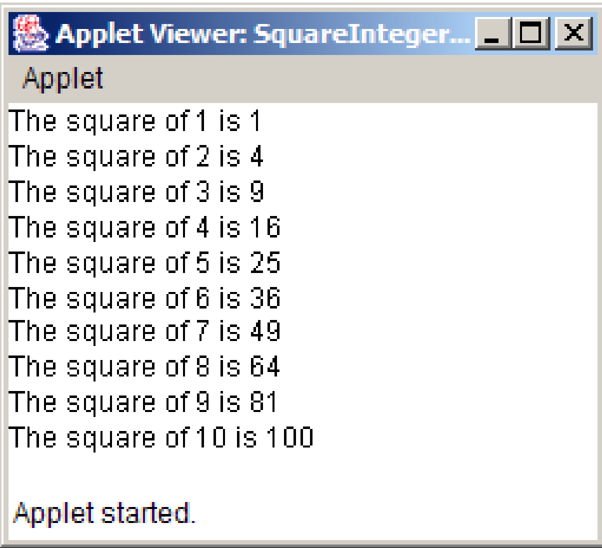
```
1  // Fig. 25.8: SquareInt.java
2  // A programmer-defined square method
3  import java.awt.Container;
4  import javax.swing.*;
5
6  public class SquareInt extends JApplet {
7     public void init()
8     {
9        String output = "";
10
11       JTextArea outputArea = new JTextArea( 10, 20 );
12
13       // get the applet's GUI component display area
14       Container c = getContentPane();
15
16       // attach outputArea to Container c
17       c.add( outputArea );
18
19       int result;
20
21       for ( int x = 1; x <= 10; x++ ) {
22          result = square( x );
23          output += "The square of " + x +
24                    " is " + result + "\n";
25       } // end for
26
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
27       outputArea.setText( output );
28    } // end method init
29
30    // square method definition
31    public int square( int y )
32    {
33       return y * y;
34    } // end method square
35 } // end class SquareInt
```

Outline

SquareInt.java (Part 2 of 2)



Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

25.5 Java API Packages

- As we have seen
  - Java has predefined, grouped classes called packages
  - Together, all the packages are the Applications Programming Interface (API)
  - Fig 25.10 has a list of the packages in the API
- Import
  - Import statements specify location of classes
  - Large number of classes, avoid reinventing the wheel

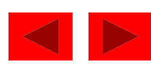
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

25.5 Java API Packages (II)

Type	Allowed promotions
double	None
float	Double
long	float or double
int	long, float or double
char	int, long, float or double
short	int, long, float or double
byte	short, int, long, float or double
boolean	None (boolean values are not considered to be numbers in Java)

Fig. 25.9 Allowed promotions for primitive data types.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

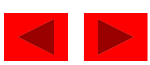


25.5 Java API Packages (III)

Package	Description
`java.applet`	*The Java Applet Package.* This package contains the `Applet` class and several interfaces that enable the creation of applets, interaction of applets with the browser and playing audio clips. In Java 2, class `javax.swing.JApplet` is used to define an applet that uses the *Swing GUI components*.
`java.awt`	*The Java Abstract Windowing Toolkit Package.* This package contains the classes and interfaces required to create and manipulate graphical user interfaces in Java 1.0 and 1.1. In Java 2, these classes can still be used, but the *Swing GUI components* of the `javax.swing` packages are often used instead.
`java.awt.color`	*The Java Color Space Package.* This package contains classes that support color spaces.
`java.awt.datatransfer`	*The Java Data Transfer Package.* This package contains classes and interfaces that enable transfer of data between a Java program and the computer's clipboard (a temporary storage area for data).
`java.awt.dnd`	*The Java Drag-and-Drop Package.* This package contains classes and interfaces that provide drag-and-drop support between programs.
`java.awt.event`	*The Java Abstract Windowing Toolkit Event Package.* This package contains classes and interfaces that enable event handling for GUI components in both the `java.awt` and `javax.swing` packages.

Fig. 25.10 Packages of the Java API. (Part 1 of 6)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

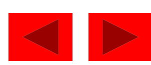


25.5 Java API Packages (IV)

Package	Description
`java.awt.font`	*The Java Font Manipulation Package.* This package contains classes and interfaces for manipulating many different fonts.
`java.awt.geom`	*The Java Two-Dimensional Objects Package.* This package contains classes for manipulating objects that represent two-dimensional graphics.
`java.awt.im`	*The Java Input Method Framework Package.* This package contains classes and an interface that support Japanese, Chinese and Korean language input into a Java program.
`java.awt.image` `java.awt.image.renderable`	*The Java Image Packages.* These packages contain classes and interfaces that enable storing and manipulation of images in a program.
`java.awt.print`	*The Java Printing Package.* This package contains classes and interfaces that support printing from Java programs.
`java.beans` `java.beans.beancontext`	*The Java Beans Packages.* These packages contain classes and interfaces that enable the programmer to create reusable software components.

Packages of the Java API. (Part 2 of 6)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



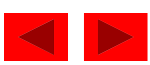
25.5 Java API Packages (V)

Package	Description
`java.io`	*The Java Input/Output Package.* This package contains classes that enable programs to input and output data.
`java.lang`	*The Java Language Package.* This package contains classes and interfaces required by many Java programs (many are discussed throughout the text) and is automatically imported by the compiler into all programs.
`java.lang.ref`	*The Reference Objects Package.* This package contains classes that enable interaction between a Java program and the garbage collector.
`java.lang.reflect`	*The Java Core Reflection Package.* This package contains classes and interfaces that enable a program to discover the accessible variables and methods of a class dynamically during the execution of a program.
`java.math`	*The Java Arbitrary Precision Math Package.* This package contains classes for performing arbitrary-precision arithmetic.
`java.net`	*The Java Networking Package.* This package contains classes that enable programs to communicate via networks.

`java.rmi`
`java.rmi.activation`
`java.rmi.dgc`
`java.rmi.registry`
`java.rmi.server`
	The Java Remote Method Invocation Packages. These packages contain classes and interfaces that enable the programmer to create distributed Java programs. Using remote method invocation, a program can call a method of a separate program on the same computer or on a computer anywhere on the Internet.

Packages of the Java API. (Part 3 of 6)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

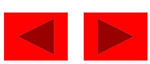


25.5 Java API Packages (VI)

Package	Description
java.security java.security.acl java.security.cert java.security.interfaces java.security.spec	*The Java Security Packages.* These packages contains classes and interfaces that enable a Java program to encrypt data and control the access privileges provided to a Java program for security purposes.
java.sql	*The Java Database Connectivity Package.* This package contain classes and interfaces that enable a Java program to interact with a database.
java.text	*The Java Text Package.* This package contains classes and interfaces that enable a Java program to manipulate numbers, dates, characters and strings. This package provides many of Java's internationalization capabilities—features that enable a program to be customized to a specific locale (e.g., an applet may display strings in different languages based on the browser in which it is executing).
java.util	*The Java Utilities Package.* This package contains utility classes and interfaces such as: date and time manipulations, random number processing capabilities (Random), storing and processing large amounts of data, breaking strings into smaller pieces called tokens (StringTokenizer) and other capabilities.
java.util.jar java.util.zip	*The Java Utilities JAR and ZIP Packages.* These packages contain utility classes and interfaces that enable a Java program to combine Java .class files and other resource files (such as images and audio) into compressed file called *Java archive (JAR) files* or *ZIP files*.
javax.accessibility	*The Java Accessibility Package.* This package contains classes and interfaces that allow a Java program to support technologies for people with disabilities; examples are screen readers and screen magnifiers.

Packages of the Java API. (Part 4 of 6)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



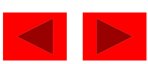
25.5 Java API Packages (VII)

Package	Description
`javax.swing`	*The Java Swing GUI Components Package.* This package contains classes and interfaces for Java’s Swing GUI components that provide support for portable GUIs.
`javax.swing.border`	*The Java Swing Borders Package.* This package contains classes and an interface for drawing borders around areas in a GUI.
`javax.swing.colorchooser`	*The Java Swing Color Chooser Package.* This package contains classes and interfaces for the `JColorChooser` predefined dialog for choosing colors.
`javax.swing.event`	*The Java Swing Event Package.* This package contains classes and interfaces that enable event handling for GUI components in the `javax.swing` package.
`javax.swing.filechooser`	*The Java Swing File Chooser Package.* This package contains classes and interfaces for the `JFileChooser` predefined dialog for locating files on disk.

`javax.swing.plaf`
`javax.swing.plaf.basic`
`javax.swing.plaf.metal`
`javax.swing.plaf.multi`
	The Java Swing Pluggable-Look-and-Feel Packages.
	These packages contain classes and an interface used to change the look-and-feel of a Swing-based GUI between the Java look-and-feel, Microsoft Windows look-and-feel and the UNIX Motif look-and-feel. The package also supports development of a customized look-and-feel for a Java program.

Packages of the Java API. (Part 5 of 6)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

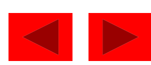


25.5 Java API Packages (VIII)

Package	Description
javax.swing.text.html javax.swing.text.html.parser	*The Java Swing HTML Text Packages.* These packages contain classes that provide support for building HTML text editors.
javax.swing.text.rtf	*The Java Swing RTF Text Package.* This package contains a class that provides support for building editors that support rich-text formatting.
javax.swing.tree	*The Java Swing Tree Package.* This package contains classes and interfaces for creating and manipulating expanding tree GUI components.
javax.swing.undo	*The Java Swing Undo Package.* This package contains classes and interfaces that support providing undo and redo capabilities in a Java program.
org.omg.CORBA org.omg.CORBA.DynAnyPackage org.omg.CORBA.ORBPackage org.omg.CORBA.portable org.omg.CORBA.TypeCodePackage org.omg.CosNaming org.omg.CosNaming.NamingContextPackage	*The Object Management Group (OMG) CORBA Packages.* These packages contain classes and interfaces that implement OMG's CORBA APIs that allow a Java program to communicate with programs written in other programming languages in a similar fashion to using Java's RMI packages to communicate between Java programs.

Packages of the Java API. (Part 6 of 6)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



25.6 Random Number Generation

- `Math.random()`
  – Returns a random `double`, greater than or equal to `0.0`, less than `1.0`
- Scaling and shifting

```
n = a + (int) ( Math.random() * b )
```

  `n` = random number

  `a` = shifting value

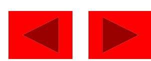
  `b` = scaling value

  In C we used %, but in Java we can use *

For a random number between 1 and 6,

```
n = 1 + (int) ( Math.random() * 6 )
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



RandomInt.java

```java
1  // Fig. 25.11: RandomInt.java
2  // Shifted, scaled random integers
3  import javax.swing.JOptionPane;
4
5  public class RandomInt {
6     public static void main( String args[] )
7     {
8        int value;
9        String output = "";
10
11        for ( int i = 1; i <= 20; i++ ) {
12           value = 1 + (int) ( Math.random() * 6 );
13           output += value + "  ";
14
15           if ( i % 5 == 0 )
16              output += "\n";
17        }
18
19        JOptionPane.showMessageDialog( null, output,
20           "20 Random Numbers from 1 to 6",
21           JOptionPane.INFORMATION_MESSAGE );
22
23        System.exit( 0 );
24     } // end main
25  } // end class RandomInt
```

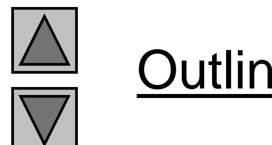
Program Output

20 Random Numbers from 1 to 6

```
5 1 1 6 5
5 6 4 6 1
1 4 1 5 6
1 5 2 5 1
```

OK

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Outline

RollDie.java (Part 1 of 2)

```
1  // Fig. 25.12: RollDie.java
2  // Roll a six-sided die 6000 times
3  import javax.swing.*;
4
5  public class RollDie {
6     public static void main( String args[] )
7     {
8        int frequency1 = 0, frequency2 = 0,
9            frequency3 = 0, frequency4 = 0,
10           frequency5 = 0, frequency6 = 0, face;
11
12       // summarize results
13       for ( int roll = 1; roll <= 6000; roll++ ) {
14          face = 1 + (int) ( Math.random() * 6 );
15
16          switch ( face ) {
17             case 1:
18                ++frequency1;
19                break;
20             case 2:
21                ++frequency2;
22                break;
23             case 3:
24                ++frequency3;
25                break;
26             case 4:
27                ++frequency4;
28                break;
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
29             case 5:
30                ++frequency5;
31                break;
32             case 6:
33                ++frequency6;
34                break;
35          } // end switch
36       } // end for
37
38       JTextArea outputArea = new JTextArea( 7, 10 );
39
40       outputArea.setText(
41          "Face\tFrequency" +
42          "\n1\t" + frequency1 +
43          "\n2\t" + frequency2 +
44          "\n3\t" + frequency3 +
45          "\n4\t" + frequency4 +
46          "\n5\t" + frequency5 +
47          "\n6\t" + frequency6 );
48
49       JOptionPane.showMessageDialog( null, outputArea,
50          "Rolling a Die 6000 Times",
51          JOptionPane.INFORMATION_MESSAGE );
52       System.exit( 0 );
53    } // end main
54 } // end class RollDie
```

Outline

RollDie.java (Part 1 of 2)

Program Output

Rolling a Die 6000 Times

Face	Frequency
1	1023
2	982
3	993
4	1000
5	1034
6	968

OK

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

25.7 Example: A Game of Chance

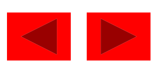
- Redo "craps" simulator from Chapter 5
- Rules
  - Roll two dice
    - 7 or 11 on first throw, player wins
    - 2, 3, or 12 on first throw, player loses
    - 4, 5, 6, 8, 9, 10 - value becomes player's "point"
  - player must roll his point before rolling 7 to win

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

25.7 Example: A Game of Chance (II)

- User input
  - Till now, used message dialog and input dialog
    - Tedious, only show one message/ get one input at a time
  - Now, we will use event handling for more complex GUI
- `extends` keyword
  - Class inherits data and methods from another class
  - A class can also implement an interface
    - Keyword `implements`
    - Interface - specifies methods you must define in your class
- Event handling
  - Event: user interaction (i.e., user clicking a button)
  - Event handler: method called in response to an event

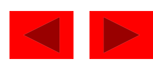
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



25.7 Example: A Game of Chance (III)

- Interface `ActionListener`
  - Requires that you define method `actionPerformed`
    - `actionPerformed` is the event handler
- Class `JTextField`
  - Can display or input a line of text
- Class `JButton`
  - Displays a button which can perform an action if pushed
  - Method `addActionListener( this );`
    - Specifies this applet should listen for events from the `JButton` object
  - Each component must know which method will handle its events
    - Registering the event handler

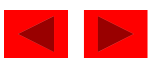
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



25.7 Example: A Game of Chance (IV)

- Class `JButton` (continued)
  - We registered `this` applet with our `JButton`
    - The applet "listens" for events from the
  - `actionPerformed` is the event handler
- Event-driven programming
  - User's interaction with GUI drives program
- `final`
  - Defines a variable constant
    - Cannot be modified
    - Must be initialized at definition
    - `const int MYINT = 3;`
    - Use all uppercase for `final` variables

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



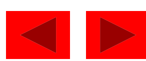
25.7 Example: A Game of Chance (V)

- Methods of class `Container`
  - Recall that the Content Pane is of class `Container`
  - Method `setLayout`
    - Define layout managers (determine position and size of all components attached to container)
    - `FlowLayout` - Most basic layout manager
      - Items placed left to right in order added to container
      - When end of line reached, continues on next line

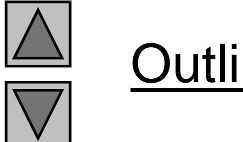
```
c = getContentPane();
c.setLayout( new FlowLayout() );
```

Initialized with object of class `FlowLayout`

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



```
1  // Fig. 25.13: Craps.java
2  // Craps
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class Craps extends JApplet implements ActionListener {
8     // constant variables for status of game
9     final int WON = 0, LOST = 1, CONTINUE = 2;
10
11     // other variables used in program
12     boolean firstRoll = true;    // true if first roll
13     int sumOfDice = 0;           // sum of the dice
14     int myPoint = 0;    // point if no win/loss on first roll
15     int gameStatus = CONTINUE;   // game not over yet
16
17     // graphical user interface components
18     JLabel die1Label, die2Label, sumLabel, pointLabel;
19     JTextField firstDie, secondDie, sum, point;
20     JButton roll;
21
```

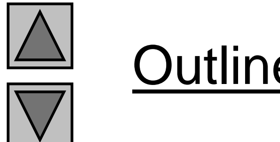


Outline

Craps.java (Pat 1 of 5)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
22    // setup graphical user interface components
23    public void init()
24    {
25       Container c = getContentPane();
26       c.setLayout( new FlowLayout() );
27
28       die1Label = new JLabel( "Die 1" );
29       c.add( die1Label );
30       firstDie = new JTextField( 10 );
31       firstDie.setEditable( false );
32       c.add( firstDie );
33
34       die2Label = new JLabel( "Die 2" );
35       c.add( die2Label );
36       secondDie = new JTextField( 10 );
37       secondDie.setEditable( false );
38       c.add( secondDie );
39
40       sumLabel = new JLabel( "Sum is" );
41       c.add( sumLabel );
42       sum = new JTextField( 10 );
43       sum.setEditable( false );
44       c.add( sum );
45
```



Craps.java (Pat 2 of 5)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
46         pointLabel = new JLabel( "Point is" );
47         c.add( pointLabel );
48         point = new JTextField( 10 );
49         point.setEditable( false );
50         c.add( point );
51
52         roll = new JButton( "Roll Dice" );
53         roll.addActionListener( this );
54         c.add( roll );
55      } // end method init
56
57      // call method play when button is pressed
58      public void actionPerformed( ActionEvent e )
59      {
60         play();
61      } // end method actionPerformed
62
63      // process one roll of the dice
64      public void play()
65      {
66         if ( firstRoll ) {             // first roll of the dice
67            sumOfDice = rollDice();
68
69            switch ( sumOfDice ) {
70               case 7: case 11:         // win on first roll
71                  gameStatus = WON;
72                  point.setText( "" );  // clear point text field
73                  break;
```

Outline

Craps.java (Pat 3 of 5)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

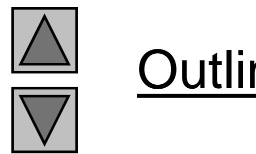
```
74             case 2: case 3: case 12: // lose on first roll
75                gameStatus = LOST;
76                point.setText( "" );  // clear point text field
77                break;
78             default:                 // remember point
79                gameStatus = CONTINUE;
80                myPoint = sumOfDice;
81                point.setText( Integer.toString( myPoint ) );
82                firstRoll = false;
83                break;
84          } // end switch
85       } // end if
86       else {
87          sumOfDice = rollDice();
88
89          if ( sumOfDice == myPoint )    // win by making point
90             gameStatus = WON;
91          else
92             if ( sumOfDice == 7 )       // lose by rolling 7
93                gameStatus = LOST;
94       } // end else
95
96       if ( gameStatus == CONTINUE )
97          showStatus( "Roll again." );
98       else {
99          if ( gameStatus == WON )
100            showStatus( "Player wins. " +
101               "Click Roll Dice to play again." );
```

Outline

Craps.java (Pat 4 of 5)

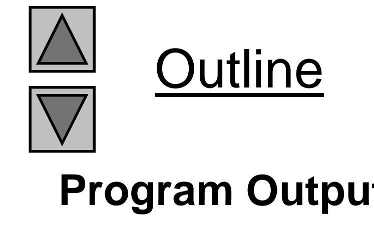
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

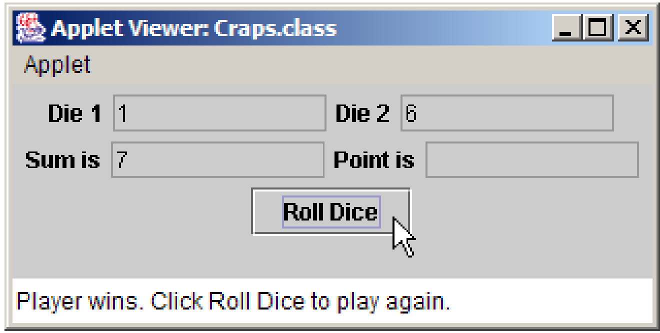
```
102          else
103             showStatus( "Player loses. " +
104                "Click Roll Dice to play again." );
105
106          firstRoll = true;
107       } // end else
108    } // end method play
109
110    // roll the dice
111    public int rollDice()
112    {
113       int die1, die2, workSum;
114
115       die1 = 1 + ( int ) ( Math.random() * 6 );
116       die2 = 1 + ( int ) ( Math.random() * 6 );
117       workSum = die1 + die2;
118
119       firstDie.setText( Integer.toString( die1 ) );
120       secondDie.setText( Integer.toString( die2 ) );
121       sum.setText( Integer.toString( workSum ) );
122
123       return workSum;
124    } // end method rollDice
125 } // end class Craps
```

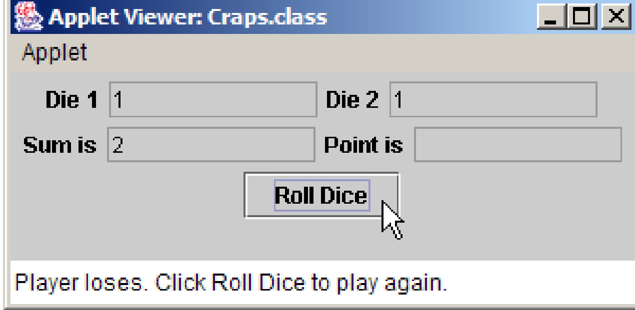


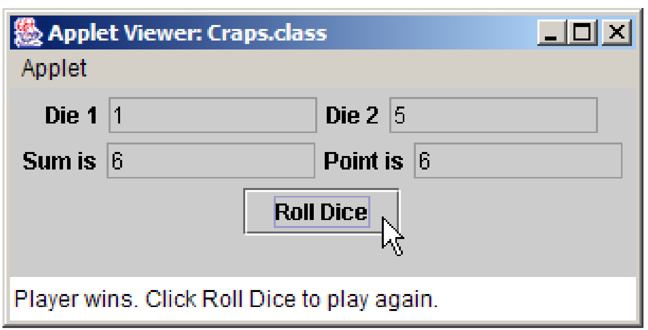
Craps.java (Pat 5 of 5)

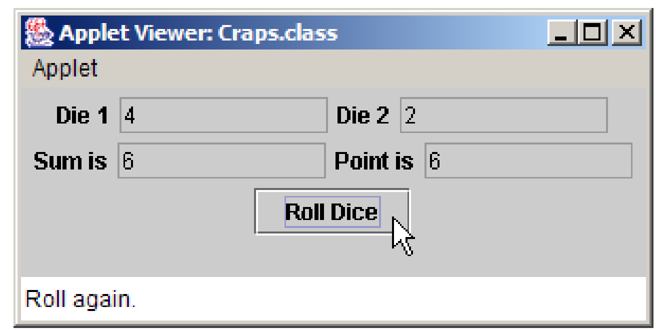
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

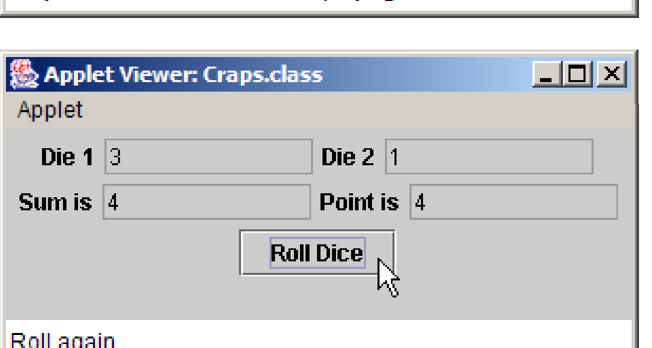


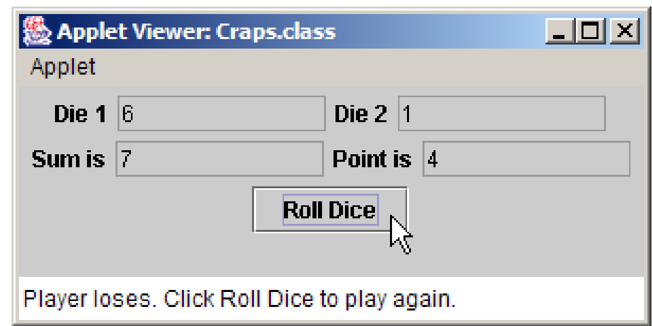










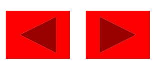


© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

25.8 Methods of Class `JApplet`

- Methods of Class `JApplet`
  - `init`, `start`, `stop`, `paint`, `destroy`
  - Called automatically during execution
  - By default, have empty bodies
  - Must define yourself, using proper first line
    - Otherwise, will not be called automatically
    - See Figure 25.14 for proper first lines
- Method `repaint`
  - Dynamically change appearance of applet
  - Cannot call `paint` (do not have a `Graphics` object)
  - `repaint();` calls `update` which passes `Graphics` object for us
    - Erases previous drawings and calls `paint`

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

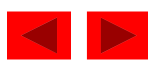


25.8 Methods of Class `JApplet` (II)

First line of `JApplet` methods (descriptions Fig. 25.14)

```
public void init()
public void start()
public void paint( Graphics g )
public void stop()
public void destroy()
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



25.9 Defining and Allocating Arrays

- Arrays
  - Specify type, use `new` operator
  - Two steps:

```
int c[];                //definition
c = new int[ 12 ];  //initialization
```

  - One step:

```
int c[] = new int[12];
```

  - Primitive elements initialized to `zero` or `false`
    - Non-primitive references are `null`
  - Multiple definitions:

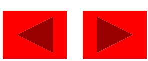
```
String b[] = new String[ 100 ], x[] = new String[ 27 ];
```

Also:

```
double[] array1, array2;
```

Put brackets after data type

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



25.10 Examples Using Arrays

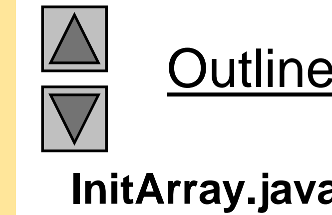
- `new`
  - Dynamically creates arrays
- Method `length`
  - Returns `length` of the array

    `myArray.length`
- Initializer lists

```
int myArray[] = { 1, 2, 3, 4, 5 };
```

  - `new` operator not needed, provided automatically

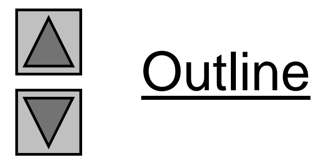
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



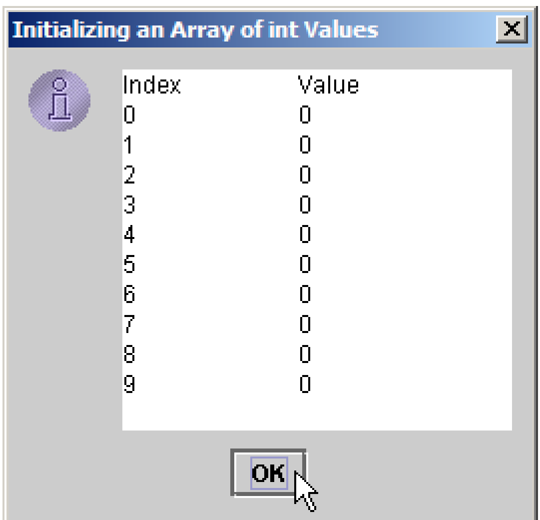
InitArray.java

```
1  // Fig. 25.15: InitArray.java
2  // initializing an array
3  import javax.swing.*;
4
5  public class InitArray {
6     public static void main( String args[] )
7     {
8        String output = "";
9        int n[];              // declare reference to an array
10
11        n = new int[ 10 ];  // dynamically allocate array
12
13        output += "Subscript\tValue\n";
14
15        for ( int i = 0; i < n.length; i++ )
16           output += i + "\t" + n[ i ] + "\n";
17
18        JTextArea outputArea = new JTextArea( 11, 10 );
19        outputArea.setText( output );
20
21        JOptionPane.showMessageDialog( null, outputArea,
22           "Initializing an Array of int Values",
23           JOptionPane.INFORMATION_MESSAGE );
24
25        System.exit( 0 );
26     } // end main
27  } // end class InitArray
```

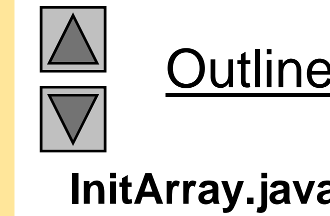
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Program Output



© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

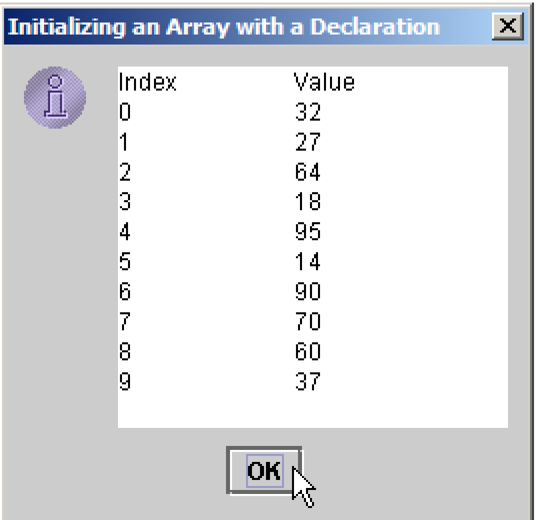


InitArray.java

```
1  // Fig. 25.16: InitArray.java
2  // initializing an array with a declaration
3  import javax.swing.*;
4
5  public class InitArray {
6     public static void main( String args[] )
7     {
8        String output = "";
9
10        // Initializer list specifies number of elements and
11        // value for each element.
12        int n[] = { 32, 27, 64, 18, 95, 14, 90, 70, 60, 37 };
13
14        output += "Subscript\tValue\n";
15
16        for ( int i = 0; i < n.length; i++ )
17           output += i + "\t" + n[ i ] + "\n";
18
19        JTextArea outputArea = new JTextArea( 11, 10 );
20        outputArea.setText( output );
21
22        JOptionPane.showMessageDialog( null, outputArea,
23           "Initializing an Array with a Declaration",
24           JOptionPane.INFORMATION_MESSAGE );
25
26        System.exit( 0 );
27     } // end main
28  } // end class InitArray
```

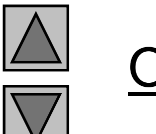
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Program Output



© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
1  // Fig. 25.17: InitArray.java
2  // initialize array n to the even integers from 2 to 20
3  import javax.swing.*;
4
5  public class InitArray {
6     public static void main( String args[] )
7     {
8        final int ARRAY_SIZE = 10;
9        int n[];                        // reference to int array
10       String output = "";
11
12       n = new int[ ARRAY_SIZE ];  // allocate array
13
14       // Set the values in the array
15       for ( int i = 0; i < n.length; i++ )
16          n[ i ] = 2 + 2 * i;
17
18       output += "Subscript\tValue\n";
19
20       for ( int i = 0; i < n.length; i++ )
21          output += i + "\t" + n[ i ] + "\n";
22
23       JTextArea outputArea = new JTextArea( 11, 10 );
24       outputArea.setText( output );
25
```



Outline

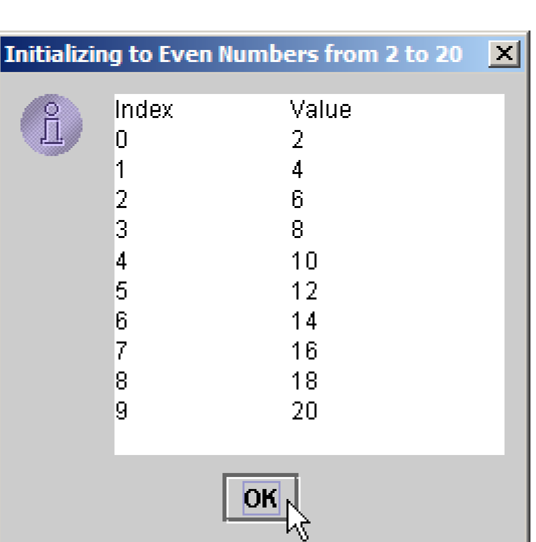
InitArray.java (Part 1 of 2)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
26        JOptionPane.showMessageDialog( null, outputArea,
27           "Initializing to Even Numbers from 2 to 20",
28           JOptionPane.INFORMATION_MESSAGE );
29
30        System.exit( 0 );
31     } // end main
32  } // end class InitArray
```

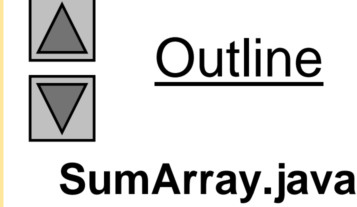
Outline

InitArray.java (Part 2 of 2)



Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



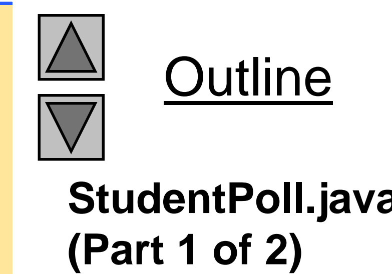
SumArray.java

```
1  // Fig. 25.18: SumArray.java
2  // Compute the sum of the elements of the array
3  import javax.swing.*;
4
5  public class SumArray {
6     public static void main( String args[] )
7     {
8        int a[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
9        int total = 0;
10
11       for ( int i = 0; i < a.length; i++ )
12          total += a[ i ];
13
14       JOptionPane.showMessageDialog( null,
15          "Total of array elements: " + total,
16          "Sum the Elements of an Array",
17          JOptionPane.INFORMATION_MESSAGE );
18
19       System.exit( 0 );
20    } // end main
21 } // end class SumArray
```

Program Output

```
Sum the Elements of an Array
Total of array elements: 55
OK
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



StudentPoll.java (Part 1 of 2)

```
1  // Fig. 25.19: StudentPoll.java
2  // Student poll program
3  import javax.swing.*;
4
5  public class StudentPoll {
6     public static void main( String args[] )
7     {
8        int responses[] = { 1, 2, 6, 4, 8, 5, 9, 7, 8, 10,
9                            1, 6, 3, 8, 6, 10, 3, 8, 2, 7,
10                           6, 5, 7, 6, 8, 6, 7, 5, 6, 6,
11                           5, 6, 7, 5, 6, 4, 8, 6, 8, 10 };
12       int frequency[] = new int[ 11 ];
13       String output = "";
14
15       for ( int answer = 0;                 // initialize
16             answer < responses.length;      // condition
17             answer++ )                      // increment
18          ++frequency[ responses[ answer ] ];
19
20       output += "Rating\tFrequency\n";
21
22       for ( int rating = 1;
23             rating < frequency.length;
24             rating++ )
25          output += rating + "\t" + frequency[ rating ] + "\n";
26
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
27       JTextArea outputArea = new JTextArea( 11, 10 );
28       outputArea.setText( output );
29
30       JOptionPane.showMessageDialog( null, outputArea,
31          "Student Poll Program",
32          JOptionPane.INFORMATION_MESSAGE );
33
34       System.exit( 0 );
35    } // end main
36 } // end class StudentPoll
```

Outline

StudentPoll.java (Part 2 of 2)

Program Output

Student Poll Program

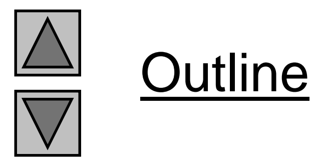
Rating	Frequency
1	2
2	2
3	2
4	2
5	5
6	11
7	5
8	7
9	1
10	3

OK

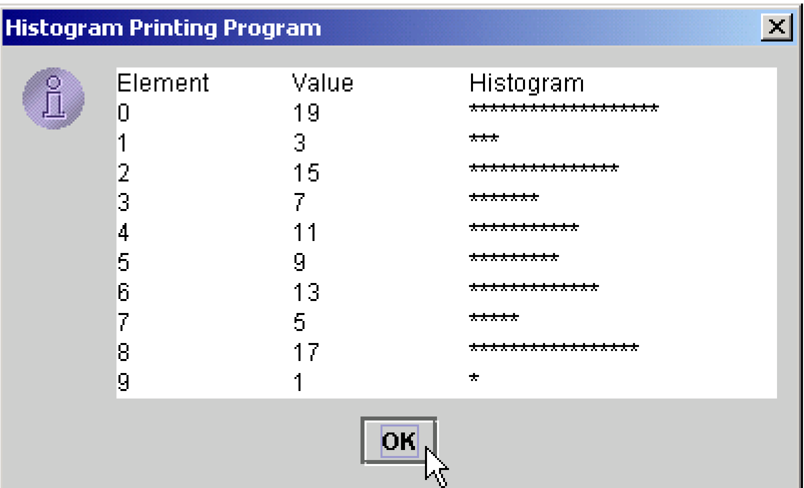
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Histogram.java

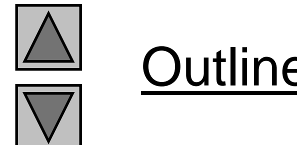
```
1  // Fig. 25.20: Histogram.java
2  // Histogram printing program
3  import javax.swing.*;
4
5  public class Histogram {
6     public static void main( String args[] )
7     {
8        int n[] = { 19, 3, 15, 7, 11, 9, 13, 5, 17, 1 };
9        String output = "";
10
11       output += "Element\tValue\tHistogram";
12
13       for ( int i = 0; i < n.length; i++ ) {
14          output += "\n" + i + "\t" + n[ i ] + "\t";
15
16          for ( int j = 1; j <= n[ i ]; j++ ) // print a bar
17             output += "*";
18       } // end for
19
20       JTextArea outputArea = new JTextArea( 11, 30 );
21       outputArea.setText( output );
22
23       JOptionPane.showMessageDialog( null, outputArea,
24          "Histogram Printing Program",
25          JOptionPane.INFORMATION_MESSAGE );
26
27       System.exit( 0 );
28    } // end main
29 } // end class Histogram
```



Program Output



© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



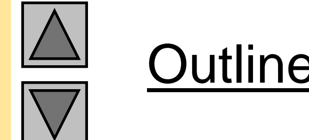
Outline

RollDie.java (Part 1 of 2)

```
1  // Fig. 25.21: RollDie.java
2  // Roll a six-sided die 6000 times
3  import javax.swing.*;
4
5  public class RollDie {
6     public static void main( String args[] )
7     {
8        int face, frequency[] = new int[ 7 ];
9        String output = "";
10
11        for ( int roll = 1; roll <= 6000; roll++ ) {
12           face = 1 + ( int ) ( Math.random() * 6 );
13           ++frequency[ face ];
14        }
15
16        output += "Face\tFrequency";
17
18        for ( face = 1; face < frequency.length; face++ )
19           output += "\n" + face + "\t" + frequency[ face ];
20
21        JTextArea outputArea = new JTextArea( 7, 10 );
22        outputArea.setText( output );
23
```

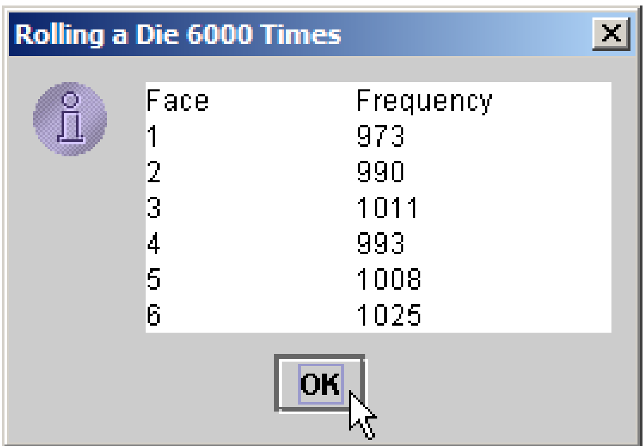
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
24         JOptionPane.showMessageDialog( null, outputArea,
25            "Rolling a Die 6000 Times",
26            JOptionPane.INFORMATION_MESSAGE );
27
28         System.exit( 0 );
29      } // end main
30   } // end class RollDie
```



RollDie.java (Part 2 of 2)

Program Output

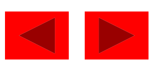


© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

25.11 References and Reference Parameters

- Passing arguments to methods
  - Call-by-value: pass copy of argument
  - Call-by-reference: pass original argument
    - Improve performance, weaken security
- In Java, cannot choose how to pass arguments
  - Primitive data types passed call-by-value
  - References to objects passed call-by-reference
    - Original object can be changed in method
  - Arrays in Java treated as objects
    - Passed call-by-reference

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



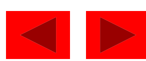
25.12 Multiple-Subscripted Arrays

- Multiple-Subscripted Arrays
  - Represent tables
    - Arranged by *m* rows and *n* columns (*m* by *n* array)
    - Can have more than two subscripts
  - Java does not support multiple subscripts directly
    - Create an array with arrays as its elements
    - Array of arrays
- Definition
  - Double brackets

```
int b[][];
b = new int[ 3 ][ 3 ];
```

    - Defines a 3 by 3 array

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



25.12 Multiple-Subscripted Arrays (II)

- Definition (continued)
  - Initializer lists

```
int b[][] = { { 1, 2 }, { 3, 4 } };
```

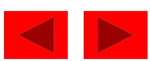
1	2
3	4

  - Each row can have a different number of columns:

```
int b[][];
b = new int[ 2 ][ ];    // allocate rows
b[ 0 ] = new int[ 5 ]; // allocate columns for row 0
b[ 1 ] = new int[ 3 ]; // allocate columns for row 1
```

  - Notice how **`b[ 0 ]`** is initialized as a new **`int`** array

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Outline

InitArray.java (Part 1 of 2)

```
1  // Fig. 25.23: InitArray.java
2  // Initializing multidimensional arrays
3  import java.awt.Container;
4  import javax.swing.*;
5
6  public class InitArray extends JApplet {
7     JTextArea outputArea;
8
9     // initialize the applet
10     public void init()
11     {
12        outputArea = new JTextArea();
13        Container c = getContentPane();
14        c.add( outputArea );
15
16        int array1[][] = { { 1, 2, 3 }, { 4, 5, 6 } };
17        int array2[][] = { { 1, 2 }, { 3 }, { 4, 5, 6 } };
18
19        outputArea.setText( "Values in array1 by row are\n" );
20        buildOutput( array1 );
21
22        outputArea.append( "\nValues in array2 by row are\n" );
23        buildOutput( array2 );
24     } // end method init
25
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

InitArray.java (Part 2 of 2)

```
26    public void buildOutput( int a[][] )
27    {
28       for ( int i = 0; i < a.length; i++ ) {
29
30          for ( int j = 0; j < a[ i ].length; j++ )
31             outputArea.append( a[ i ][ j ] + "  " );
32
33          outputArea.append( "\n" );
34       } // end for
35    } // end method buildOutput
36 } // end class InitArray
```

Program Output

```
Applet Viewer: InitArray.class
Applet
Values in array1 by row are
1  2  3
4  5  6

Values in array2 by row are
1  2
3
4  5  6

Applet started.
```

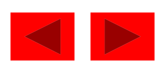
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Chapter 26 - Java Object-Based Programming

Outline

26.1 Introduction
26.2 Implementing a Time Abstract Data Type with a Class
26.3 Class Scope
26.4 Creating Packages
26.5 Initializing Class Objects: Constructors
26.6 Using Set and Get Methods
26.7 Using the this Reference
26.8 Finalizers
26.9 Static Class Members

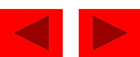
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Objectives

- In this chapter, you will learn:
  - To understand encapsulation and data hiding.
  - To understand the notions of data abstraction and abstract data types (ADTs).
  - To create Java ADTs, namely classes.
  - To be able to create, use and destroy objects.
  - To be able to control access to object instance variables and methods.
  - To appreciate the value of object orientation.
  - To understand the use of the `this` reference.
  - To understand class variables and class methods.

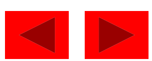
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



26.1 Introduction

- Object-oriented programming (OOP)
  - *Encapsulates* data (attributes) and functions (behavior) into packages called *classes*
  - Data and functions closely related
- Information hiding
  - Implementation details are hidden within the classes themselves
- Unit of Java programming: the class
  - A class is like a blueprint – reusable
  - Objects are *instantiated* (created) from the class
  - For example, a house is an instance of a "blueprint class"
  - C programmers concentrate on functions

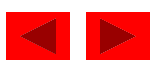
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



26.2 Implementing a Time Abstract Data Type with a Class

- In our example
  - Define two classes, `Time1` and `TimeTest` in separate files
    - Only one `public` class per file
- Class definitions
  - Never really create definition from scratch
    - Use `extends` to inherit data and methods from base class
    - Derived class: class that inherits
  - Every class in Java subclass of `Object`
    - Gets useful methods, discussed later
  - Class body
    - Delineated by braces `{ }`
    - Define instance variables and methods

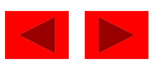
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



26.2 Implementing a Time Abstract Data Type with a Class (II)

- Member-access modifiers
  - `public`: accessible whenever program has a reference to an object of the class
  - `private`: accessible only to member methods of that class
  - Member variables are usually `private`
- Methods
  - Access methods: `public` methods that read/display data
    - `public` interface
    - Clients use references to interact with objects
  - Utility methods: `private` methods that support access methods

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



26.2 Implementing a Time Abstract Data Type with a Class (II)

- Constructor
  - Special member function
    - Same name as the class
  - Initializes data members of a class object
  - Constructors cannot return values
- Definitions
  - Once class defined, can be used as a data type
  - Define objects of the class

```
Time1 myTimeObject = new myTimeObject;
```

    - Defines object, initializes with constructor

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

26.2 Implementing a Time Abstract Data Type with a Class (III)

- `import`
  - If no package specified for class, class put in default package
    - Includes compiled classes of current directory
  - If class in same package as another, `import` not required
  - `import` when classes not of same package
- Classes simplify programming
  - Client only concerned with `public` operations
  - Client not dependent on implementation details
    - If implementation changes, client unaffected
  - Software reuse

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

26.2 Implementing a Time Abstract Data Type with a Class (IV)

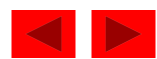
- Method `toString`
  - Class `Object`
  - Takes no arguments, returns a `String`
  - Used as a placeholder, usually overridden
- Class `DecimalFormat (java.text)`
  - Create object of class, initialize with format control string

```
DecimalFormat twoDigits = new DecimalFormat( "00" );
```

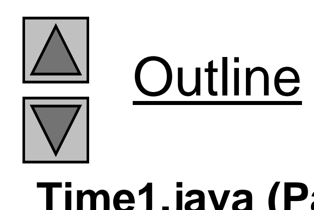
  - Each `0` is a placeholder for a digit
    - Prints in form `08, 10, 15`...
  - Method `format` returns `String` with proper formatting

```
twoDigits.format( myInt );
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



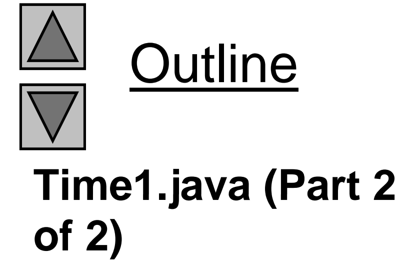
```
1  // Fig. 26.1: Time1.java
2  // Time1 class definition
3  import java.text.DecimalFormat;  // used for number formatting
4
5  // This class maintains the time in 24-hour format
6  public class Time1 extends Object {
7     private int hour;      // 0 - 23
8     private int minute;    // 0 - 59
9     private int second;    // 0 - 59
10
11     // Time1 constructor initializes each instance variable
12     // to zero. Ensures that each Time1 object starts in a
13     // consistent state.
14     public Time1()
15     {
16        setTime( 0, 0, 0 );
17     } // end Time1 constructor
18
19     // Set a new time value using universal time. Perform
20     // validity checks on the data. Set invalid values to zero.
21     public void setTime( int h, int m, int s )
22     {
23        hour = ( ( h >= 0 && h < 24 ) ? h : 0 );
24        minute = ( ( m >= 0 && m < 60 ) ? m : 0 );
25        second = ( ( s >= 0 && s < 60 ) ? s : 0 );
26     } // end method setTime
27
```



Time1.java (Part 1 of 2)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

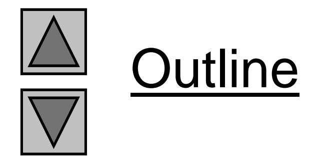
```
28    // Convert to String in universal-time format
29    public String toUniversalString()
30    {
31       DecimalFormat twoDigits = new DecimalFormat( "00" );
32
33       return twoDigits.format( hour ) + ":" +
34              twoDigits.format( minute ) + ":" +
35              twoDigits.format( second );
36    } // end method toUniversalString
37
38    // Convert to String in standard-time format
39    public String toString()
40    {
41       DecimalFormat twoDigits = new DecimalFormat( "00" );
42
43       return ( ( hour == 12 || hour == 0 ) ? 12 : hour % 12 ) +
44              ":" + twoDigits.format( minute ) +
45              ":" + twoDigits.format( second ) +
46              ( hour < 12 ? " AM" : " PM" );
47    } // end method toString
48 } // end class Time1
```



Time1.java (Part 2 of 2)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
49 // Fig. 26.1: TimeTest.java
50 // Class TimeTest to exercise class Time1
51 import javax.swing.JOptionPane;
52
53 public class TimeTest {
54    public static void main( String args[] )
55    {
56       Time1 t = new Time1();  // calls Time1 constructor
57       String output;
58
59       output = "The initial universal time is: " +
60                t.toUniversalString() +
61                "\nThe initial standard time is: " +
62                t.toString() +
63                "\nImplicit toString() call: " + t;
64
65       t.setTime( 13, 27, 6 );
66       output += "\n\nUniversal time after setTime is: " +
67                 t.toUniversalString() +
68                 "\nStandard time after setTime is: " +
69                 t.toString();
70
71       t.setTime( 99, 99, 99 );  // all invalid values
72       output += "\n\nAfter attempting invalid settings: " +
73                 "\nUniversal time: " + t.toUniversalString() +
74                 "\nStandard time: " + t.toString();
75
```



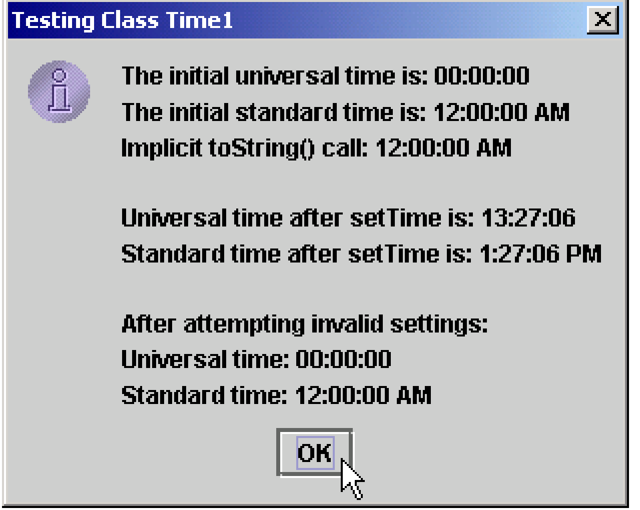
Outline

TimeTest.java (Part 1 of 2)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
76        JOptionPane.showMessageDialog( null, output,
77           "Testing Class Time1",
78           JOptionPane.INFORMATION_MESSAGE );
79
80        System.exit( 0 );
81     } // end main
82  } // end class TimeTest
```


TimeTest.java (Part 2 of 2)



Program Output

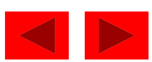
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

26.3 Class Scope

- Class scope
  - Instance variables and methods
  - Class members accessible to methods
    - Can be referenced by name
  - Outside scope, cannot be referenced by name
  - Visible (`public`) members accessed through a handle

    `objectReferenceName.VariableName`
- Block scope
  - Variables defined in a method known only to that method
  - If variable has same name as class variable, class variable hidden
  - Can be accessed using keyword `this` (discussed later)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



26.4 Creating Packages

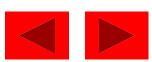
- Packages
  - Directory structures that organize classes and interfaces
  - Mechanism for software reuse
- Creating packages
  - Create a `public` class
    - If not `public`, can only be used by classes in same package
  - Choose a package name and add a `package` statement to source code file
  - Compile class (placed into appropriate directory)
  - Import into other programs

    Naming: Internet domain name in reverse order
    - After name reversed, choose your own structure

```
package com.deitel.chtp3.ch26;
```

  - See text for detailed instructions

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

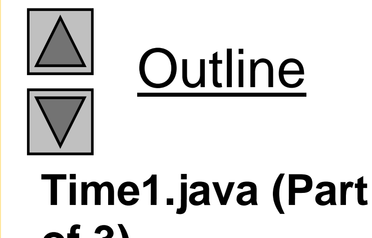


Time1.java (Part 1 of 3)

```
1  // Fig. 26.2: Time1.java
2  // Time1 class definition
3  package com.deitel.chtp4.ch26;
4  import java.text.DecimalFormat;  // used for number formatting
5
6  // This class maintains the time in 24-hour format
7  public class Time1 extends Object {
8     private int hour;     // 0 - 23
9     private int minute;   // 0 - 59
10    private int second;   // 0 - 59
11
12    // Time1 constructor initializes each instance variable
13    // to zero. Ensures that each Time1 object starts in a
14    // consistent state.
15    public Time1()
16    {
17       setTime( 0, 0, 0 );
18    } // end Time1 constructor
19
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

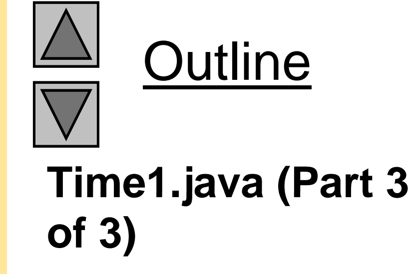
```
20    // Set a new time value using military time. Perform
21    // validity checks on the data. Set invalid values
22    // to zero.
23    public void setTime( int h, int m, int s )
24    {
25       hour = ( ( h >= 0 && h < 24 ) ? h : 0 );
26       minute = ( ( m >= 0 && m < 60 ) ? m : 0 );
27       second = ( ( s >= 0 && s < 60 ) ? s : 0 );
28    } // end method setTime
29
30    // Convert to String in universal-time format
31    public String toUniversalString()
32    {
33       DecimalFormat twoDigits = new DecimalFormat( "00" );
34
35       return twoDigits.format( hour ) + ":" +
36              twoDigits.format( minute ) + ":" +
37              twoDigits.format( second );
38    } // end method toUniversalString
39
```



Time1.java (Part 2 of 3)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
40    // Convert to String in standard-time format
41    public String toString()
42    {
43       DecimalFormat twoDigits = new DecimalFormat( "00" );
44
45       return ( (hour == 12 || hour == 0) ? 12 : hour % 12 ) +
46              ":" + twoDigits.format( minute ) +
47              ":" + twoDigits.format( second ) +
48              ( hour < 12 ? " AM" : " PM" );
49    } // end method toString
50 } // end class Time1
```



Time1.java (Part 3 of 3)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

TimeTest.java

```
51 // Fig. 26.2: TimeTest.java
52 // Class TimeTest to use imported class Time1
53 import javax.swing.JOptionPane;
54 import com.deitel.chtp4.ch26.Time1;  // import Time1 class
55
56 public class TimeTest {
57    public static void main( String args[] )
58    {
59       Time1 t = new Time1();
60
61       t.setTime( 13, 27, 06 );
62       String output =
63          "Universal time is: " + t.toUniversalString() +
64          "\nStandard time is: " + t.toString();
65
66       JOptionPane.showMessageDialog( null, output,
67          "Packaging Class Time1 for Reuse",
68          JOptionPane.INFORMATION_MESSAGE );
69
70       System.exit( 0 );
71    } // end main
72 } // end class TimeTest
```

Program Output

Packaging Class Time1 for Reuse

Universal time is: 13:27:06
Standard time is: 1:27:06 PM

OK

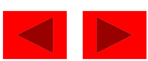
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

26.5 Initializing Class Objects: Constructors

- Constructor
  - Can initialize members of an object
  - Cannot have return type
  - Class may have overloaded constructors
  - Initializers passed as arguments to constructor
  - Definition/initialization of new objects takes form:

    `ref = new ClassName( arguments );`
    - Constructor has same name as class
  - If no constructor defined, compiler makes default constructor
    - Defaults: `0` for primitive numeric types, `false` for `boolean`, `null` for references
    - If constructor defined, no default constructor
  - Can have constructor with no arguments

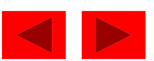
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



26.6 Using Set and Get Methods

- Set methods
  - `public` method that sets `private` variables
  - Does not violate notion of `private` data
    - Change only the variables you want
  - Called mutator methods (change value)
- Get methods
  - `public` method that displays `private` variables
  - Again, does not violate notion of `private` data
    - Only display information you want to display
  - Called accessor or query methods

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

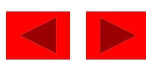


26.6 Using Set and Get Methods (II)

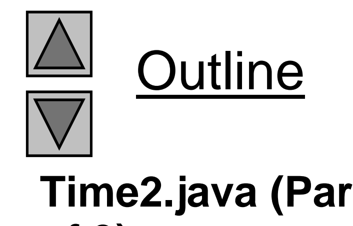
- Every event has a source
  - GUI component with which user interacted
  - `ActionEvent` parameter can check its source
    - Method `getSource`

```
public void actionPerformed( ActionEvent e )
   if ( e.getSource() == myButton )
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



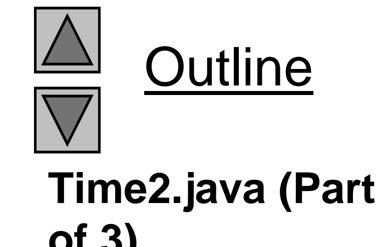
```
1  // Fig. 26.3: Time2.java
2  // Time2 class definition
3  package com.deitel.chtp4.ch26;    // place Time2 in a package
4  import java.text.DecimalFormat;   // used for number formatting
5
6  // This class maintains the time in 24-hour format
7  public class Time2 extends Object {
8     private int hour;      // 0 - 23
9     private int minute;    // 0 - 59
10    private int second;    // 0 - 59
11
12    // Time2 constructor initializes each instance variable
13    // to zero. Ensures that Time object starts in a
14    // consistent state.
15    public Time2() { setTime( 0, 0, 0 ); }
16
17    // Set Methods
18    // Set a new time value using universal time. Perform
19    // validity checks on the data. Set invalid values to zero.
20    public void setTime( int h, int m, int s )
21    {
22       setHour( h );    // set the hour
23       setMinute( m );  // set the minute
24       setSecond( s );  // set the second
25    } // end method setTime
26
```



Time2.java (Part 1 of 3)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

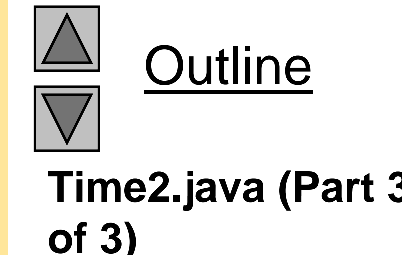
```
27    // set the hour
28    public void setHour( int h )
29       { hour = ( ( h >= 0 && h < 24 ) ? h : 0 ); }
30
31    // set the minute
32    public void setMinute( int m )
33       { minute = ( ( m >= 0 && m < 60 ) ? m : 0 ); }
34
35    // set the second
36    public void setSecond( int s )
37       { second = ( ( s >= 0 && s < 60 ) ? s : 0 ); }
38
39    // Get Methods
40    // get the hour
41    public int getHour() { return hour; }
42
43    // get the minute
44    public int getMinute() { return minute; }
45
46    // get the second
47    public int getSecond() { return second; }
48
```



Time2.java (Part 2 of 3)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

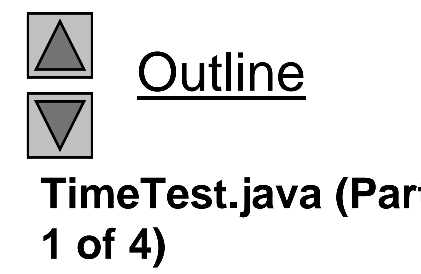
```
49    // Convert to String in universal-time format
50    public String toUniversalString()
51    {
52       DecimalFormat twoDigits = new DecimalFormat( "00" );
53
54       return twoDigits.format( getHour() ) + ":" +
55              twoDigits.format( getMinute() ) + ":" +
56              twoDigits.format( getSecond() );
57    } // end method toUniversalString
58
59    // Convert to String in standard-time format
60    public String toString()
61    {
62       DecimalFormat twoDigits = new DecimalFormat( "00" );
63
64       return ( ( getHour() == 12 || getHour() == 0 ) ?
65                 12 : getHour() % 12 ) + ":" +
66              twoDigits.format( getMinute() ) + ":" +
67              twoDigits.format( getSecond() ) +
68              ( getHour() < 12 ? " AM" : " PM" );
69    } // end method toString
70 } // end class Time2
```



Time2.java (Part 3 of 3)

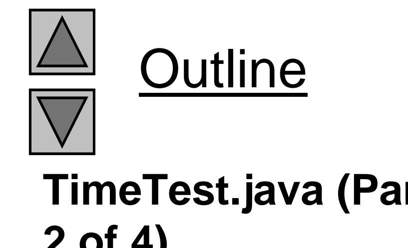
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
71 // Fig. 26.3: TimeTest.java
72 // Demonstrating the Time2 class set and get methods
73 import java.awt.*;
74 import java.awt.event.*;
75 import javax.swing.*;
76 import com.deitel.chtp4.ch26.Time2;
77
78 public class TimeTest extends JApplet
79                       implements ActionListener {
80    private Time2 t;
81    private JLabel hourLabel, minuteLabel, secondLabel;
82    private JTextField hourField, minuteField,
83                       secondField, display;
84    private JButton tickButton;
85
86    public void init()
87    {
88       t = new Time2();
89
90       Container c = getContentPane();
91
```



TimeTest.java (Part 1 of 4)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

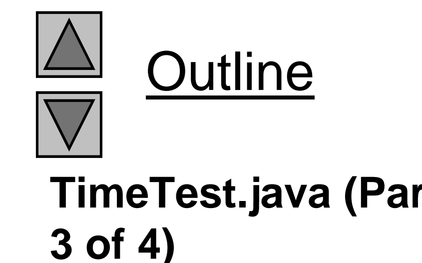


TimeTest.java (Part 2 of 4)

```
92      c.setLayout( new FlowLayout() );
93      hourLabel = new JLabel( "Set Hour" );
94      hourField = new JTextField( 10 );
95      hourField.addActionListener( this );
96      c.add( hourLabel );
97      c.add( hourField );
98
99      minuteLabel = new JLabel( "Set minute" );
100      minuteField = new JTextField( 10 );
101      minuteField.addActionListener( this );
102      c.add( minuteLabel );
103      c.add( minuteField );
104
105      secondLabel = new JLabel( "Set Second" );
106      secondField = new JTextField( 10 );
107      secondField.addActionListener( this );
108      c.add( secondLabel );
109      c.add( secondField );
110
111      display = new JTextField( 30 );
112      display.setEditable( false );
113      c.add( display );
114
```

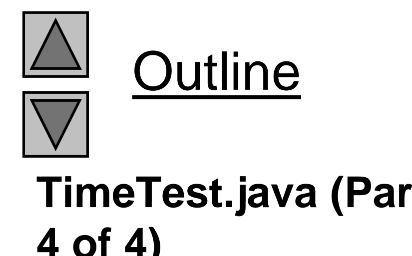
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
115       tickButton = new JButton( "Add 1 to Second" );
116       tickButton.addActionListener( this );
117       c.add( tickButton );
118
119       updateDisplay();
120    } // end method init
121
122    public void actionPerformed( ActionEvent e )
123    {
124       if ( e.getSource() == tickButton )
125          tick();
126       else if ( e.getSource() == hourField ) {
127          t.setHour(
128             Integer.parseInt( e.getActionCommand() ) );
129          hourField.setText( "" );
130       }
131       else if ( e.getSource() == minuteField ) {
132          t.setMinute(
133             Integer.parseInt( e.getActionCommand() ) );
134          minuteField.setText( "" );
135       }
136       else if ( e.getSource() == secondField ) {
137          t.setSecond(
138             Integer.parseInt( e.getActionCommand() ) );
139          secondField.setText( "" );
140       }
```



TimeTest.java (Part 3 of 4)

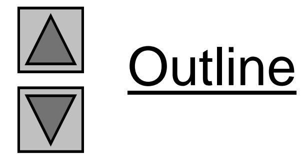
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



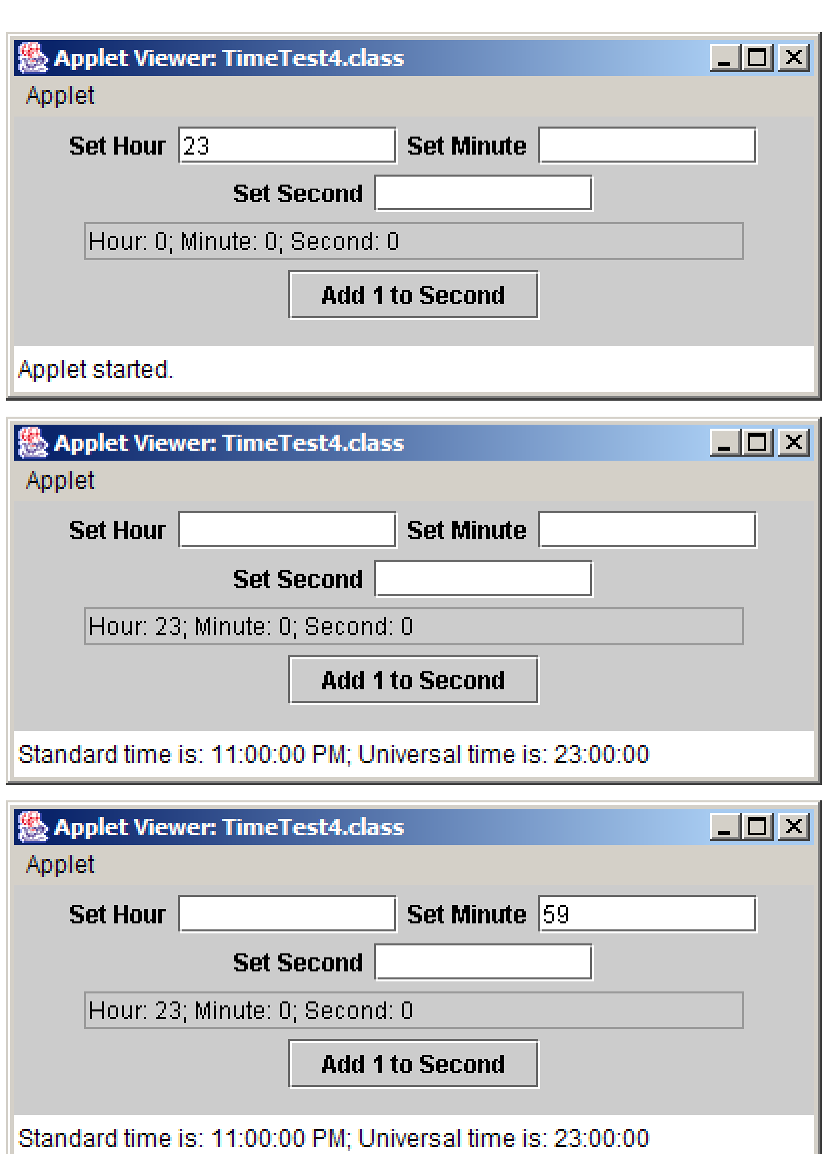
TimeTest.java (Part 4 of 4)

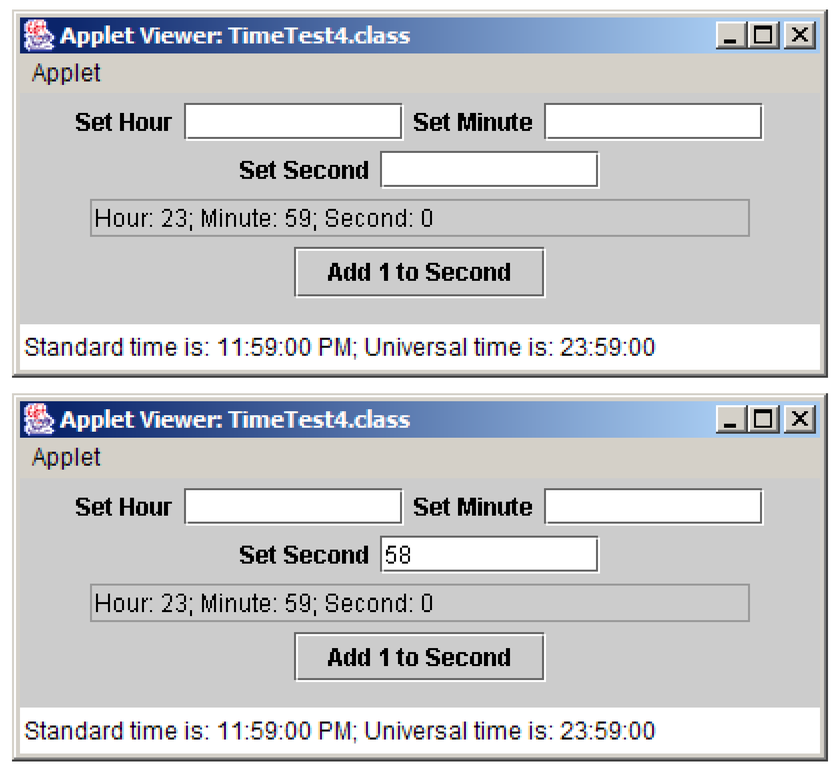
```
141
142         updateDisplay();
143      } // end method actionPerformed
144
145      public void updateDisplay()
146      {
147         display.setText( "Hour: " + t.getHour() +
148            "; Minute: " + t.getMinute() +
149            "; Second: " + t.getSecond() );
150         showStatus( "Standard time is: " + t.toString() +
151            "; Universal time is: " + t.toUniversalString() );
152      } // end method updateDisplay
153
154      public void tick()
155      {
156         t.setSecond( ( t.getSecond() + 1 ) % 60 );
157
158         if ( t.getSecond() == 0 ) {
159            t.setMinute( ( t.getMinute() + 1 ) % 60 );
160
161            if ( t.getMinute() == 0 )
162               t.setHour( ( t.getHour() + 1 ) % 24 );
163         } // end if
164      } // end method tick
165   } // end class TimeTest
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

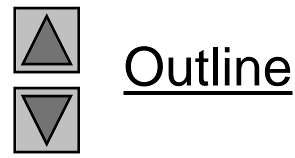


Program Output

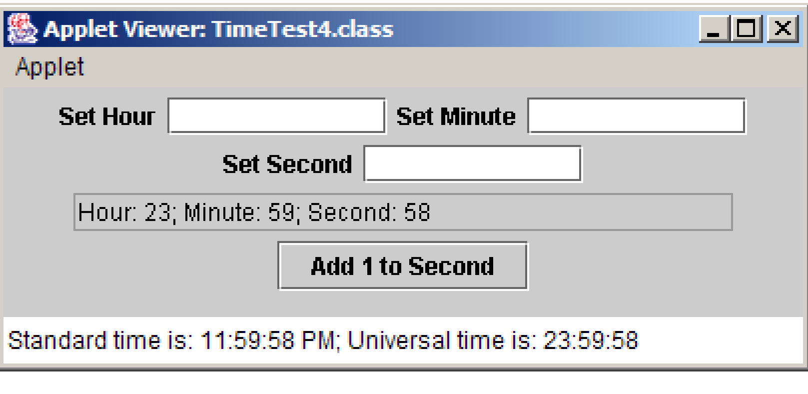


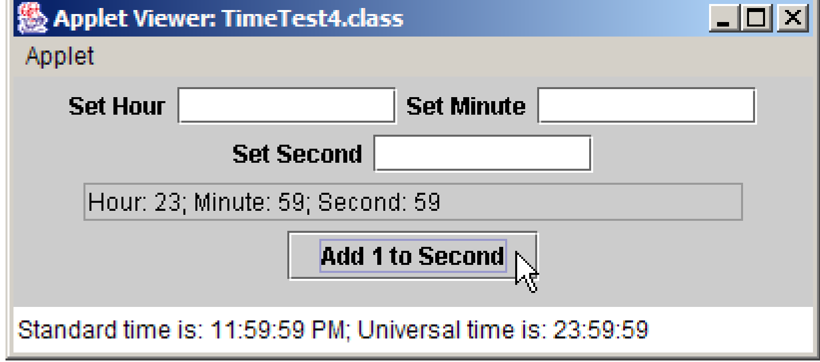


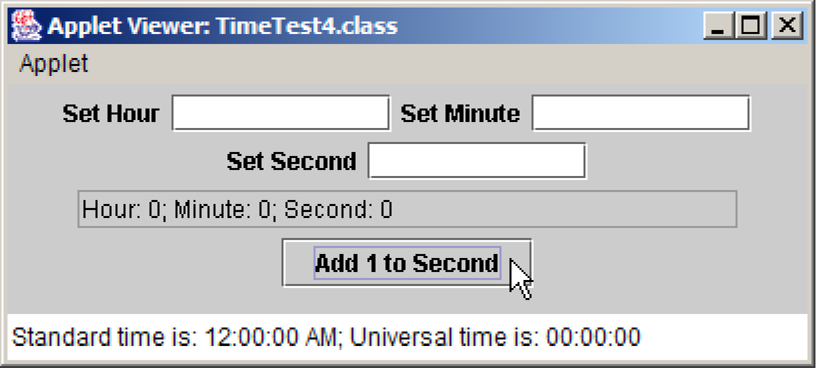
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Program Output





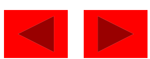


© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

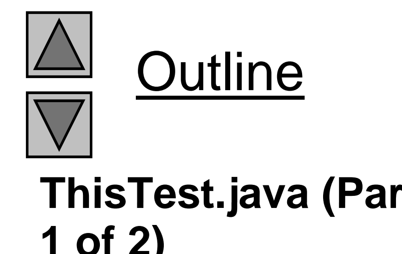
26.7 Using the `this` Reference

- Each object has a reference to itself
  - The `this` reference
    - Implicitly used to refer to instance variables and methods
- Inside methods
  - If parameter has same name as instance variable
    - Instance variable hidden
  - Use `this.variableName` to explicitly refer to the instance variable
  - Use `variableName` to refer to the parameter

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



```
1  // Fig. 26.4: ThisTest.java
2  // Using the this reference to refer to
3  // instance variables and methods.
4  import javax.swing.*;
5  import java.text.DecimalFormat;
6
7  public class ThisTest {
8     public static void main( String args[] )
9     {
10       SimpleTime t = new SimpleTime( 12, 30, 19 );
11
12       JOptionPane.showMessageDialog( null, t.buildString(),
13          "Demonstrating the \"this\" Reference",
14          JOptionPane.INFORMATION_MESSAGE );
15
16       System.exit( 0 );
17    } // end method main
18 } // end class ThisTest
19
20 class SimpleTime {
21    private int hour, minute, second;
22
```



ThisTest.java (Part 1 of 2)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
23    public SimpleTime( int hour, int minute, int second )
24    {
25       this.hour = hour;
26       this.minute = minute;
27       this.second = second;
28    } // end SimpleTime constructor
29
30    public String buildString()
31    {
32       return "this.toString(): " + this.toString() +
33              "\ntoString(): " + toString() +
34              "\nthis (with implicit toString() call): " +
35              this;
36    } // end method buildString
37
38    public String toString()
39    {
40       DecimalFormat twoDigits = new DecimalFormat( "00" );
41
42       return twoDigits.format( this.hour ) + ":" +
43              twoDigits.format( this.minute ) + ":" +
44              twoDigits.format( this.second );
45    } // end method toString
46 } // end class SimpleTime
```

Outline

ThisTest.java (Part 2 of 2)

Program Output

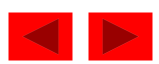
```
Demonstrating the "this" Reference
this.toString(): 12:30:19
toString(): 12:30:19
this (with implicit toString() call): 12:30:19
OK
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

26.8 Finalizers

- Memory
  - Constructors use memory when creating new objects
  - Automatic garbage collection
    - When object no longer used, object marked for garbage collection
    - Garbage collector executes, memory can be reclaimed
    - Memory leaks less common in Java than in C and C++
- `finalizer` method
  - In every class, returns resources to system
    - Performs termination housekeeping on object
  - Name always `finalize`
    - Takes no parameters, returns no value
  - Defined in class `Object` as a placeholder
    - Every class gets a `finalize` method

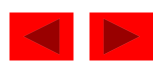
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



26.9 Static Class Members

- Static variables
  - Usually, each object gets its own copy of each variable
  - `static` class variables shared among all objects of the class
    - One copy exists for entire class to use
  - Keyword `static`
  - Only have class scope (not global)
  - `static` class variables exist even when no objects do
  - `public static` members accessed through references or class name and dot operator
  - `private static` members accessed through methods
    - If no objects exist, classname and `public static` method must be used

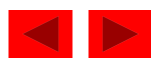
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

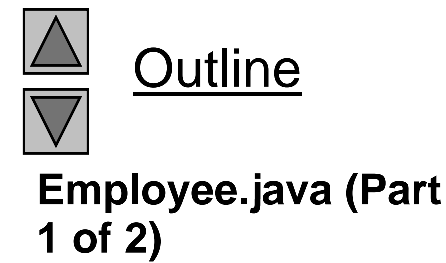


26.9 Static Class Members (II)

- `static` methods
  - Can only access class `static` members
  - `static` methods have no `this` reference
    - `static` variables are independent of objects
- Method `gc`
  - `public static` method of class `System`
  - Suggests garbage collector execute immediately
    - Can be ignored
    - Garbage collector not guaranteed to collect objects in a specific order

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

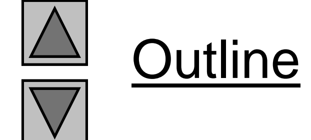




Employee.java (Part 1 of 2)

```
1  // Fig. 26.5: Employee.java
2  // Declaration of the Employee class.
3  public class Employee extends Object {
4     private String firstName;
5     private String lastName;
6     private static int count;  // # of objects in memory
7
8     public Employee( String fName, String lName )
9     {
10       firstName = fName;
11       lastName = lName;
12
13       ++count;  // increment static count of employees
14       System.out.println( "Employee object constructor: " +
15                           firstName + " " + lastName );
16    } // end Employee constructor
17
18    protected void finalize()
19    {
20       --count;  // decrement static count of employees
21       System.out.println( "Employee object finalizer: " +
22                           firstName + " " + lastName +
23                           "; count = " + count );
24    } // end method finalize
25
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Employee.java (Part 2 of 2)

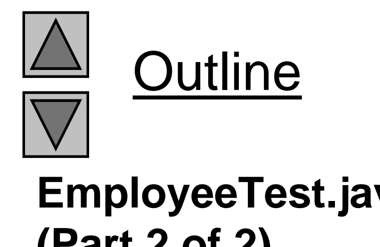
EmployeeTest.java (Part 1 of 2)

```
26    public String getFirstName() { return firstName; }
27
28    public String getLastName() { return lastName; }
29
30    public static int getCount() { return count; }
31 } // end class Employee
```

```
32 // Fig. 8.12: EmployeeTest.java
33 // Test Employee class with static class variable,
34 // static class method, and dynamic memory.
35 import javax.swing.*;
36
37 public class EmployeeTest {
38    public static void main( String args[] )
39    {
40       String output;
41
42       output = "Employees before instantiation: " +
43                Employee.getCount();
44
45       Employee e1 = new Employee( "Susan", "Baker" );
46       Employee e2 = new Employee( "Bob", "Jones" );
47
```

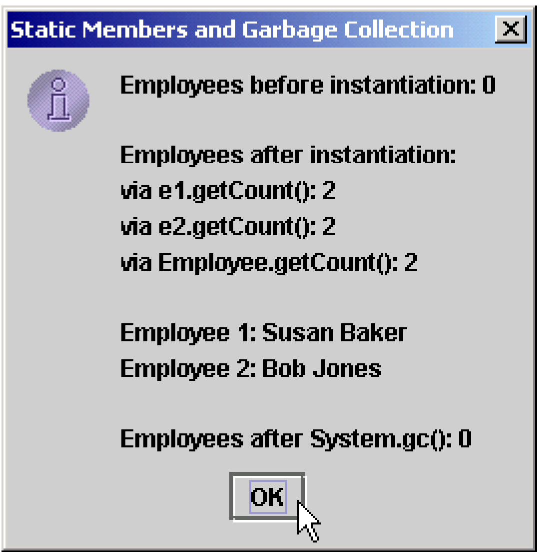
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

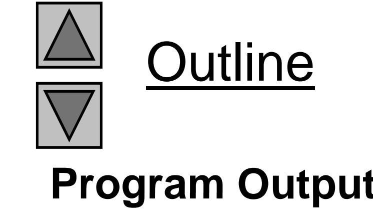
```
48       output += "\n\nEmployees after instantiation: " +
49                 "\nvia e1.getCount(): " + e1.getCount() +
50                 "\nvia e2.getCount(): " + e2.getCount() +
51                 "\nvia Employee.getCount(): " +
52                 Employee.getCount();
53
54       output += "\n\nEmployee 1: " + e1.getFirstName() +
55                 " " + e1.getLastName() +
56                 "\nEmployee 2: " + e2.getFirstName() +
57                 " " + e2.getLastName();
58
59       // mark objects referred to by e1 and e2
60       // for garbage collection
61       e1 = null;
62       e2 = null;
63
64       System.gc(); // suggest that garbage collector be called
65
66       output += "\n\nEmployees after System.gc(): " +
67                 Employee.getCount();
68
69       JOptionPane.showMessageDialog( null, output,
70          "Static Members and Garbage Collection",
71          JOptionPane.INFORMATION_MESSAGE );
72       System.exit( 0 );
73    } // end main
74 } // end class EmployeeTest
```



EmployeeTest.java (Part 2 of 2)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.





Program Output

```
Employee object constructor: Susan Baker
Employee object constructor: Bob Jones
Employee object finalizer: Susan Baker; count = 1
Employee object finalizer: Bob Jones; count = 0
```

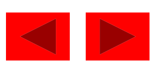
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Chapter 27 - Java Object-Oriented Programming

Outline

27.1 Introduction
27.2 Superclasses and Subclasses
27.3 `protected` Members
27.4 Relationship between Superclass Objects and Subclass Objects
27.5 Implicit Subclass-Object-to-Superclass-Object Conversion
27.6 Software Engineering with Inheritance
27.7 Composition vs. Inheritance
27.8 Introduction to Polymorphism
27.9 Type Fields and `switch` Statements
27.10 Dynamic Method Binding
27.11 `final` Methods and Classes
27.12 Abstract Superclasses and Concrete Classes
27.13 Polymorphism Example
27.14 New Classes and Dynamic Binding
27.15 Case Study: Inheriting Interface and Implementation
27.16 Case Study: Creating and Using Interfaces
27.17 Inner Class Definitions
27.18 Notes on Inner Class Definitions
27.19 Type-Wrapper Classes for Primitive Classes

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Objectives

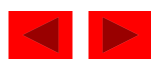
- In this chapter, you will learn:
  - To understand inheritance and software reusability.
  - To understand superclasses and subclasses.
  - To appreciate how polymorphism makes systems extensible and maintainable.
  - To understand the distinction between `abstract` classes and concrete classes.
  - To learn how to create `abstract` classes and interfaces.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

27.1 Introduction

- Object-Oriented Programming (OOP)
  - Inheritance - form of software reusability
    - New classes created from existing ones
      - Absorb attributes and behaviors, and add in their own
    - Subclass inherits from superclass
      - Direct superclass - subclass explicitly inherits
      - Indirect superclass - subclass inherits from two or more levels up the class hierarchy
  - Polymorphism
    - Write programs in a general fashion to handle a wide variety of classes

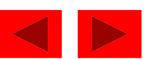
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



27.1 Introduction

- Object-Oriented Programming
  - Introduce `protected` member access
    - Subclass methods and methods of other classes in the same package can access `protected` superclass members.
  - Abstraction - Seeing the big picture
  - Relationships
    - "is a" - inheritance
      - Object of subclass "is an" object of the superclass
    - "has a" - composition
      - Object "has an" object of another class as a member

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



27.1 Introduction

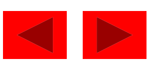
- Object-Oriented Programming
  - A subclass cannot directly access `private` members of its superclass.
  - Class libraries
    - Someday software may be constructed from standardized, reusable components (like hardware)
    - Create more powerful software

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

27.2 Superclasses and Subclasses

- Inheritance example
  - A rectangle "is a" quadrilateral
    - Rectangle is a specific type of quadrilateral
    - Quadrilateral is the superclass, rectangle is the subclass
    - Incorrect to say quadrilateral "is a" rectangle
  - Naming can be confusing because subclass has more features than superclass
    - Subclass more specific than superclass
    - Every subclass "is an" object of its superclass, but not vice-versa

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



27.2 Superclasses and Subclasses

Superclass	Subclasses
`Student`	`GraduateStudent` `UndergraduateStudent`

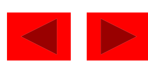
`Shape`	`Circle`
	`Triangle`
	`Rectangle`

`Loan`	`CarLoan`
	`HomeImprovementLoan`
	`MortgageLoan`

`Employee`	`FacultyMember` `StaffMember`
`Account`	`CheckingAccount` `SavingsAccount`

Fig. 27.1 Some simple inheritance examples.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



27.2 Superclasses and Subclasses

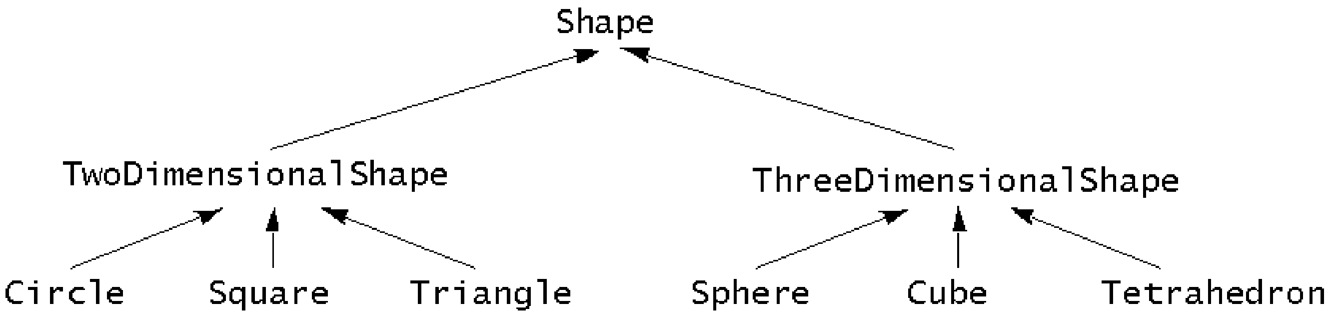


Fig. 27.2 A portion of a Shape class hierarchy.

- Form tree-like hierarchal structures
  - Create a hierarchy for class Shape

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

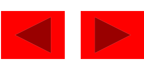
27.2 Superclasses and Subclasses

- Using inheritance
  - Use keyword `extends`

```
class TwoDimensionalShape extends Shape{ ... }
```

  - `private` members of superclass not directly accessible to subclass
  - All other variables keep their member access

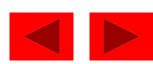
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



27.3 `protected` Members

- In a superclass
  - `public` members
    - Accessible anywhere program has a reference to a superclass or subclass type
  - `private` members
    - Accessible only in methods of the superclass
  - `protected` members
    - Intermediate protection between `private` and `public`
    - Only accessible by methods of superclass, of subclass, or classes in the same package
- Subclass methods
  - Can refer to `public` or `protected` members by name
  - Overridden methods accessible with `super.methodName`

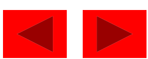
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

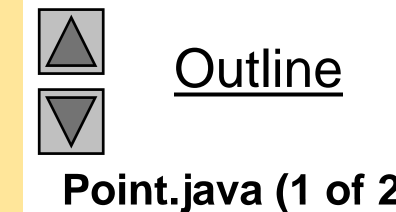


27.4 Relationship between Superclass Objects and Subclass Objects

- Object of subclass
  - Can be treated as object of superclass
    - Reverse not true
  - Suppose many classes inherit from one superclass
    - Can make an array of superclass references
    - Treat all objects like superclass objects
  - Explicit cast
    - Convert superclass reference to a subclass reference (downcasting)
    - Can only be done when superclass reference actually referring to a subclass object

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.





Point.java (1 of 2)

```
1  // Fig. 27.3: Point.java
2  // Definition of class Point
3
4  public class Point {
5     protected int x, y; // coordinates of the Point
6
7     // No-argument constructor
8     public Point()
9     {
10       // implicit call to superclass constructor occurs here
11       setPoint( 0, 0 );
12    } // end Point constructor
13
14    // Constructor
15    public Point( int a, int b )
16    {
17       // implicit call to superclass constructor occurs here
18       setPoint( a, b );
19    } // end Point constructor
20
21    // Set x and y coordinates of Point
22    public void setPoint( int a, int b )
23    {
24       x = a;
25       y = b;
26    } // end method setPoint
27
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

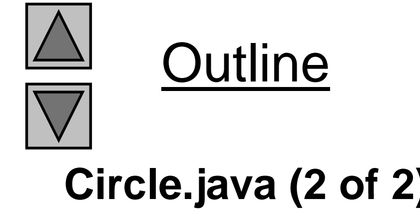
```
28    // get x coordinate
29    public int getX() { return x; }
30
31    // get y coordinate
32    public int getY() { return y; }
33
34    // convert the point into a String representation
35    public String toString()
36       { return "[" + x + ", " + y + "]"; }
37 } // end class Point
```

```
38 // Fig. 27.3: Circle.java
39 // Definition of class Circle
40
41 public class Circle extends Point {  // inherits from Point
42    protected double radius;
43
44    // No-argument constructor
45    public Circle()
46    {
47       // implicit call to superclass constructor occurs here
48       setRadius( 0 );
49    } // end Circle constructor
50
```


Point.java (2 of 2)

Circle.java (1 of 2)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Circle.java (2 of 2)

```
51    // Constructor
52    public Circle( double r, int a, int b )
53    {
54       super( a, b );  // call to superclass constructor
55       setRadius( r );
56    } // end Circle constructor
57
58    // Set radius of Circle
59    public void setRadius( double r )
60       { radius = ( r >= 0.0 ? r : 0.0 ); }
61
62    // Get radius of Circle
63    public double getRadius() { return radius; }
64
65    // Calculate area of Circle
66    public double area() { return Math.PI * radius * radius; }
67
68    // convert the Circle to a String
69    public String toString()
70    {
71       return "Center = " + "[" + x + ", " + y + "]" +
72              "; Radius = " + radius;
73    } // end method toString
74 } // end class Circle
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

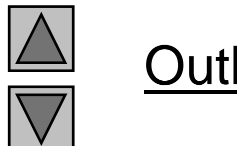
```
75 // Fig. 27.3: InheritanceTest.java
76 // Demonstrating the "is a" relationship
77 import java.text.DecimalFormat;
78 import javax.swing.JOptionPane;
79
80 public class InheritanceTest {
81    public static void main( String args[] )
82    {
83       Point pointRef, p;
84       Circle circleRef, c;
85       String output;
86
87       p = new Point( 30, 50 );
88       c = new Circle( 2.7, 120, 89 );
89
90       output = "Point p: " + p.toString() +
91                "\nCircle c: " + c.toString();
92
93       // use the "is a" relationship to refer to a Circle
94       // with a Point reference
95       pointRef = c;   // assign Circle to pointRef
96
97       output += "\n\nCircle c (via pointRef): " +
98                 pointRef.toString();
99
```

Outline

InheritanceTest.java (1 of 2)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
100      // Use downcasting (casting a superclass reference to a
101      // subclass data type) to assign pointRef to circleRef
102      circleRef = (Circle) pointRef;
103
104      output += "\n\nCircle c (via circleRef): " +
105               circleRef.toString();
106
107      DecimalFormat precision2 = new DecimalFormat( "0.00" );
108      output += "\nArea of c (via circleRef): " +
109               precision2.format( circleRef.area() );
110
111      // Attempt to refer to Point object
112      // with Circle reference
113      if ( p instanceof Circle ) {
114         circleRef = (Circle) p;  // line 40 in Test.java
115         output += "\n\ncast successful";
116      }
117      else
118         output += "\n\np does not refer to a Circle";
119
120      JOptionPane.showMessageDialog( null, output,
121         "Demonstrating the \"is a\" relationship",
122         JOptionPane.INFORMATION_MESSAGE );
123
124      System.exit( 0 );
125   } // end main
126 } // end class InheritanceTest
```



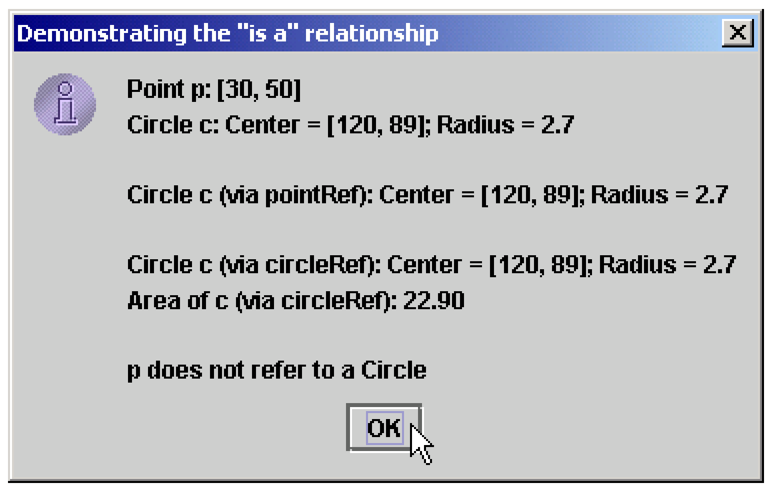
Outline

InheritanceTest.java (2 of 2)

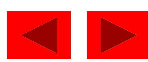
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

27.4 Relationship between Superclass Objects and Subclass Objects

Figure 27.3 Assigning subclass references to superclass references - `InheritanceTest.java`



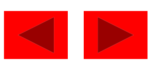
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



27.4 Relationship between Superclass Objects and Subclass Objects

- Extending a class
  - To invoke superclass constructor explicitly (called implicitly by default)
    - `super(); //can pass arguments if needed`
    - If called explicitly, must be first statement
- Overriding methods
  - Subclass can redefine superclass method
    - When method mentioned in subclass, subclass version called
    - Access original superclass method with `super.methodName`

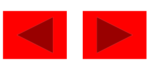
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



27.4 Relationship between Superclass Objects and Subclass Objects

- Every Applet has used these techniques
  - Java implicitly uses class `Object` as superclass for all classes
  - We have overridden `init` and `paint` when we extended `JApplet`
- `instanceof` operator
  - `if (p instanceof Circle)`
  - Returns `true` if the object to which `p` points "is a" `Circle`

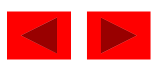
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



27.5 Implicit Subclass-Object-to-Superclass-Object Conversion

- References to subclass objects
  - May be implicitly converted to superclass references
    - Makes sense - subclass contains members corresponding to those of superclass
  - Referring to a subclass object with a superclass reference
    - Allowed - a subclass object "is a" superclass object
    - Can only refer to superclass members
  - Referring to a superclass object with a subclass reference
    - Error
    - Must first be cast to a superclass reference
  - Need way to use superclass references but call subclass methods
    - Discussed later in the chapter

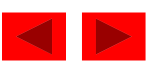
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



27.6 Software Engineering with Inheritance

- Inheritance
  - Customize existing software
    - Create a new class, add attributes and behaviors as needed
- Software reuse key to large-scale projects
  - Java and OOP does this
  - Availability of class libraries and inheritance
- Superclass
  - Specifies commonality
  - Look for commonality among a set of classes
    - "Factor it out" to form the superclass
  - Subclasses are then customized

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

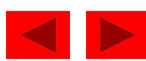


27.7 Composition vs. Inheritance

- "is a" relationship
  - Inheritance
- "has a" relationship
  - Composition, having other objects as members
- Example

```
Employee "is a" BirthDate;            //Wrong!
Employee "has a" BirthDate;           //Composition
```

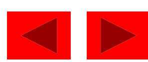
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



27.8 Introduction to Polymorphism

- With polymorphism
  - Design and implement extensible programs
  - Generically process superclass objects
  - Easy to add classes to hierarchy
    - Little or no modification required
    - Only parts of program that need direct knowledge of new class must be changed

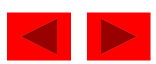
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



27.9 Type Fields and `switch` Statements

- `switch` statements
  - Can be used to deal with many objects of different types
    - Appropriate action based on type
- Problems
  - Programmer may forget to include a type
  - Might forget to test all possible cases
  - Every addition/deletion of a class requires all `switch` statements to be changed
    - Tracking all these changes is time consuming and error prone
  - Polymorphic programming can eliminate the need for `switch` logic
    - Avoids all these problems

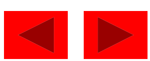
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



27.10 Dynamic Method Binding

- Dynamic Method Binding
  - At execution time, method calls routed to appropriate version
- Example
  - `Circle`, `Triangle`, `Rectangle` and `Square` all subclasses of `Shape`
    - Each has an overridden `draw` method
  - Call `draw` using superclass references
    - At execution time, program determines to which class the reference is actually pointing
    - Calls appropriate `draw` method

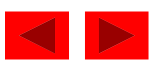
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



27.11 `final` Methods and Classes

- Defining variables `final`
  - Indicates they cannot be modified after definition
  - Must be initialized when defined
- Defining methods `final`
  - Cannot be overridden in a subclass
  - `static` and `private` methods are implicitly `final`
  - Program can inline `final` methods
    - Actually inserts method code at method call locations
    - Improves program performance
- Defining classes `final`
  - Cannot be a superclass (cannot inherit from it)
  - All methods in class are implicitly `final`

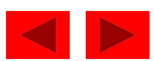
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



27.12 Abstract Superclasses and Concrete Classes

- Abstract classes (abstract superclasses)
  - Contain one or more `abstract` methods
  - Cannot be instantiated
    - Causes syntax error
    - Can still have instance data and non-`abstract` methods
    - Can still define constructor
  - Sole purpose is to be a superclass
    - Other classes inherit from it
  - Too generic to define real objects
  - Declare class with keyword `abstract`

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



27.12 Abstract Superclasses and Concrete Classes

- Concrete class
  - Can instantiate objects
  - Provide specifics
- Class hierarchies
  - Most general classes are usually `abstract`
    - `TwoDimensionalShape` - too generic to be concrete

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

27.13 Polymorphism Example

- Class `Quadrilateral`
  - `Rectangle` "is a" `Quadrilateral`
  - `getPerimeter` method can be performed on any subclass
    - `Square`, `Parallelogram`, `Trapezoid`
    - Same method takes on "many forms" - polymorphism
  - Can method is called with superclass reference
    - Java chooses correct overriden method
- References
  - Can create references to `abstract` classes

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

27.13 Polymorphism Example

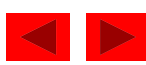
- Iterator classes
  - Walks through all the objects in a container (such as an array)
  - Used in polymorphic programming
    - Walk through an array of superclass references
    - Call `draw` method for each reference

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

27.14 New Classes and Dynamic Binding

- Dynamic binding (late binding)
  - Accommodates new classes
  - Object's type does not need to be known at compile time
  - At execution time, method call matched with object

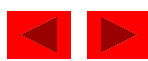
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



27.15 Case Study: Inheriting Interface and Implementation

- Polymorphism example
  - abstract superclass Shape
    - Subclasses Point, Circle, Cylinder
    - abstract method
      - getName
    - non-abstract methods
      - area (return 0.0)
      - volume (return 0.0)
  - Class Shape used to define a set of common methods
    - Interface is the three common methods
    - Implementation of area and volume used for first levels of hierarchy
  - Create an array of Shape references
    - Point them to various subclass objects
    - Call methods through the Shape reference

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



```
1 // Fig. 27.4: Shape.java
2 // Definition of abstract base class Shape
3
4 public abstract class Shape extends Object {
5    public double area() { return 0.0; }
6    public double volume() { return 0.0; }
7    public abstract String getName();
8 } // end class Shape
```

```
9 // Fig. 27.4: Point.java
10 // Definition of class Point
11
12 public class Point extends Shape {
13    protected int x, y; // coordinates of the Point
14
15    // no-argument constructor
16    public Point() { setPoint( 0, 0 ); }
17
18    // constructor
19    public Point( int a, int b ) { setPoint( a, b ); }
20
21    // Set x and y coordinates of Point
22    public void setPoint( int a, int b )
23    {
24       x = a;
25       y = b;
26    } // end method setPoint
27
```

Outline

Shape.java

Point.java (1 of 2)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Outline

Point.java (2 of 2)

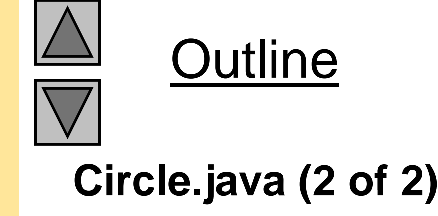
```
28    // get x coordinate
29    public int getX() { return x; }
30
31    // get y coordinate
32    public int getY() { return y; }
33
34    // convert the point into a String representation
35    public String toString()
36       { return "[" + x + ", " + y + "]"; }
37
38    // return the class name
39    public String getName() { return "Point"; }
40 } // end class Point
```

Circle.java (1 of 2)

```
41 // Fig. 27.4: Circle.java
42 // Definition of class Circle
43
44 public class Circle extends Point {  // inherits from Point
45    protected double radius;
46
47    // no-argument constructor
48    public Circle()
49    {
50       // implicit call to superclass constructor here
51       setRadius( 0 );
52    } // end Circle constructor
53
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

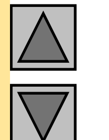
```
54    // Constructor
55    public Circle( double r, int a, int b )
56    {
57       super( a, b );  // call the superclass constructor
58       setRadius( r );
59    } // end Circle constructor
60
61    // Set radius of Circle
62    public void setRadius( double r )
63       { radius = ( r >= 0 ? r : 0 ); }
64
65    // Get radius of Circle
66    public double getRadius() { return radius; }
67
68    // Calculate area of Circle
69    public double area() { return Math.PI * radius * radius; }
70
71    // convert the Circle to a String
72    public String toString()
73       { return "Center = " + super.toString() +
74                "; Radius = " + radius; }
75
76    // return the class name
77    public String getName() { return "Circle"; }
78 } // end class Circle
```



Circle.java (2 of 2)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

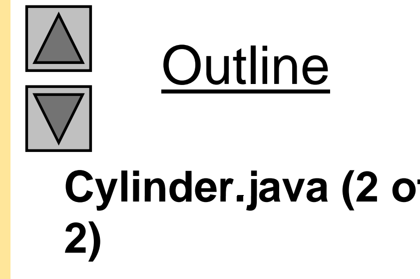
```java
79 // Fig. 27.4: Cylinder.java
80 // Definition of class Cylinder
81
82 public class Cylinder extends Circle {
83    protected double height;  // height of Cylinder
84
85    // no-argument constructor
86    public Cylinder()
87    {
88       // implicit call to superclass constructor here
89       setHeight( 0 );
90    } // end Cylinder constructor
91
92    // constructor
93    public Cylinder( double h, double r, int a, int b )
94    {
95       super( r, a, b );   // call superclass constructor
96       setHeight( h );
97    } // end Cylinder constructor
98
99    // Set height of Cylinder
100    public void setHeight( double h )
101       { height = ( h >= 0 ? h : 0 ); }
102
103    // Get height of Cylinder
104    public double getHeight() { return height; }
105
```



Cylinder.java (1 of 2)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
106    // Calculate area of Cylinder (i.e., surface area)
107    public double area()
108    {
109       return 2 * super.area() +
110              2 * Math.PI * radius * height;
111    } // end method area
112
113    // Calculate volume of Cylinder
114    public double volume() { return super.area() * height; }
115
116    // Convert a Cylinder to a String
117    public String toString()
118       { return super.toString() + "; Height = " + height; }
119
120    // Return the class name
121    public String getName() { return "Cylinder"; }
122 } // end class Cylinder
```



Cylinder.java (2 of 2)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
123 // Fig. 27.4: Test.java
124 // Driver for point, circle, cylinder hierarchy
125 import javax.swing.JOptionPane;
126 import java.text.DecimalFormat;
127
128 public class Test {
129    public static void main( String args[] )
130    {
131       Point point = new Point( 7, 11 );
132       Circle circle = new Circle( 3.5, 22, 8 );
133       Cylinder cylinder = new Cylinder( 10, 3.3, 10, 10 );
134
135       Shape arrayOfShapes[];
136
137       arrayOfShapes = new Shape[ 3 ];
138
139       // aim arrayOfShapes[0] at subclass Point object
140       arrayOfShapes[ 0 ] = point;
141
142       // aim arrayOfShapes[1] at subclass Circle object
143       arrayOfShapes[ 1 ] = circle;
144
145       // aim arrayOfShapes[2] at subclass Cylinder object
146       arrayOfShapes[ 2 ] = cylinder;
147
```


© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
148        String output =
149           point.getName() + ": " + point.toString() + "\n" +
150           circle.getName() + ": " + circle.toString() + "\n" +
151          cylinder.getName() + ": " + cylinder.toString();
152
153       DecimalFormat precision2 = new DecimalFormat( "0.00" );
154
155       // Loop through arrayOfShapes and print the name,
156       // area, and volume of each object.
157       for ( int i = 0; i < arrayOfShapes.length; i++ ) {
158          output += "\n\n" +
159             arrayOfShapes[ i ].getName() + ": " +
160             arrayOfShapes[ i ].toString() +
161             "\nArea = " +
162             precision2.format( arrayOfShapes[ i ].area() ) +
163             "\nVolume = " +
164             precision2.format( arrayOfShapes[ i ].volume() );
165       } // end for
166
167       JOptionPane.showMessageDialog( null, output,
168          "Demonstrating Polymorphism",
169          JOptionPane.INFORMATION_MESSAGE );
170
171       System.exit( 0 );
172    } // end main
173} // end class Test
```

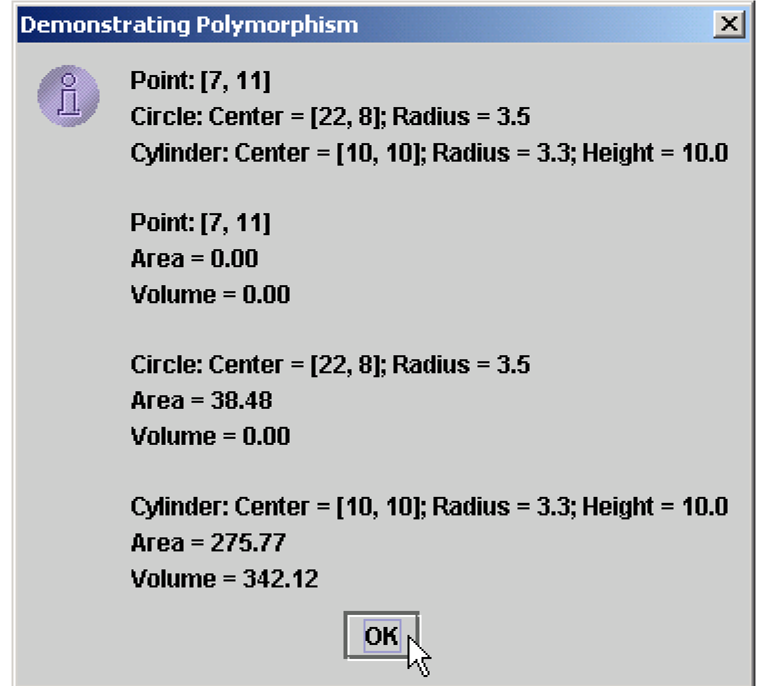
Outline

Test.java (2 of 2)

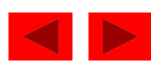
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

27.15 Case Study: Inheriting Interface and Implementation

Figure 27.4 Shape, point, circle, cylinder hierarchy - `Test.java`



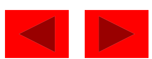
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



27.16 Case Study: Creating and Using Interfaces

- Creating an interface
  - Keyword `interface`
  - Has set of `public abstract` methods
  - Can contain `public final static` data
- Using interfaces
  - Class specifies it uses interface with keyword `implements`
  - Class must define all `abstract` methods in interface
    - Must use same number of arguments, same return type
  - Using interface like signing a contract
    - "I will define all methods specified in the interface"

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



27.16 Case Study: Creating and Using Interfaces

- Using interfaces (continued)
  - Interfaces used in place of `abstract` classes
    - Used when no default implementation
  - Typically `public` data types
    - Interface defined in its own `.java` file
    - Interface name same as file name
  - Same "is a" relationship as inheritance
- Reexamine previous hierarchy
  - Replace `abstract` class `Shape` with interface `Shape`

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Outline

Shape.java

```
1  // Fig. 27.5: Shape.java
2  // Definition of interface Shape
3
4  public interface Shape {
5     public abstract double area();
6     public abstract double volume();
7     public abstract String getName();
8  } // end interface Shape
```

Point.java (1 of 2)

```
9  // Fig. 27.5: Point.java
10 // Definition of class Point
11
12 public class Point extends Object implements Shape {
13    protected int x, y; // coordinates of the Point
14
15    // no-argument constructor
16    public Point() { setPoint( 0, 0 ); }
17
18    // constructor
19    public Point( int a, int b ) { setPoint( a, b ); }
20
21    // Set x and y coordinates of Point
22    public void setPoint( int a, int b )
23    {
24       x = a;
25       y = b;
26    } // end method setPoint
27
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Outline

Point.java (2 of 2)

```
28    // get x coordinate
29    public int getX() { return x; }
30
31    // get y coordinate
32    public int getY() { return y; }
33
34    // convert the point into a String representation
35    public String toString()
36       { return "[" + x + ", " + y + "]"; }
37
38    // return the area
39    public double area() { return 0.0; }
40
41    // return the volume
42    public double volume() { return 0.0; }
43
44    // return the class name
45    public String getName() { return "Point"; }
46 } // end class Point
```

Circle.java (1 of 3)

```
1  // Fig. 27.5: Circle.java
2  // Definition of class Circle
3
4  public class Circle extends Point {  // inherits from Point
5     protected double radius;
6
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
7     // no-argument constructor
8     public Circle()
9     {
10       // implicit call to superclass constructor here
11       setRadius( 0 );
12    } // end Circle constructor
13
14    // Constructor
15    public Circle( double r, int a, int b )
16    {
17       super( a, b );  // call the superclass constructor
18       setRadius( r );
19    } // end Circle constructor
20
21    // Set radius of Circle
22    public void setRadius( double r )
23       { radius = ( r >= 0 ? r : 0 ); }
24
25    // Get radius of Circle
26    public double getRadius() { return radius; }
27
28    // Calculate area of Circle
29    public double area() { return Math.PI * radius * radius; }
30
```


© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Outline

Circle.java (3 of 3)

```
31    // convert the Circle to a String
32    public String toString()
33       { return "Center = " + super.toString() +
34                "; Radius = " + radius; }
35
36    // return the class name
37    public String getName() { return "Circle"; }
38 } // end class Circle
```

Cylinder.java (1 of 2)

```
39 // Fig. 27.5: Cylinder.java
40 // Definition of class Cylinder
41
42 public class Cylinder extends Circle {
43    protected double height;  // height of Cylinder
44
45    // no-argument constructor
46    public Cylinder()
47    {
48       // implicit call to superclass constructor here
49       setHeight( 0 );
50    } // end Cylinder constructor
51
52    // constructor
53    public Cylinder( double h, double r, int a, int b )
54    {
55       super( r, a, b );   // call superclass constructor
56       setHeight( h );
57    } // end Cylinder constructor
58
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
59    // Set height of Cylinder
60    public void setHeight( double h )
61       { height = ( h >= 0 ? h : 0 ); }
62
63    // Get height of Cylinder
64    public double getHeight() { return height; }
65
66    // Calculate area of Cylinder (i.e., surface area)
67    public double area()
68    {
69       return 2 * super.area() +
70              2 * Math.PI * radius * height;
71    } // end method area
72
73    // Calculate volume of Cylinder
74    public double volume() { return super.area() * height; }
75
76    // Convert a Cylinder to a String
77    public String toString()
78       { return super.toString() + "; Height = " + height; }
79
80    // Return the class name
81    public String getName() { return "Cylinder"; }
82 } // end class Cylinder
```


© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
83 // Fig. 27.5: Test.java
84 // Driver for point, circle, cylinder hierarchy
85 import javax.swing.JOptionPane;
86 import java.text.DecimalFormat;
87
88 public class Test {
89    public static void main( String args[] )
90    {
91       Point point = new Point( 7, 11 );
92       Circle circle = new Circle( 3.5, 22, 8 );
93       Cylinder cylinder = new Cylinder( 10, 3.3, 10, 10 );
94
95       Shape arrayOfShapes[];
96
97       arrayOfShapes = new Shape[ 3 ];
98
99       // aim arrayOfShapes[0] at subclass Point object
100       arrayOfShapes[ 0 ] = point;
101
102       // aim arrayOfShapes[1] at subclass Circle object
103       arrayOfShapes[ 1 ] = circle;
104
105       // aim arrayOfShapes[2] at subclass Cylinder object
106       arrayOfShapes[ 2 ] = cylinder;
107
```

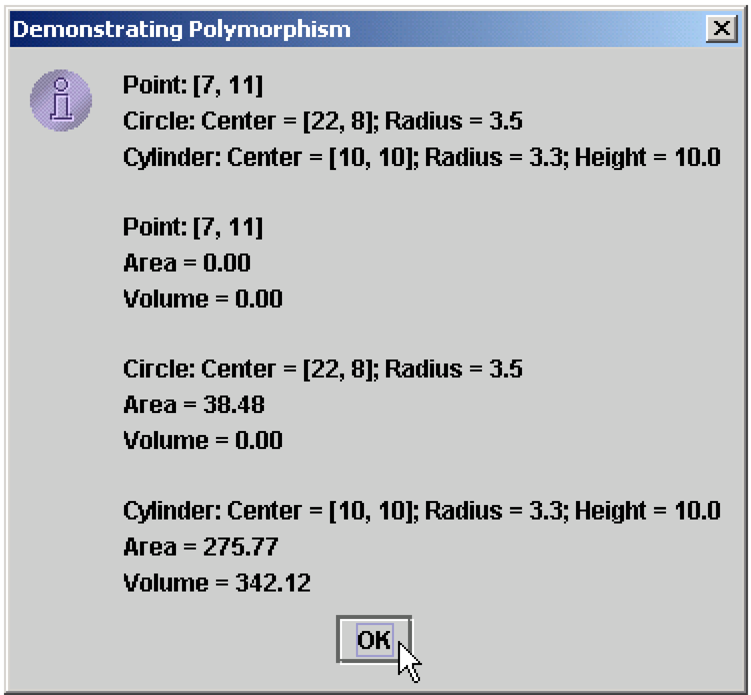

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
108       String output =
109          point.getName() + ": " + point.toString() + "\n" +
110          circle.getName() + ": " + circle.toString() + "\n" +
111          cylinder.getName() + ": " + cylinder.toString();
112
113       DecimalFormat precision2 = new DecimalFormat( "#0.00" );
114
115       // Loop through arrayOfShapes and print the name,
116       // area, and volume of each object.
117       for ( int i = 0; i < arrayOfShapes.length; i++ ) {
118          output += "\n\n" +
119             arrayOfShapes[ i ].getName() + ": " +
120             arrayOfShapes[ i ].toString() +
121             "\nArea = " +
122             precision2.format( arrayOfShapes[ i ].area() ) +
123             "\nVolume = " +
124             precision2.format( arrayOfShapes[ i ].volume() );
125       }
126
127       JOptionPane.showMessageDialog( null, output,
128          "Demonstrating Polymorphism",
129          JOptionPane.INFORMATION_MESSAGE );
130
131       System.exit( 0 );
132    } // end main
133 } // end class Test
```

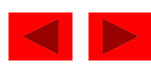

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

27.16 Case Study: Creating and Using Interfaces

Figure 27.5 Point, circle, cylinder hierarchy with a Shape interface—`Circle.java`



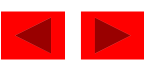
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



27.17 Inner Class Definitions

- Inner classes
  - Till now, all classes defined at file scope (not inside other classes)
  - Inner classes defined inside other classes
  - Anonymous inner class
    - Has no name
    - Frequently used with GUI event handling

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Outline

Time.java (1 of 3)

```
1  // Fig. 27.6: Time.java
2  // Time class definition
3  import java.text.DecimalFormat;  // used for number formatting
4
5  // This class maintains the time in 24-hour format
6  public class Time extends Object {
7     private int hour;     // 0 - 23
8     private int minute;   // 0 - 59
9     private int second;   // 0 - 59
10
11    // Time constructor initializes each instance variable
12    // to zero. Ensures that Time object starts in a
13    // consistent state.
14    public Time() { setTime( 0, 0, 0 ); }
15
16    // Set a new time value using universal time. Perform
17    // validity checks on the data. Set invalid values to zero.
18    public void setTime( int h, int m, int s )
19    {
20       setHour( h );    // set the hour
21       setMinute( m );  // set the minute
22       setSecond( s );  // set the second
23    } // end method setTime
24
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
25    // set the hour
26    public void setHour( int h )
27       { hour = ( ( h >= 0 && h < 24 ) ? h : 0 ); }
28
29    // set the minute
30    public void setMinute( int m )
31       { minute = ( ( m >= 0 && m < 60 ) ? m : 0 ); }
32
33    // set the second
34    public void setSecond( int s )
35       { second = ( ( s >= 0 && s < 60 ) ? s : 0 ); }
36
37    // get the hour
38    public int getHour() { return hour; }
39
40    // get the minute
41    public int getMinute() { return minute; }
42
43    // get the second
44    public int getSecond() { return second; }
45
46    // Convert to String in standard-time format
47    public String toString()
48    {
49       DecimalFormat twoDigits = new DecimalFormat( "00" );
50
```


© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
51       return ( ( getHour() == 12 || getHour() == 0 ) ?
52                  12 : getHour() % 12 ) + ":" +
53              twoDigits.format( getMinute() ) + ":" +
54              twoDigits.format( getSecond() ) +
55              ( getHour() < 12 ? " AM" : " PM" );
56    } // end method toString
57 } // end class Time
```

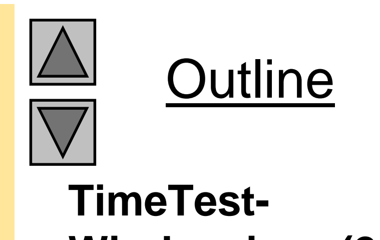
```
58 // Fig. 27.6: TimeTestWindow.java
59 // Demonstrating the Time class set and get methods
60 import java.awt.*;
61 import java.awt.event.*;
62 import javax.swing.*;
63
64 public class TimeTestWindow extends JFrame {
65    private Time t;
66    private JLabel hourLabel, minuteLabel, secondLabel;
67    private JTextField hourField, minuteField,
68                       secondField, display;
69    private JButton exitButton;
70
71    public TimeTestWindow()
72    {
73       super( "Inner Class Demonstration" );
74
75       t = new Time();
76
77       Container c = getContentPane();
78
```

Outline

Time.java (3 of 3)

TimeTest-Window.java (1 of 4)

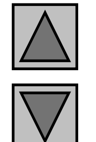
```
79        // create an instance of the inner class
80        ActionEventHandler handler = new ActionEventHandler();
81
82        c.setLayout( new FlowLayout() );
83        hourLabel = new JLabel( "Set Hour" );
84        hourField = new JTextField( 10 );
85        hourField.addActionListener( handler );
86        c.add( hourLabel );
87        c.add( hourField );
88
89        minuteLabel = new JLabel( "Set minute" );
90        minuteField = new JTextField( 10 );
91        minuteField.addActionListener( handler );
92        c.add( minuteLabel );
93        c.add( minuteField );
94
95        secondLabel = new JLabel( "Set Second" );
96        secondField = new JTextField( 10 );
97        secondField.addActionListener( handler );
98        c.add( secondLabel );
99        c.add( secondField );
100
101        display = new JTextField( 30 );
102        display.setEditable( false );
103        c.add( display );
104
```



TimeTest-Window.java (2 of 4)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
105       exitButton = new JButton( "Exit" );
106       exitButton.addActionListener( handler );
107       c.add( exitButton );
108    } // end TimeTestWindow constructor
109
110    public void displayTime()
111    {
112       display.setText( "The time is: " + t );
113    } // end method displayTime
114
115    public static void main( String args[] )
116    {
117       TimeTestWindow window = new TimeTestWindow();
118
119       window.setSize( 400, 140 );
120       window.show();
121    } // end main
122
123    // Inner class definition for event handling
124    private class ActionEventHandler implements ActionListener {
125       public void actionPerformed( ActionEvent e )
126       {
127          if ( e.getSource() == exitButton )
128             System.exit( 0 );    // terminate the application
129          else if ( e.getSource() == hourField ) {
130             t.setHour(
131                Integer.parseInt( e.getActionCommand() ) );
132             hourField.setText( "" );
133          }
```



TimeTest-Window.java (3 of 4)

```
134          else if ( e.getSource() == minuteField ) {
135             t.setMinute(
136                Integer.parseInt( e.getActionCommand() ) );
137             minuteField.setText( "" );
138          }
139          else if ( e.getSource() == secondField ) {
140             t.setSecond(
141                Integer.parseInt( e.getActionCommand() ) );
142             secondField.setText( "" );
143          }
144
145          displayTime();
146       } // end method actionPerformed
147    } // end class ActionEventHandler
148 } // end class TimeTestWindow
```

Outline

TimeTest-Window.java (4 of 4)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

27.17 Inner Class Definitions

Figure 27.6 Demonstrating an inner class in a windowed application–`TimeTestWindow.java`.

Title Bar

Close box

Maximize

Minimize

Inner Class Demonstration

Set Hour 13 Set Minute

Set Second

Exit

Inner Class Demonstration

Set Hour Set Minute

Set Second

The time is: 1:00:00 PM

Exit

Inner Class Demonstration

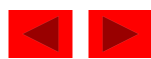
Set Hour Set Minute 26

Set Second

The time is: 1:00:00 PM

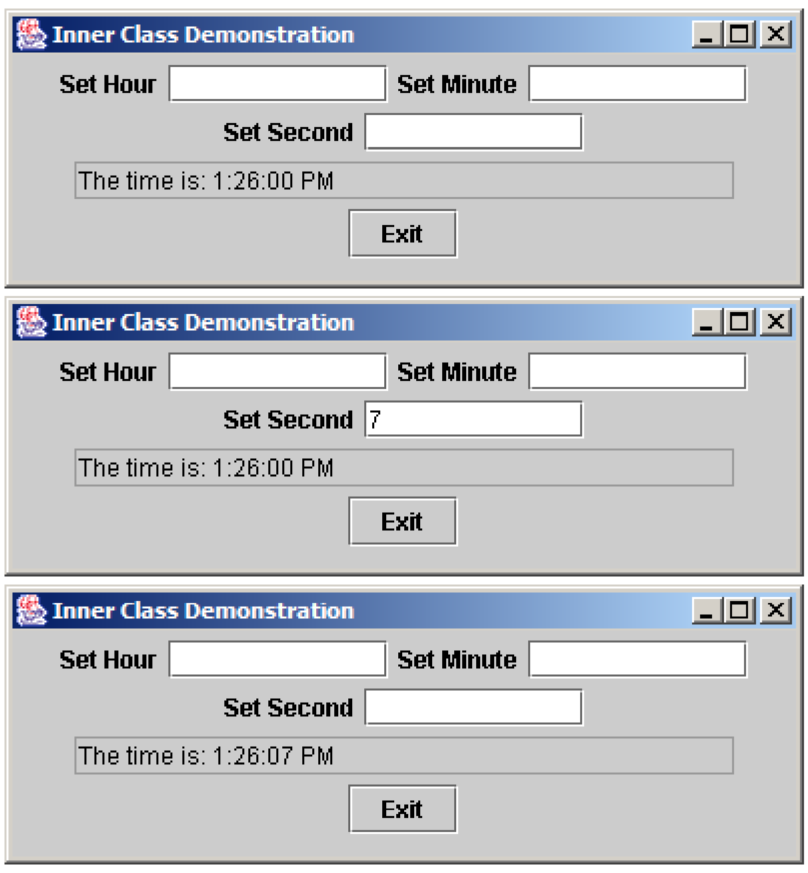
Exit

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

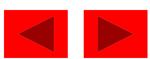


27.17 Inner Class Definitions

Figure 27.6 Demonstrating an inner class in a windowed application–`TimeTestWindow.java`.



© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



27.17 Inner Class Definitions

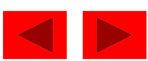
- Windowed applications
  - Execute an application in its own window (like an Applet)
    - Inherit from class `JFrame` (`javax.swing`) rather than `JApplet`
  - `init` method replaced by constructor
    - Instead, create GUI components in constructor
    - Instantiate object in `main` (guaranteed to be called)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

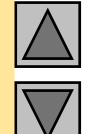
27.17 Inner Class Definitions

- Event handling
  - Some class must implement interface `ActionListener`
    - Must define method `actionPerformed`
    - Class that implements `ActionListener` "is an" `ActionListener`
  - Method `addActionListener`
    - Takes object of type `ActionListener`
    - We can pass it an instance of the class that implements `ActionListener` ("is a" relationship)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



```
1  // Fig. 27.7: TimeTestWindow.java
2  // Demonstrating the Time class set and get methods
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class TimeTestWindow extends JFrame {
8     private Time t;
9     private JLabel hourLabel, minuteLabel, secondLabel;
10    private JTextField hourField, minuteField,
11                       secondField, display;
12
13    public TimeTestWindow()
14    {
15       super( "Inner Class Demonstration" );
16
17       t = new Time();
18
19       Container c = getContentPane();
20
21       c.setLayout( new FlowLayout() );
22       hourLabel = new JLabel( "Set Hour" );
23       hourField = new JTextField( 10 );
```

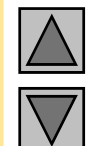


Outline

TimeTest-Window.java (1 of 4)

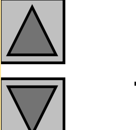
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
24        hourField.addActionListener(
25           new ActionListener() {  // anonymous inner class
26              public void actionPerformed( ActionEvent e )
27              {
28                 t.setHour(
29                    Integer.parseInt( e.getActionCommand() ) );
30                 hourField.setText( "" );
31                 displayTime();
32              } // end method actionPerformed
33           } // end anonymous inner class
34        ); // end addActionListener
35        c.add( hourLabel );
36        c.add( hourField );
37
38        minuteLabel = new JLabel( "Set minute" );
39        minuteField = new JTextField( 10 );
40        minuteField.addActionListener(
41           new ActionListener() {  // anonymous inner class
42              public void actionPerformed( ActionEvent e )
43              {
44                 t.setMinute(
45                    Integer.parseInt( e.getActionCommand() ) );
46                 minuteField.setText( "" );
47                 displayTime();
48              }
49           }
50        );
```



TimeTest-Window.java (2 of 4)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



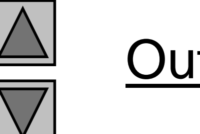
Outline

TimeTest-Window.java (3 of 4)

```
51       c.add( minuteLabel );
52       c.add( minuteField );
53
54       secondLabel = new JLabel( "Set Second" );
55       secondField = new JTextField( 10 );
56       secondField.addActionListener(
57          new ActionListener() {  // anonymous inner class
58             public void actionPerformed( ActionEvent e )
59             {
60                t.setSecond(
61                   Integer.parseInt( e.getActionCommand() ) );
62                secondField.setText( "" );
63                displayTime();
64             } // end method actionPerformed
65          } // end anonymous inner class
66       ); // end addActionListener
67       c.add( secondLabel );
68       c.add( secondField );
69
70       display = new JTextField( 30 );
71       display.setEditable( false );
72       c.add( display );
73    } // end TimeTestWindow constructor
74
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
75    public void displayTime()
76    {
77       display.setText( "The time is: " + t );
78    } // end method displayTime
79
80    public static void main( String args[] )
81    {
82       TimeTestWindow window = new TimeTestWindow();
83
84       window.addWindowListener(
85          new WindowAdapter() {
86             public void windowClosing( WindowEvent e )
87             {
88                System.exit( 0 );
89             } // end method windowClosing
90          } // end anonymous inner class
91       ); // end addWindowListener
92
93       window.setSize( 400, 120 );
94       window.show();
95    } // end main
96 } // end class TimeTestWindow
```



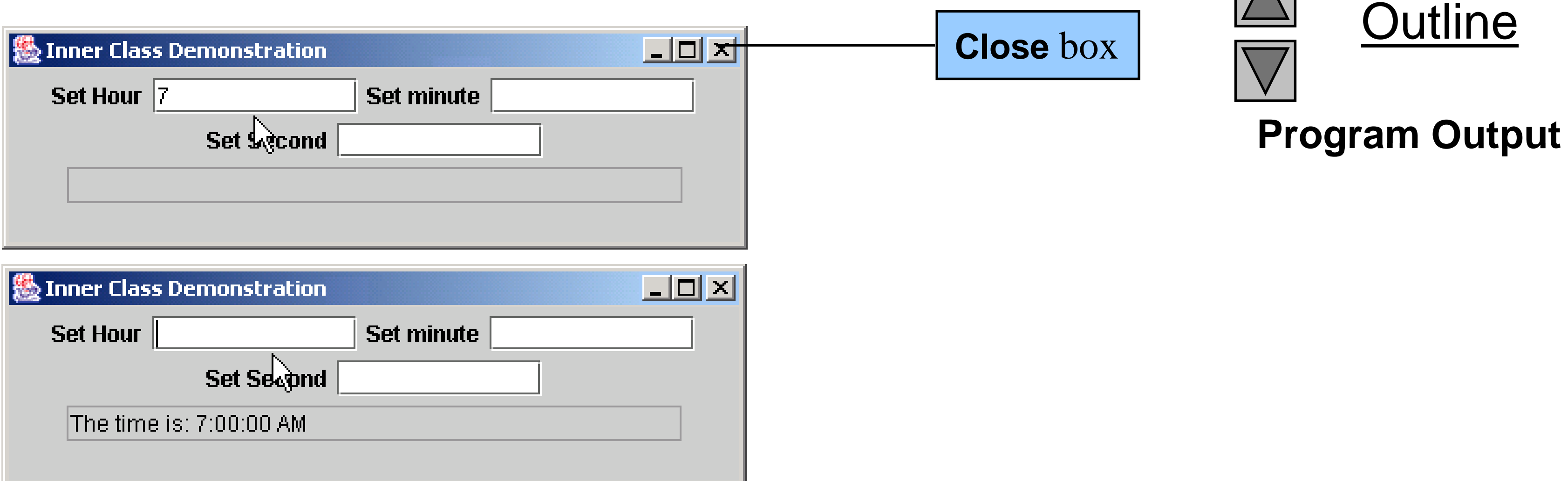
Outline

TimeTest-Window.java (4 of 4)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Outline

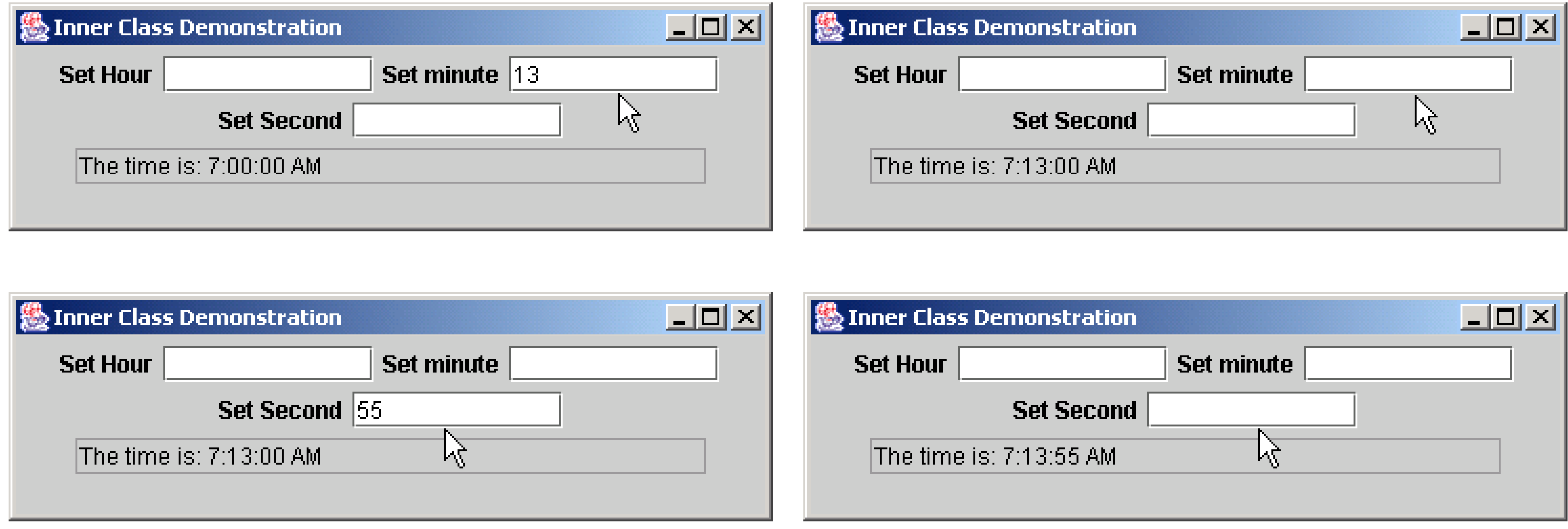
Program Output



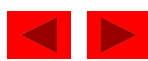
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

27.17 Inner Class Definitions

Figure 27.7 Demonstrating anonymous inner class - `TimeTestWindow.java`



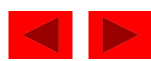
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



27.17 Inner Class Definitions

- Event handling with anonymous Inner classes
  - Define the inner class inside the call to `addActionListener`
    - Create an instance of the class inside the method call
    - `addActionListener` takes an object of class `ActionListener`

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



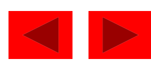
27.17 Inner Class Definitions

- Example:

```
myField.addActionListener(
     new ActionListener() {  // anonymous inner class
       public void actionPerformed( ActionEvent e)
       {
            Actions
       }
     }
);
```

  - `new` creates an object
  - `ActionListener()` begins definition of anonymous class and calls default constructor
    - Similar to `public class myHandler implements ActionListener`
  - Brace (`{`) begins class definition

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

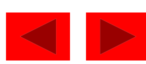


27.17 Inner Class Definitions

- Use the following code to allow the user to close windows using the close button:

```
window.addWindowListener(
   new WindowAdapter() {
      public void windowClosing( WindowEvent e )
      {
         System.exit( 0 );
      }
   }
);
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

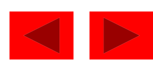


27.18 Notes on Inner Class Definitions

- Notes
  - Every class (including inner classes) have their own `.class` file
  - Named inner classes can be `public`, `protected`, `private`, or have package access
    - Same restrictions as other members of a class
  - To access outer class's `this` reference
    - `OuterClassName.this`
  - To create an object of another class's inner class
    - Create an object of outer class and assign it a reference (`ref`)
    - Type statement of form:

OuterClassName`.`*InnerClassName* `innerRef = ref.new` *InnerClassName*`();`

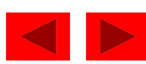
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



27.18 Notes on Inner Class Definitions

- Notes (continued)
  - Inner class can be `static`
    - Does not require object of outer class to be defined
    - Does not have access to outer class's non-`static` members

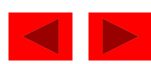
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



27.19 Type-Wrapper Classes for Primitive Types

- Each primitive type has a type-wrapper class
  - Enables manipulation of primitive types as objects of class `Object`
  - Each type wrapper is declared `final`, so their methods are implicitly `final` and may not be overridden
  - To manipulate a primitive value in your program, first refer to the documentation for the type-wrapper classes—the required method may already be defined.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

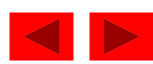


Chapter 28 - Java Graphics and Java2D

Outline

28.1	Introduction
28.2	Graphics Contexts and Graphics Objects
28.3	Color Control
28.4	Font Control
28.5	Drawing Lines, Rectangles and Ovals
28.6	Drawing Arcs
28.7	Drawing Polygons and Polylines
28.8	The Java2D API
28.9	Java2D Shapes

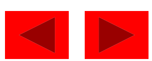
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Objectives

- In this chapter, you will learn:
  - To understand graphics contexts and graphics objects.
  - To understand and be able to manipulate colors.
  - To understand and be able to manipulate fonts.
  - To understand and be able to use `Graphics` methods for drawing lines, rectangles, rectangles with rounded corners, three-dimensional rectangles, ovals, arcs and polygons.
  - To use methods of class `Graphics2D` from the Java2D API to draw lines, rectangles, rectangles with rounded corners, ellipses, arcs and general paths.
  - To be able to specify `Paint` and `Stroke` characteristics of shapes displayed with `Graphics2D`.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



28.1 Introduction

- In this chapter
  - Draw 2D shapes
  - Colors
  - Fonts
- Java appealing for its graphics support
  - Has a class hierarchy for its graphics classes and 2D API classes

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

28.1 Introduction

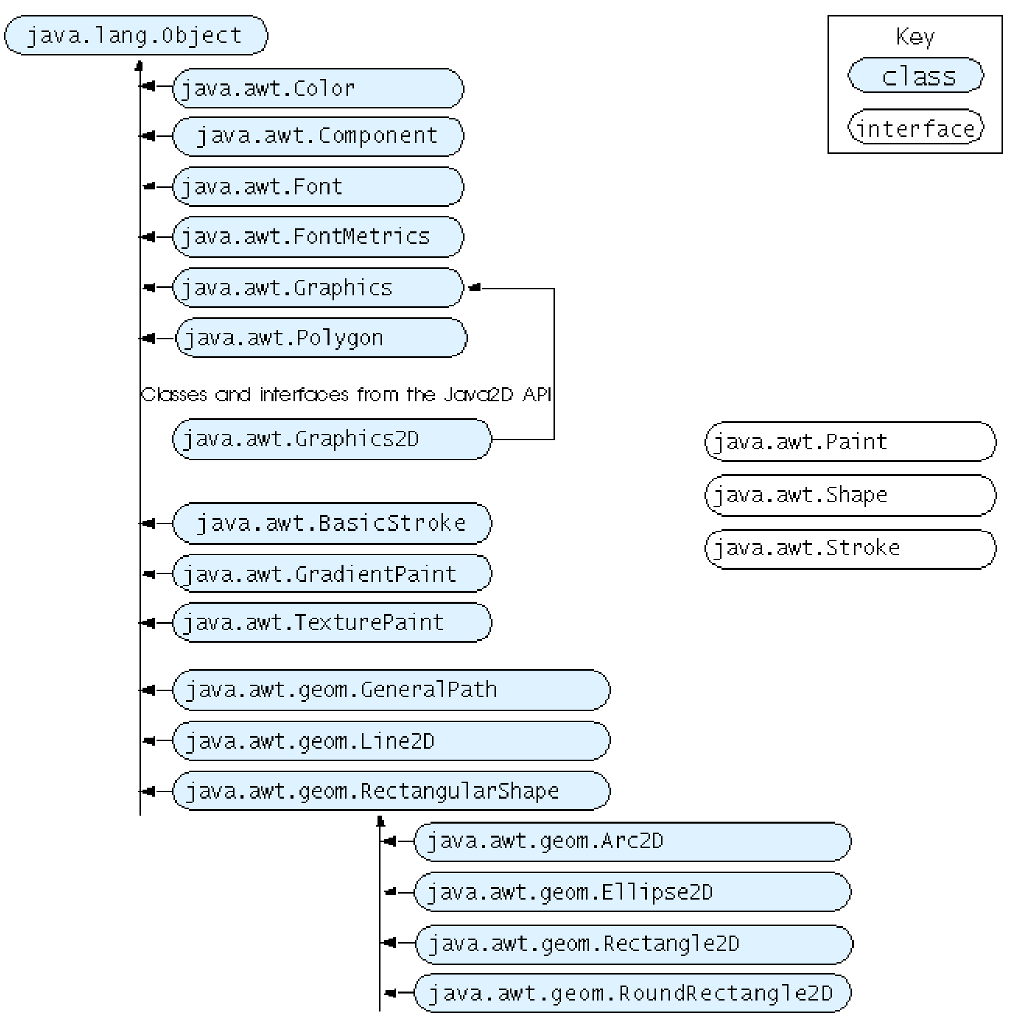
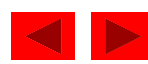


Fig. 28.1 Some classes and interfaces used in this chapter from Java's original graphics capabilities and from the Java2D API.

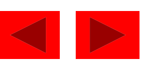
© Copyright 1992–200



28.1 Introduction

- Java coordinate system
  - (x,y) pairs
    - x - horizontal axis
    - y - vertical axis
  - Upper left corner is (0,0)
  - Coordinates measured in pixels (smallest unit of resolution)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



28.1 Introduction

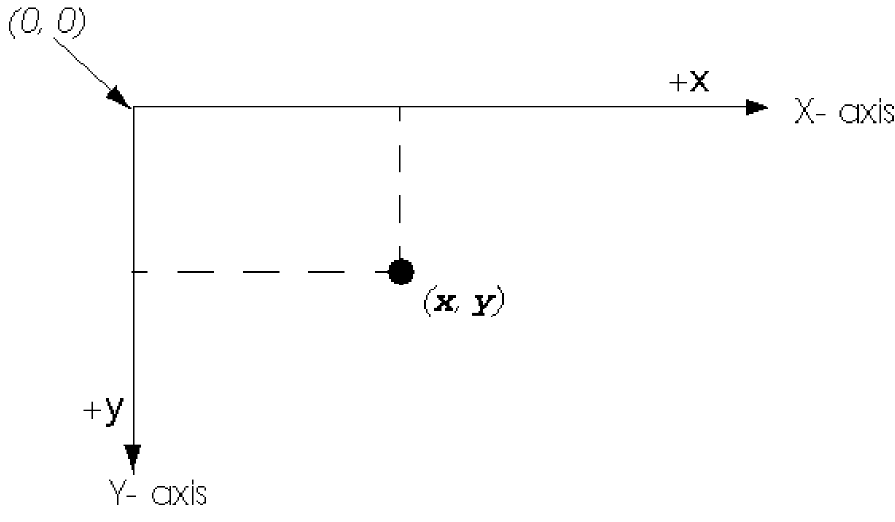
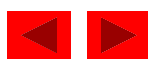


Fig. 28.2 Java coordinate system. Units are measured in pixels.

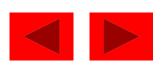
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



28.2 Graphics Contexts and Graphics Objects

- Graphics context
  - Enables drawing on screen
  - `Graphics` object manages graphics context
    - Controls how information is drawn
    - Has methods for drawing, font manipulation, etc
    - We have used `Graphics` object `g` for applets
  - `Graphics` an `abstract` class
    - Cannot instantiate objects
    - Implementation hidden - more portable
- Class `Component`
  - Superclass for many classes in `java.awt`
  - Method `paint` takes `Graphics` object as argument

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



28.2 Graphics Contexts and Graphics Objects

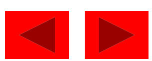
- Class `Component`
  - `paint` called automatically when applet starts
  - `paint` often called in response to an event
    - Drawing graphics is an event-driven process.
  - `repaint` calls `update`, which forces a `paint` operation
    - `update` rarely called directly
    - Sometimes overridden to reduce "flicker"

    Headers:

```
public void repaint()
public void update( Graphics g )
```

- In this chapter
  - Focus on `paint` method

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



28.3 Color Control

- Class `Color`
  - Defines methods and constants for manipulating colors
  - Colors created from red, green, and blue component
    - RGB value: 3 integers from 0 to 255 each, or three floating point values from 0 to 1.0 each
    - Larger the value, more of that color
  - Color methods `getRed`, `getGreen`, `getBlue` return an integer between 0 and 255 representing amount
  - Graphics method `setColor` sets drawing color
    - Takes `Color` object
  - Method `getColor` gets current color setting

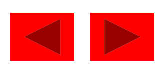
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

28.3 Color Control

`Color` Constant	Color	RGB value
`public final static Color orange`	orange	255, 200, 0
`public final static Color pink`	pink	255, 175, 175
`public final static Color cyan`	cyan	0, 255, 255
`public final static Color magenta`	magenta	255, 0, 255
`public final static Color yellow`	yellow	255, 255, 0
`public final static Color black`	black	0, 0, 0
`public final static Color white`	white	255, 255, 255
`public final static Color gray`	gray	128, 128, 128
`public final static Color lightGray`	light gray	192, 192, 192
`public final static Color darkGray`	dark gray	64, 64, 64
`public final static Color red`	red	255, 0, 0
`public final static Color green`	green	0, 255, 0
`public final static Color blue`	blue	0, 0, 255

Fig. 28.3 `Color` class `static` constants and RGB values.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

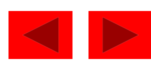


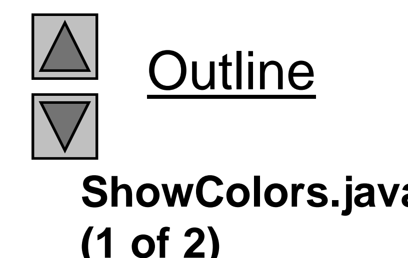
28.3 Color Control

Method	Description
`public Color( int r, int g, int b )`	Creates a color based on red, green and blue contents expressed as integers from 0 to 255.
`public Color( float r, float g, float b )`	Creates a color based on red, green and blue contents expressed as floating-point values from 0.0 to 1.0.
`public int getRed() // Color class`	Returns a value between 0 and 255 representing the red content.
`public int getGreen() // Color class`	Returns a value between 0 and 255 representing the green content.
`public int getBlue() // Color class`	Returns a value between 0 and 255 representing the blue content.
`public Color getColor() // Graphics class`	Returns a `Color` object representing the current color for the graphics context.
`public void setColor( Color c ) // Graphics class`	Sets the current color for drawing with the graphics context.

Fig. 28.4 `Color` methods and color-related `Graphics` methods.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



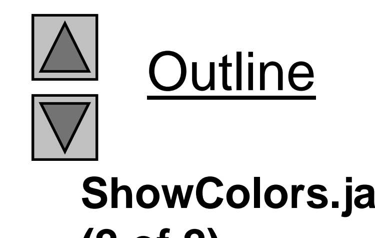


ShowColors.java (1 of 2)

```
1  // Fig. 28.5: ShowColors.java
2  // Demonstrating Colors
3  import java.awt.*;
4  import javax.swing.*;
5  import java.awt.event.*;
6
7  public class ShowColors extends JFrame {
8     public ShowColors()
9     {
10       super( "Using colors" );
11
12       setSize( 400, 130 );
13       show();
14    } // end ShowColors constructor
15
16    public void paint( Graphics g )
17    {
18       // set new drawing color using integers
19       g.setColor( new Color( 255, 0, 0 ) );
20       g.fillRect( 25, 25, 100, 20 );
21       g.drawString( "Current RGB: " + g.getColor(), 130, 40 );
22
23       // set new drawing color using floats
24       g.setColor( new Color( 0.0f, 1.0f, 0.0f ) );
25       g.fillRect( 25, 50, 100, 20 );
26       g.drawString( "Current RGB: " + g.getColor(), 130, 65 );
27
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
28       // set new drawing color using static Color objects
29       g.setColor( Color.blue );
30       g.fillRect( 25, 75, 100, 20 );
31       g.drawString( "Current RGB: " + g.getColor(), 130, 90 );
32
33       // display individual RGB values
34       Color c = Color.magenta;
35       g.setColor( c );
36       g.fillRect( 25, 100, 100, 20 );
37       g.drawString( "RGB values: " + c.getRed() + ", " +
38          c.getGreen() + ", " + c.getBlue(), 130, 115 );
39    } // end method paint
40
41    public static void main( String args[] )
42    {
43       ShowColors app = new ShowColors();
44
45       app.addWindowListener(
46          new WindowAdapter() {
47             public void windowClosing( WindowEvent e )
48             {
49                System.exit( 0 );
50             } // end method windowClosing
51          } // end anonymous inner class
52       ); // end addWindowListener
53    } // end method main
54 } // end class ShowColors
```



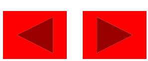
ShowColors.java (2 of 2)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

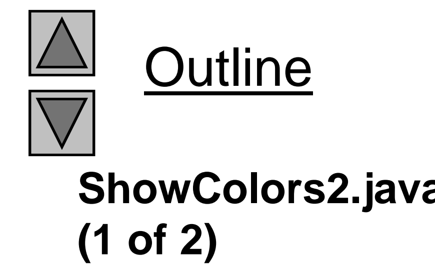
28.3 Color Control

- Component `JColorChooser`
  - Displays dialog allowing user to select a color
  - Method `showDialog`
    - First argument: reference to parent `Component` (window from which dialog being displayed)
      - Modal dialog - user cannot interact with other dialogs while active
    - Second argument: `String` for title bar
    - Third argument: Initial selected color
    - Returns a `Color` object
- Class `Container`
  - Method `setBackground` - takes `Color` object
  - Sets background color

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

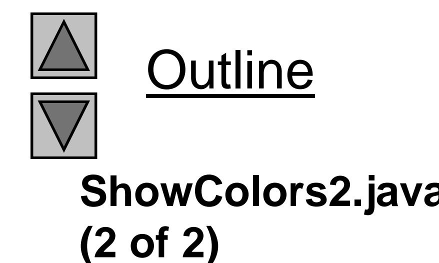


```java
1  // Fig. 28.6: ShowColors2.java
2  // Demonstrating JColorChooser
3  import java.awt.*;
4  import javax.swing.*;
5  import java.awt.event.*;
6
7  public class ShowColors2 extends JFrame {
8     private JButton changeColor;
9     private Color color = Color.lightGray;
10     private Container c;
11
12     public ShowColors2()
13     {
14        super( "Using JColorChooser" );
15
16        c = getContentPane();
17        c.setLayout( new FlowLayout() );
18
19        changeColor = new JButton( "Change Color" );
20        changeColor.addActionListener(
21           new ActionListener() {
22              public void actionPerformed( ActionEvent e )
23              {
24                 color =
25                    JColorChooser.showDialog( ShowColors2.this,
26                       "Choose a color", color );
27
```



ShowColors2.java

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



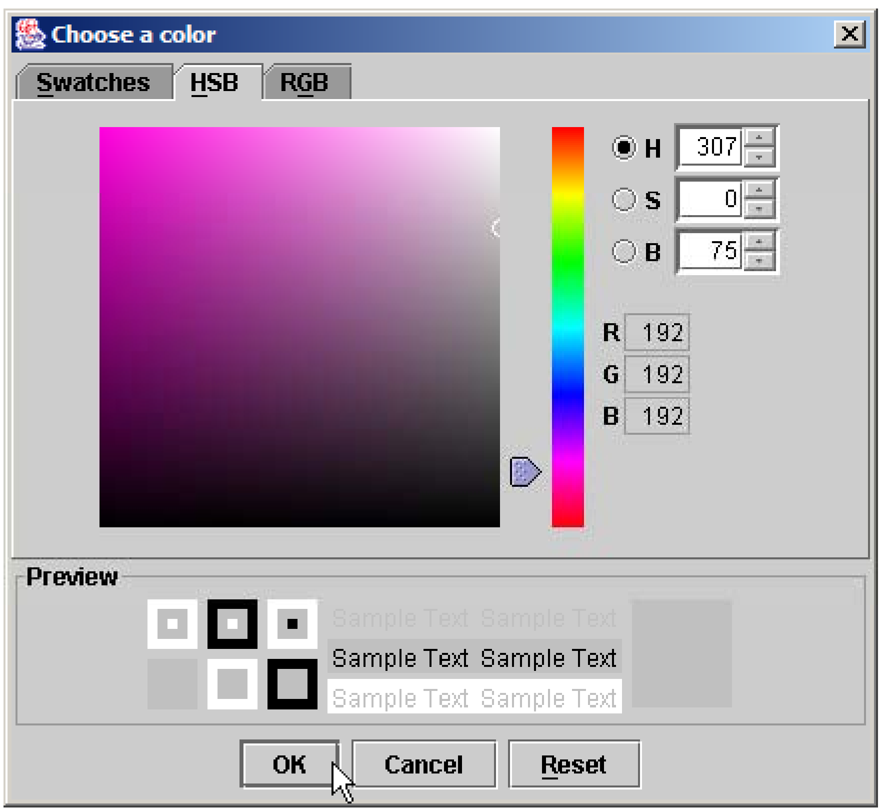
ShowColors2.java (2 of 2)

```
28              if ( color == null )
29                 color = Color.lightGray;
30
31              c.setBackground( color );
32              c.repaint();
33           } // end method actionPerformed
34        } // end anonymous inner class
35     ); // end addActionListener
36     c.add( changeColor );
37
38     setSize( 400, 130 );
39     show();
40  } // end ShowColors2 constructor
41
42  public static void main( String args[] )
43  {
44     ShowColors2 app = new ShowColors2();
45
46     app.addWindowListener(
47        new WindowAdapter() {
48           public void windowClosing( WindowEvent e )
49           {
50              System.exit( 0 );
51           } // end method windowClosing
52        } // end anonymous inner class
53     ); // end addWindowListener
54  } // end main
55 } // end class ShowColors2
```

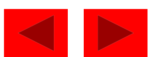
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

28.3 Color Control

Fig. 28.7 The HSB and RGB tabs of the `JColorChooser` dialog.

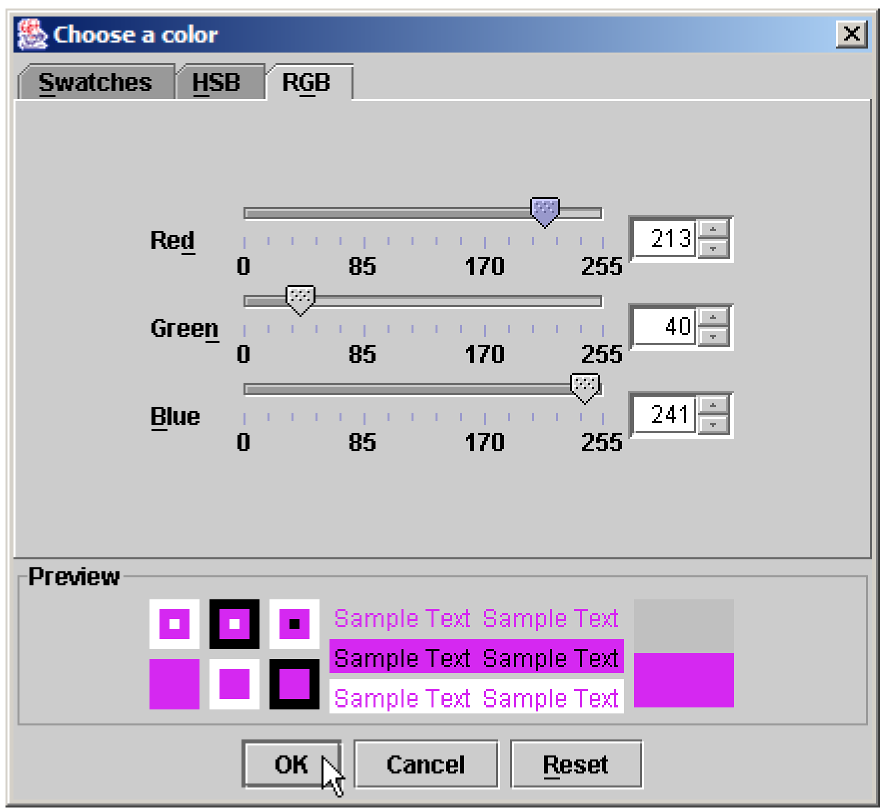


© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

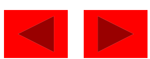


28.3 Color Control

Fig. 28.7 The HSB and RGB tabs of the `JColorChooser` dialog.



© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



28.4 Font Control

Method or constant	Description
`public final static int PLAIN // Font class`	A constant representing a plain font style.
`public final static int BOLD // Font class`	A constant representing a bold font style.
`public final static int ITALIC // Font class`	A constant representing an italic font style.
`public Font( String name, int style, int size )`	Creates a `Font` object with the specified font, style and size.
`public int getStyle() // Font class`	Returns an integer value indicating the current font style.
`public int getSize() // Font class`	Returns an integer value indicating the current font size.

Fig. 28.8 `Font` methods, constants and font-related `Graphics` methods (Part 1 of 2).

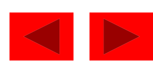
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

28.4 Font Control

Method or constant	Description
`public String getName() // Font class`	Returns the current font name as a string.
`public String getFamily() // Font class`	Returns the font's family name as a string.
`public boolean isPlain() // Font class`	Tests a font for a plain font style. Returns `true` if the font is plain.
`public boolean isBold() // Font class`	Tests a font for a bold font style. Returns `true` if the font is bold.
`public boolean isItalic() // Font class`	Tests a font for an italic font style. Returns `true` if the font is italic.
`public Font getFont() // Graphics class`	Returns a `Font` object reference representing the current font.
`public void setFont(Font f) // Graphics class`	Sets the current font to the font, style and size specified by the `Font` object reference `f`.

Fig. 28.8 `Font` methods, constants and font-related `Graphics` methods (Part 2 of 2).

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



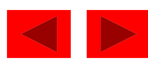
28.4 Font Control

- Class `Font`
  - Constructor takes three arguments

```
public Font( String name, int style, int size)
```

    - name: any font supported by system (`Serif`, `Monospaced`)
    - style: constants `FONT.PLAIN`, `FONT.ITALIC`, `FONT.BOLD`
      - Combinations: `FONT.ITALIC + FONT.BOLD`
    - size: measured in points (1/72 of an inch)
  - Usage:
    - `g.setFont( fontObject );`

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Outline

Fonts.java (1 of 2)

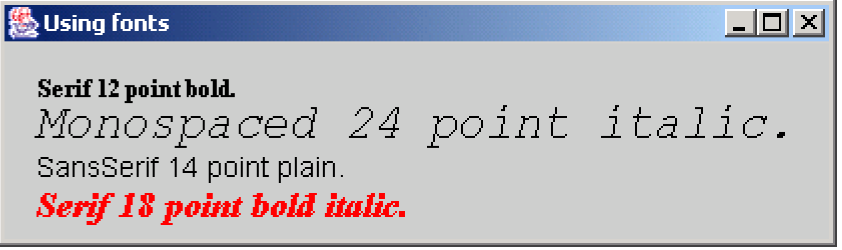
```
1  // Fig. 28.9: Fonts.java
2  // Using fonts
3  import java.awt.*;
4  import javax.swing.*;
5  import java.awt.event.*;
6
7  public class Fonts extends JFrame {
8     public Fonts()
9     {
10       super( "Using fonts" );
11
12       setSize( 420, 125 );
13       show();
14    } // end Fonts constructor
15
16    public void paint( Graphics g )
17    {
18       // set current font to Serif (Times), bold, 12pt
19       // and draw a string
20       g.setFont( new Font( "Serif", Font.BOLD, 12 ) );
21       g.drawString( "Serif 12 point bold.", 20, 50 );
22
23       // set current font to Monospaced (Courier),
24       // italic, 24pt and draw a string
25       g.setFont( new Font( "Monospaced", Font.ITALIC, 24 ) );
26       g.drawString( "Monospaced 24 point italic.", 20, 70 );
27
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

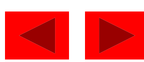
```
28       // set current font to SansSerif (Helvetica),
29       // plain, 14pt and draw a string
30       g.setFont( new Font( "SansSerif", Font.PLAIN, 14 ) );
31       g.drawString( "SansSerif 14 point plain.", 20, 90 );
32
33       // set current font to Serif (times), bold/italic,
34       // 18pt and draw a string
35       g.setColor( Color.red );
36       g.setFont(
37          new Font( "Serif", Font.BOLD + Font.ITALIC, 18 ) );
38       g.drawString( g.getFont().getName() + " " +
39                     g.getFont().getSize() +
40                     " point bold italic.", 20, 110 );
41    } // end method paint
42
43    public static void main( String args[] )
44    {
45       Fonts app = new Fonts();
46
47       app.addWindowListener(
48          new WindowAdapter() {
49             public void windowClosing( WindowEvent e )
50             {
51                System.exit( 0 );
52             } // end method windowClosing
53          } // end anonymous inner class
54       ); // end addWindowListener
55    } // end main
56 } // end class Fonts
```

28.4 Font Control

Figure 28.9 Using `Graphics` method `setFont` to change `Fonts`.



© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



28.4 Font Control

- Class `FontMetrics`
  - Has methods for getting font metrics
  - `g.getFontMetrics` - returns `FontMetrics` object

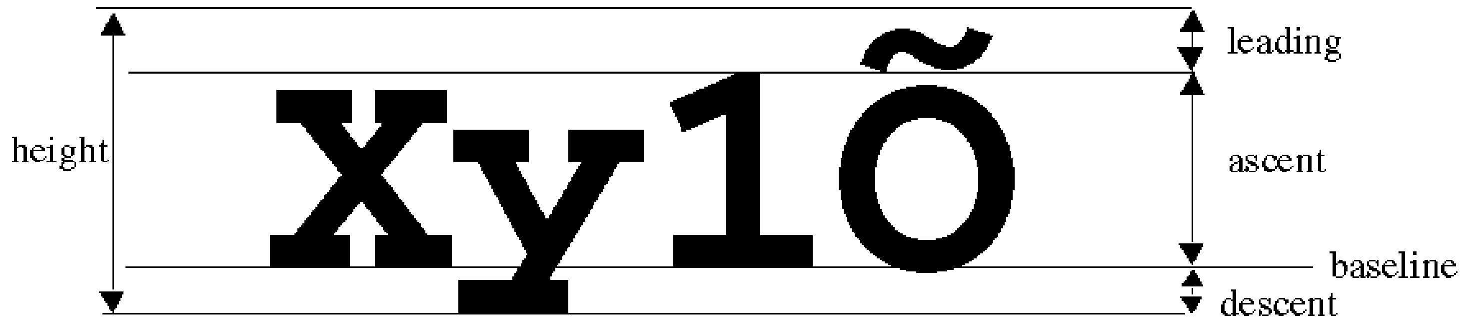
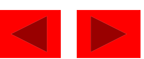


Fig. 28.10 Font metrics.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

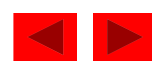


28.4 Font Control

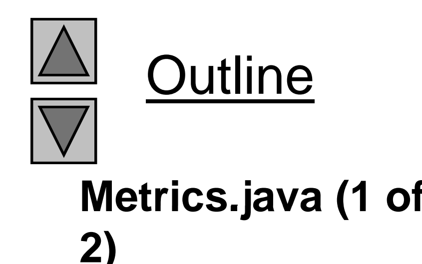
Method	Description
`public int getAscent()` `// FontMetrics class`	Returns a value representing the ascent of a font in points.
`public int getDescent()` `// FontMetrics class`	Returns a value representing the descent of a font in points.
`public int getLeading()` `// FontMetrics class`	Returns a value representing the leading of a font in points.
`public int getHeight()` `// FontMetrics class`	Returns a value representing the height of a font in points.
`public FontMetrics getFontMetrics()` `// Graphics class`	Returns the `FontMetrics` object for the current drawing `Font`.
`public FontMetrics getFontMetrics( Font f )` `// Graphics class`	Returns the `FontMetrics` object for the specified `Font` argument.

Fig. 28.11 `FontMetrics` and `Graphics` methods for obtaining font metrics.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



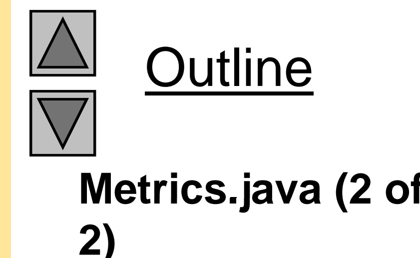
```
1  // Fig. 28.12: Metrics.java
2  // Demonstrating methods of class FontMetrics and
3  // class Graphics useful for obtaining font metrics
4  import java.awt.*;
5  import java.awt.event.*;
6  import javax.swing.*;
7
8  public class Metrics extends JFrame {
9     public Metrics()
10    {
11       super( "Demonstrating FontMetrics" );
12
13       setSize( 510, 210 );
14       show();
15    } // end Metrics constructor
16
17    public void paint( Graphics g )
18    {
19       g.setFont( new Font( "SansSerif", Font.BOLD, 12 ) );
20       FontMetrics fm = g.getFontMetrics();
21       g.drawString( "Current font: " + g.getFont(), 10, 40 );
22       g.drawString( "Ascent: " + fm.getAscent(), 10, 55 );
23       g.drawString( "Descent: " + fm.getDescent(), 10, 70 );
24       g.drawString( "Height: " + fm.getHeight(), 10, 85 );
25       g.drawString( "Leading: " + fm.getLeading(), 10, 100 );
26
```



Metrics.java (1 of 2)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
27       Font font = new Font( "Serif", Font.ITALIC, 14 );
28       fm = g.getFontMetrics( font );
29       g.setFont( font );
30       g.drawString( "Current font: " + font, 10, 130 );
31       g.drawString( "Ascent: " + fm.getAscent(), 10, 145 );
32       g.drawString( "Descent: " + fm.getDescent(), 10, 160 );
33       g.drawString( "Height: " + fm.getHeight(), 10, 175 );
34       g.drawString( "Leading: " + fm.getLeading(), 10, 190 );
35    } // end method paint
36
37    public static void main( String args[] )
38    {
39       Metrics app = new Metrics();
40
41       app.addWindowListener(
42          new WindowAdapter() {
43             public void windowClosing( WindowEvent e )
44             {
45                System.exit( 0 );
46             } // end method windowClosing
47          } // end anonymous inner class
48       ); // end addWindowListener
49    } // end main
50 } // end class Metrics
```

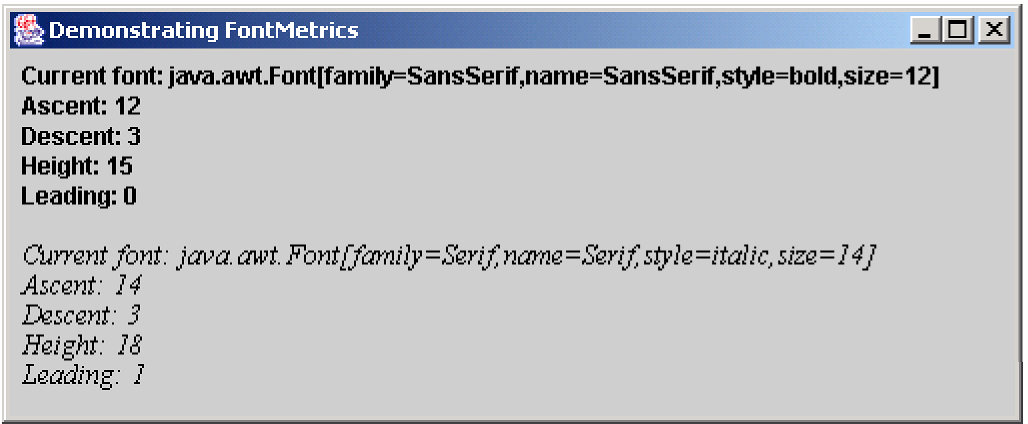


Metrics.java (2 of 2)

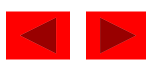
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

28.4 Font Control

Figure 28.12 Obtaining font metric information.



© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

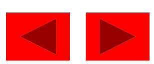


28.5 Drawing Lines, Rectangles and Ovals

Method	Description
`public void drawLine( int x1, int y1, int x2, int y2 )`	Draws a line between the point (x1, y1) and the point (x2, y2).
`public void drawRect( int x, int y, int width, int height )`	Draws a rectangle of the specified width and height. The top-left corner of the rectangle has the coordinates (x, y).
`public void fillRect( int x, int y, int width, int height )`	Draws a solid rectangle with the specified width and height. The top-left corner of the rectangle has the coordinate (x, y).
`public void clearRect( int x, int y, int width, int height )`	Draws a solid rectangle with the specified width and height in the current background color. The top-left corner of the rectangle has the coordinate (x, y).
`public void drawRoundRect( int x, int y, int width, int height, int arcWidth, int arcHeight )`	Draws a rectangle with rounded corners in the current color with the specified width and height. The arcWidth and arcHeight determine the rounding of the corners (see Fig. 28.15).
`public void fillRoundRect( int x, int y, int width, int height, int arcWidth, int arcHeight )`	Draws a solid rectangle with rounded corners in the current color with the specified width and height. The arcWidth and arcHeight determine the rounding of the corners (see Fig. 28.15).

Fig. 28.13 Graphics methods that draw lines, rectangles and ovals. (Part 1)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

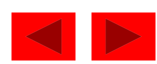


28.5 Drawing Lines, Rectangles and Ovals

Method	Description
`public void draw3DRect( int x, int y, int width, int height, boolean b )`	Draws a three-dimensional rectangle in the current color with the specified `width` and `height`. The top-left corner of the rectangle has the coordinates (`x`, `y`). The rectangle appears raised when `b` is `true` and is lowered when `b` is `false`.
`public void fill3DRect( int x, int y, int width, int height, boolean b )`	Draws a filled three-dimensional rectangle in the current color with the specified `width` and `height`. The top-left corner of the rectangle has the coordinates (`x`, `y`). The rectangle appears raised when `b` is `true` and is lowered when `b` is `false`.
`public void drawOval( int x, int y, int width, int height )`	Draws an oval in the current color with the specified `width` and `height`. The bounding rectangle's top-left corner is at the coordinates (`x`, `y`). The oval touches all four sides of the bounding rectangle at the center of each side (see Fig. 28.16).
`public void fillOval( int x, int y, int width, int height )`	Draws a filled oval in the current color with the specified `width` and `height`. The bounding rectangle's top-left corner is at the coordinates (`x`, `y`). The oval touches all four sides of the bounding rectangle at the center of each side (see Fig. 28.16).

Fig. 28.13 `Graphics` methods that draw lines, rectangles and ovals. (Part 2)

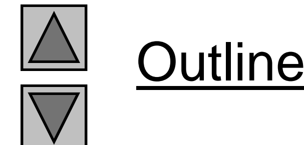
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



```java
1  // Fig. 28.14: LinesRectsOvals.java
2  // Drawing lines, rectangles and ovals
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class LinesRectsOvals extends JFrame {
8     private String s = "Using drawString!";
9
10     public LinesRectsOvals()
11     {
12        super( "Drawing lines, rectangles and ovals" );
13
14        setSize( 400, 165 );
15        show();
16     } // end LinesRectsOvals constructor
17
18     public void paint( Graphics g )
19     {
20        g.setColor( Color.red );
21        g.drawLine( 5, 30, 350, 30 );
22
23        g.setColor( Color.blue );
24        g.drawRect( 5, 40, 90, 55 );
25        g.fillRect( 100, 40, 90, 55 );
26
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

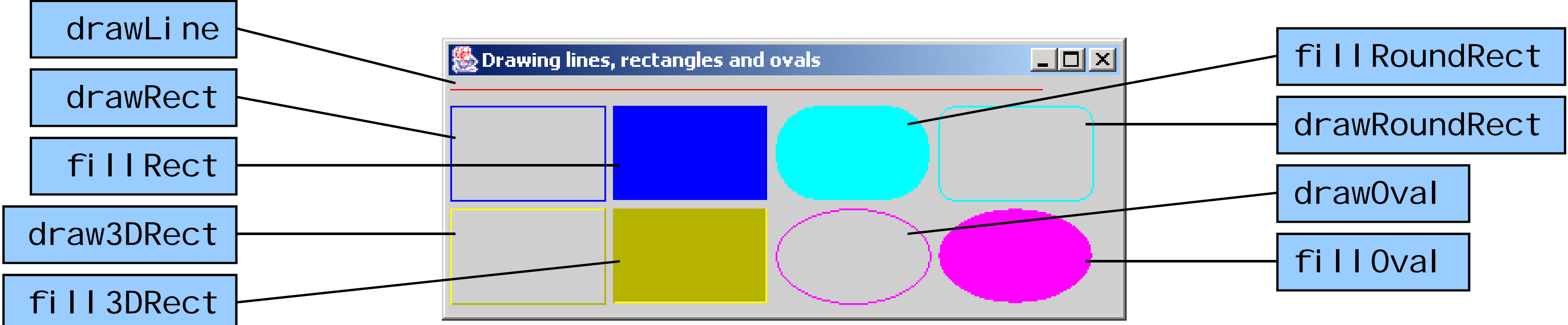
```
27       g.setColor( Color.cyan );
28       g.fillRoundRect( 195, 40, 90, 55, 50, 50 );
29       g.drawRoundRect( 290, 40, 90, 55, 20, 20 );
30
31       g.setColor( Color.yellow );
32       g.draw3DRect( 5, 100, 90, 55, true );
33       g.fill3DRect( 100, 100, 90, 55, false );
34
35       g.setColor( Color.magenta );
36       g.drawOval( 195, 100, 90, 55 );
37       g.fillOval( 290, 100, 90, 55 );
38    } // end method paint
39
40    public static void main( String args[] )
41    {
42       LinesRectsOvals app = new LinesRectsOvals();
43
44       app.addWindowListener(
45          new WindowAdapter() {
46             public void windowClosing( WindowEvent e )
47             {
48                System.exit( 0 );
49             } // end method windowClosing
50          } // end anonymous inner class
51       ); // end addWindowListener
52    } // end main
53 } // end class LinesRectsOvals
```



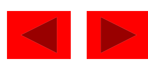
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

28.5 Drawing Lines, Rectangles and Ovals

Figure 28.14 Demonstrating `Graphics` method `drawLine`.



© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



28.5 Drawing Lines, Rectangles and Ovals

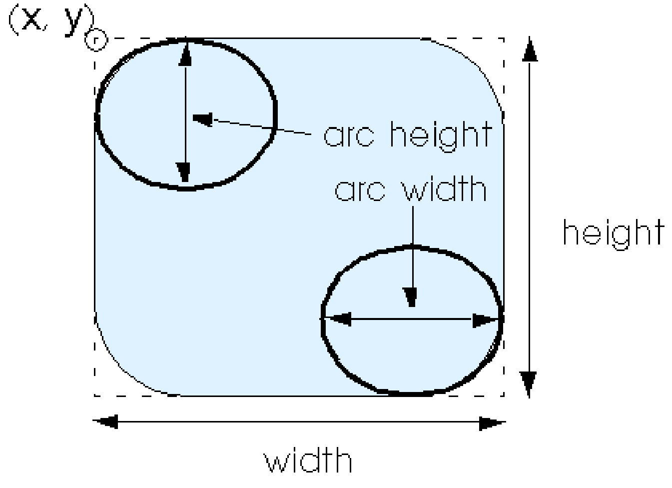
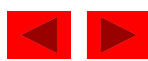


Fig. 28.15 The arc width and arc height for rounded rectangles.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



28.5 Drawing Lines, Rectangles and Ovals

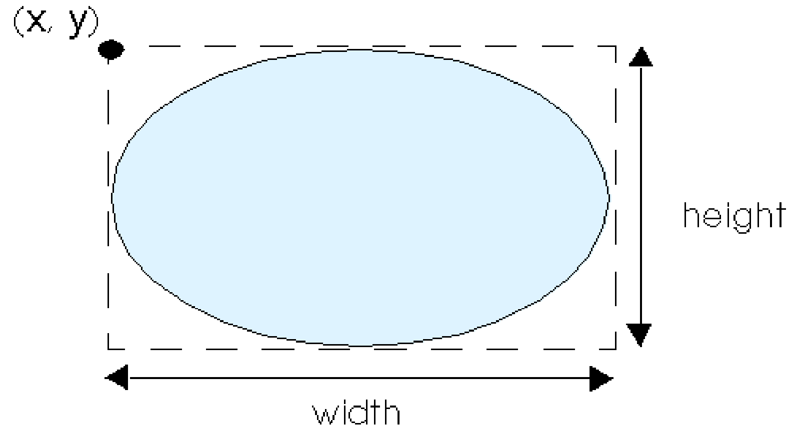


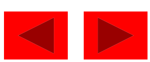
Fig. 28.16 An oval bounded by a rectangle.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

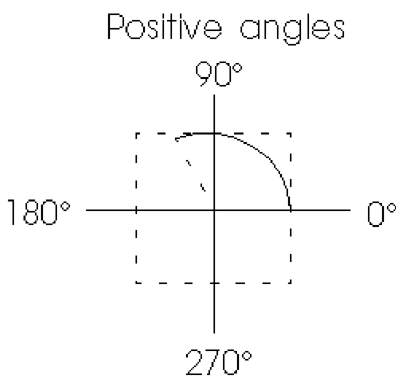
28.6 Drawing Arcs

- Arc
  - Portion of an oval
  - Arc angles measured in degrees
    - Starts at a starting angle and sweeps the number of degrees specified by arc angle
    - Positive - counterclockwise
    - Negative - clockwise
  - When drawing an arc, specify bounding rectangle for an oval
  - `drawArc( x, y, width, height, startAngle, arcAngle )`
  - `fillArc` - as above, but draws a solid arc (sector)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



28.6 Drawing Arcs



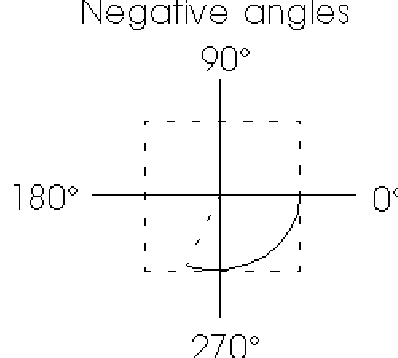
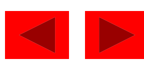


Fig. 28.17 Positive and negative arc angles.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

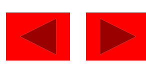


28.6 Drawing Arcs

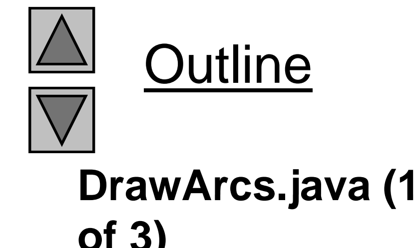
Method	Description
`public void drawArc( int x, int y, int width, int height, int startAngle, int arcAngle )`	Draws an arc relative to the bounding rectangle's top-left coordinates *(x, y)* with the specified `width` and `height`. The arc segment is drawn starting at `startAngle` and sweeps `arcAngle` degrees.
`public void fillArc( int x, int y, int width, int height, int startAngle, int arcAngle )`	Draws a solid arc (i.e., a sector) relative to the bounding rectangle's top-left coordinates *(x, y)* with the specified `width` and `height`. The arc segment is drawn starting at `startAngle` and sweeps `arcAngle` degrees.

Fig. 28.18 `Graphics` methods for drawing arcs.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



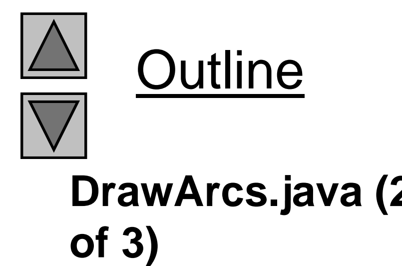
```
1  // Fig. 28.19: DrawArcs.java
2  // Drawing arcs
3  import java.awt.*;
4  import javax.swing.*;
5  import java.awt.event.*;
6
7  public class DrawArcs extends JFrame {
8     public DrawArcs()
9     {
10       super( "Drawing Arcs" );
11
12       setSize( 300, 170 );
13       show();
14    } // end DrawArcs constructor
15
16    public void paint( Graphics g )
17    {
18       // start at 0 and sweep 360 degrees
19       g.setColor( Color.yellow );
20       g.drawRect( 15, 35, 80, 80 );
21       g.setColor( Color.black );
22       g.drawArc( 15, 35, 80, 80, 0, 360 );
23
```



DrawArcs.java (1 of 3)

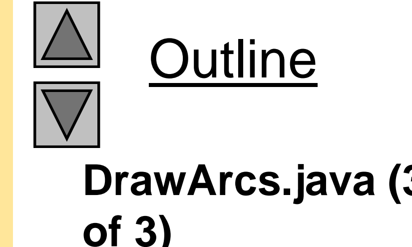
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
24       // start at 0 and sweep 110 degrees
25       g.setColor( Color.yellow );
26       g.drawRect( 100, 35, 80, 80 );
27       g.setColor( Color.black );
28       g.drawArc( 100, 35, 80, 80, 0, 110 );
29
30       // start at 0 and sweep -270 degrees
31       g.setColor( Color.yellow );
32       g.drawRect( 185, 35, 80, 80 );
33       g.setColor( Color.black );
34       g.drawArc( 185, 35, 80, 80, 0, -270 );
35
36       // start at 0 and sweep 360 degrees
37       g.fillArc( 15, 120, 80, 40, 0, 360 );
38
39       // start at 270 and sweep -90 degrees
40       g.fillArc( 100, 120, 80, 40, 270, -90 );
41
42       // start at 0 and sweep -270 degrees
43       g.fillArc( 185, 120, 80, 40, 0, -270 );
44    } // end method paint
45
46    public static void main( String args[] )
47    {
48       DrawArcs app = new DrawArcs();
49
```



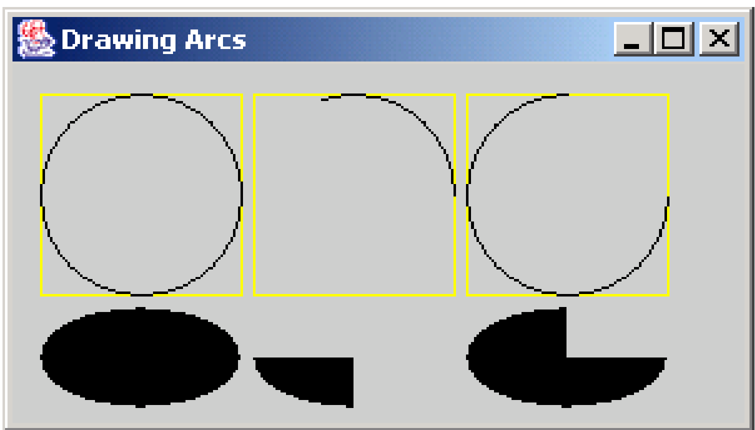
DrawArcs.java (2 of 3)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



DrawArcs.java (3 of 3)

```
50        app.addWindowListener(
51           new WindowAdapter() {
52              public void windowClosing( WindowEvent e )
53              {
54                 System.exit( 0 );
55              } // end method windowClosing
56           } // end anonymous inner class
57        ); // end addWindowListener
58     } // end main
59  } // end class DrawArcs
```

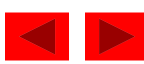


© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

28.7 Drawing Polygons and Polylines

- Polygon - multisided shape
  - In Java, class `Polygon`
    - `java.awt`
- Polyline - series of connected points

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

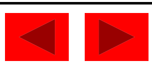


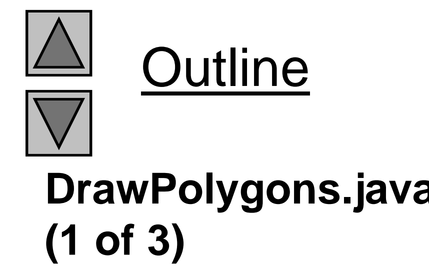
28.7 Drawing Polygons and Polylines

Method	Description
`public void drawPolygon( int xPoints[], int yPoints[], int points )`	Draws a polygon. The *x*-coordinate of each point is specified in the `xPoints` array and the *y*-coordinate of each point is specified in the `yPoints` array. The last argument specifies the number of `points`. This method draws a closed polygon—even if the last point is different from the first point.
`public void drawPolyline( int xPoints[], int yPoints[], int points )`	Draws a series of connected lines. The *x*-coordinate of each point is specified in the `xPoints` array and the *y*-coordinate of each point is specified in the `yPoints` array. The last argument specifies the number of `points`. If the last point is different from the first point, the polyline is not closed.
`public void drawPolygon( Polygon p )`	Draws the specified closed polygon.
`public void fillPolygon( int xPoints[], int yPoints[], int points )`	Draws a solid polygon. The *x*-coordinate of each point is specified in the `xPoints` array and the *y*-coordinate of each point is specified in the `yPoints` array. The last argument specifies the number of `points`. This method draws a closed polygon—even if the last point is different from the first point.
`public void fillPolygon( Polygon p )`	Draws the specified solid polygon. The polygon is closed.
`public Polygon() // Polygon class`	Constructs a new polygon object. The polygon does not contain any points.
`public Polygon( int xValues[], int yValues[], int numberOfPoints ) // Polygon class`	Constructs a new polygon object. The polygon has `numberOfPoints` sides, with each point consisting of an *x*-coordinate from `xValues` and a *y*-coordinate from `yValues`.

Fig. 28.20 `Graphics` methods for drawing polygons and class `Polygon` constructors

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



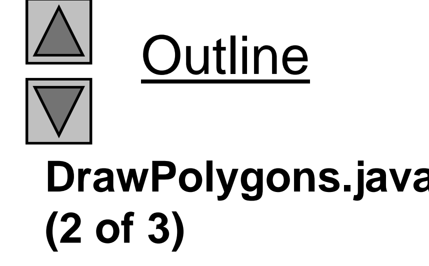


DrawPolygons.java (1 of 3)

```
1  // Fig. 28.21: DrawPolygons.java
2  // Drawing polygons
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class DrawPolygons extends JFrame {
8     public DrawPolygons()
9     {
10       super( "Drawing Polygons" );
11
12       setSize( 275, 230 );
13       show();
14    } // end DrawPolygons constructor
15
16    public void paint( Graphics g )
17    {
18       int xValues[] = { 20, 40, 50, 30, 20, 15 };
19       int yValues[] = { 50, 50, 60, 80, 80, 60 };
20       Polygon poly1 = new Polygon( xValues, yValues, 6 );
21
22       g.drawPolygon( poly1 );
23
24       int xValues2[] = { 70, 90, 100, 80, 70, 65, 60 };
25       int yValues2[] = { 100, 100, 110, 110, 130, 110, 90 };
26
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

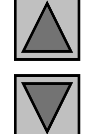
```
27        g.drawPolyline( xValues2, yValues2, 7 );
28
29        int xValues3[] = { 120, 140, 150, 190 };
30        int yValues3[] = { 40, 70, 80, 60 };
31
32        g.fillPolygon( xValues3, yValues3, 4 );
33
34        Polygon poly2 = new Polygon();
35        poly2.addPoint( 165, 135 );
36        poly2.addPoint( 175, 150 );
37        poly2.addPoint( 270, 200 );
38        poly2.addPoint( 200, 220 );
39        poly2.addPoint( 130, 180 );
40
41        g.fillPolygon( poly2 );
42     } // end method paint
43
44     public static void main( String args[] )
45     {
46        DrawPolygons app = new DrawPolygons();
47
```



DrawPolygons.java (2 of 3)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
48       app.addWindowListener(
49          new WindowAdapter() {
50             public void windowClosing( WindowEvent e )
51             {
52                System.exit( 0 );
53             } // end method windowClosing
54          } // end anonymous inner class
55       ); // end addWindowListener
56    } // end main
57 } // end class DrawPolygons
```



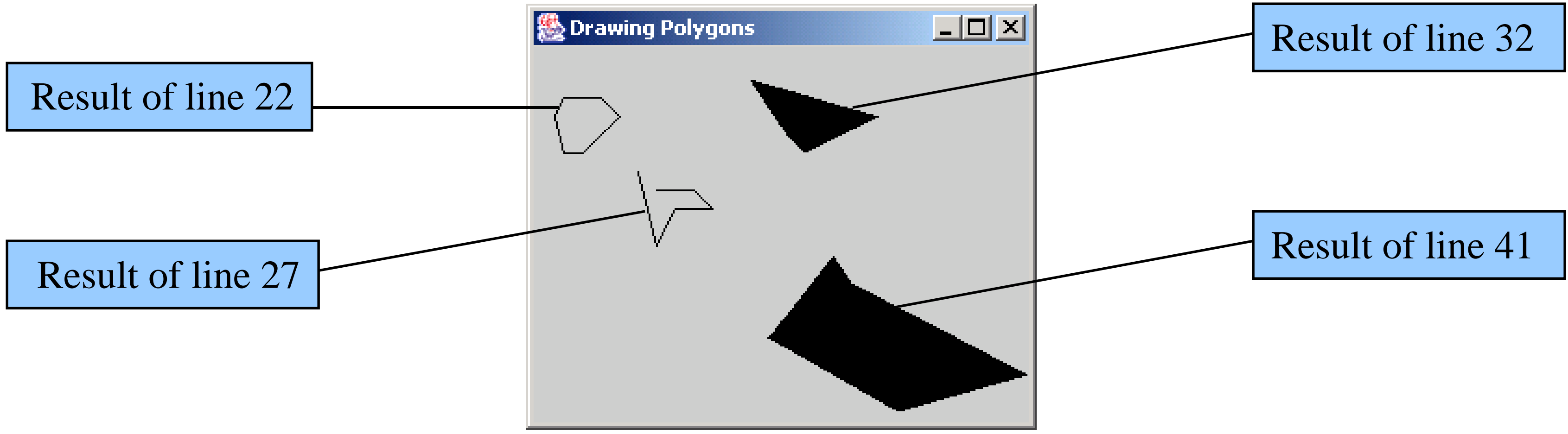
Outline

DrawPolygons.java (3 of 3)

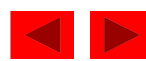
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

28.7 Drawing Polygons and Polylines

Figure 28.21 Demonstrating `drawPolygon` and `fillPolygon`.



© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



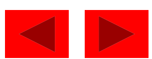
28.8 The Java2D API

- Java2D API
  - Advanced two dimensional graphics capabilities
  - Too many capabilities to cover (for overview, see demo)
- Drawing with the Java2D API
  - Use instance of class `Graphics2D` (package `java.awt`)
    - Subclass of `Graphics`
    - Has all graphics capabilities we have previously discussed
  - Must downcast `Graphics` reference passed to `paint`

```
Graphics2D g2d = ( Graphics2D ) g;
```

  - This technique used in programs of next section

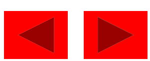
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



28.9 Java2D Shapes

- Sample methods from `Graphics2D`
  - `setPaint ( paintObject )`
    - `Paint` object is an object of a class that implements `java.awt.Paint`
    - Can be an object of class `Color`, `GradientPaint`, `SystemColor`, `TexturePaint`
  - `GradientPaint ( x1, y1, color1, x2, y2, color2, cyclic )`
    - Creates a gradient (slowly changing color) from `x1, y1`, to `x2, y2`, starting with `color1` and changing to `color2`
    - If `cyclic true`, then cyclic gradient (keeps transitioning colors)
      - If acyclic, only transitions colors once

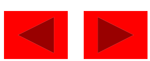
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



28.9 Java2D Shapes

- Sample methods from `Graphics2D`
  - `fill ( shapeObject )`
    - Draws a filled `Shape` object
    - Instance of any class that implements `Shape` (`java.awt`)
    - `Ellipse2D.Double`, `Rectangle2D.Double`
    - Double-precision inner classes of `Ellipse2D`
  - `setStroke( strokeObject )`
    - Set a shape's borders
    - Instance of a class that implements `Stroke` (`java.awt`)
    - `BasicStroke( width )` - One of many constructors
      - This constructor specifies width in pixels of border

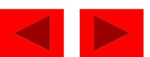
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



28.9 Java2D Shapes

- Sample methods from `Graphics2D`
  - `draw ( shapeObject )`
    - Draws specified Shape object
  - Class `BufferedImage`
    - Can produce images in color or grayscale
    - Can create patterns by drawing into the `BufferedImage` object
  - Class `TexturePaint`
    - Constructor can take `BufferedImage` and shape to fill
    - Object of class `TexturePaint` then drawn using `setPaint`
    - Book has further details

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



28.9 Java2D Shapes

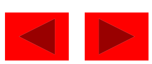
- Class `Arc2D.Double`
  - Similar to normal arcs, except has another argument at end
    - `Arc2D.PIE` - close arc with two lines
    - `Arc2D.CHORD` - draws line from endpoints of arc
    - `Arc2D.OPEN` - arc not closed
- Class `BasicStroke`
  - Can be used to create customized dashed lines, set how lines end and join

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

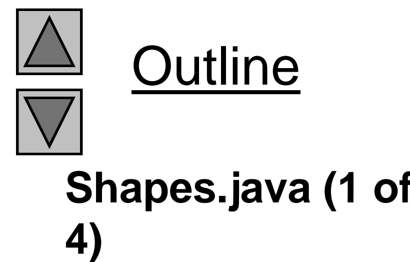
28.9 Java2D Shapes

- Class `GeneralPath`
  - A general path is a shape made from lines and curves
  - Method `moveTo`
    - Specifies first point in a general path
  - Method `lineTo`
    - Draws a line to next point in general path
  - Method `closePath`
    - Draws line from last point to point specified in last call to `moveTo`
- Other methods
  - `rotate( radians )` – rotate next shape around origin
  - `translate(x, y)` – translates origin to x, y

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



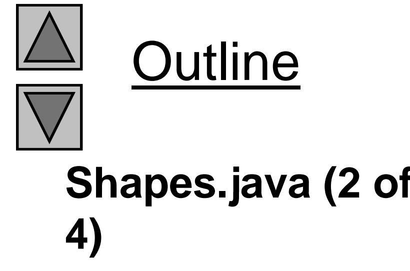
```
1  // Fig. 28.22: Shapes.java
2  // Demonstrating some Java2D shapes
3  import javax.swing.*;
4  import java.awt.event.*;
5  import java.awt.*;
6  import java.awt.geom.*;
7  import java.awt.image.*;
8
9  public class Shapes extends JFrame {
10    public Shapes()
11    {
12       super( "Drawing 2D shapes" );
13
14       setSize( 425, 160 );
15       show();
16    } // end Shapes constructor
17
18    public void paint( Graphics g )
19    {
20       // create 2D by casting g to Graphics2D
21       Graphics2D g2d = ( Graphics2D ) g;
22
```



Shapes.java (1 of 4)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

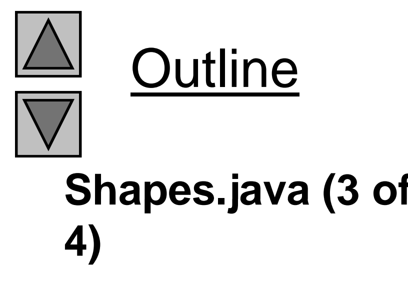
```
23        // draw 2D ellipse filled with a blue-yellow gradient
24        g2d.setPaint(
25           new GradientPaint( 5, 30,         // x1, y1
26                              Color.blue,    // initial Color
27                              35, 100,       // x2, y2
28                              Color.yellow,  // end Color
29                              true ) );      // cyclic
30        g2d.fill( new Ellipse2D.Double( 5, 30, 65, 100 ) );
31
32        // draw 2D rectangle in red
33        g2d.setPaint( Color.red );
34        g2d.setStroke( new BasicStroke( 10.0f ) );
35        g2d.draw(
36           new Rectangle2D.Double( 80, 30, 65, 100 ) );
37
38        // draw 2D rounded rectangle with a buffered background
39        BufferedImage buffImage =
40           new BufferedImage(
41              10, 10, BufferedImage.TYPE_INT_RGB );
42
```



Shapes.java (2 of 4)

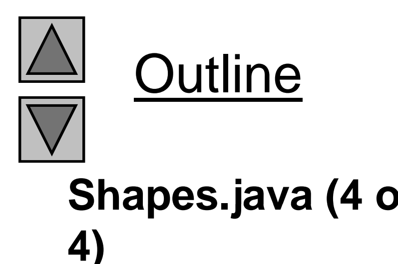
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
43        Graphics2D gg = buffImage.createGraphics();
44        gg.setColor( Color.yellow ); // draw in yellow
45        gg.fillRect( 0, 0, 10, 10 ); // draw a filled rectangle
46        gg.setColor( Color.black );  // draw in black
47        gg.drawRect( 1, 1, 6, 6 );   // draw a rectangle
48        gg.setColor( Color.blue );   // draw in blue
49        gg.fillRect( 1, 1, 3, 3 );   // draw a filled rectangle
50        gg.setColor( Color.red );    // draw in red
51        gg.fillRect( 4, 4, 3, 3 );   // draw a filled rectangle
52
53        // paint buffImage onto the JFrame
54        g2d.setPaint(
55           new TexturePaint(
56              buffImage, new Rectangle( 10, 10 ) ) );
57        g2d.fill(
58           new RoundRectangle2D.Double(
59              155, 30, 75, 100, 50, 50 ) );
60
61        // draw 2D pie-shaped arc in white
62        g2d.setPaint( Color.white );
63        g2d.setStroke( new BasicStroke( 6.0f ) );
64        g2d.draw(
65           new Arc2D.Double(
66              240, 30, 75, 100, 0, 270, Arc2D.PIE ) );
67
```



Shapes.java (3 of 4)

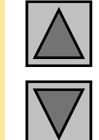
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Shapes.java (4 of 4)

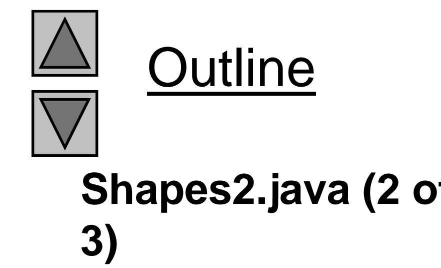
```
68       // draw 2D lines in green and yellow
69       g2d.setPaint( Color.green );
70       g2d.draw( new Line2D.Double( 395, 30, 320, 150 ) );
71
72       float dashes[] = { 10 };
73
74       g2d.setPaint( Color.yellow );
75       g2d.setStroke(
76          new BasicStroke( 4,
77                           BasicStroke.CAP_ROUND,
78                           BasicStroke.JOIN_ROUND,
79                           10, dashes, 0 ) );
80       g2d.draw( new Line2D.Double( 320, 30, 395, 150 ) );
81    } // end method paint
82
83    public static void main( String args[] )
84    {
85       Shapes app = new Shapes();
86
87       app.addWindowListener(
88          new WindowAdapter() {
89             public void windowClosing( WindowEvent e )
90             {
91                System.exit( 0 );
92             } // end method windowClosing
93          } // end anonymous inner class
94       ); // end addWindowListener
95    } // end main
96 } // end class Shapes
```

```
1  // Fig. 28.23: Shapes2.java
2  // Demonstrating a general path
3  import javax.swing.*;
4  import java.awt.event.*;
5  import java.awt.*;
6  import java.awt.geom.*;
7
8  public class Shapes2 extends JFrame {
9     public Shapes2()
10    {
11       super( "Drawing 2D Shapes" );
12
13       setBackground( Color.yellow );
14       setSize( 400, 400 );
15       show();
16    } // end Shapes2 constructor
17
18    public void paint( Graphics g )
19    {
20       int xPoints[] =
21          { 55, 67, 109, 73, 83, 55, 27, 37, 1, 43 };
22       int yPoints[] =
23          { 0, 36, 36, 54, 96, 72, 96, 54, 36, 36 };
24
25       Graphics2D g2d = ( Graphics2D ) g;
26
```



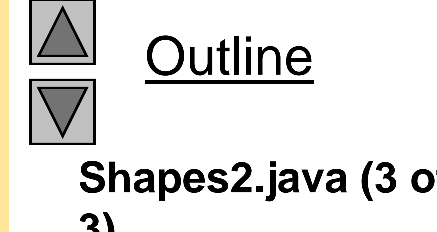
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
27        // create a star from a series of points
28        GeneralPath star = new GeneralPath();
29
30        // set the initial coordinate of the General Path
31        star.moveTo( xPoints[ 0 ], yPoints[ 0 ] );
32
33        // create the star--this does not draw the star
34        for ( int k = 1; k < xPoints.length; k++ )
35           star.lineTo( xPoints[ k ], yPoints[ k ] );
36
37        // close the shape
38        star.closePath();
39
40        // translate the origin to (200, 200)
41        g2d.translate( 200, 200 );
42
43        // rotate around origin and draw stars in random colors
44        for ( int j = 1; j <= 20; j++ ) {
45           g2d.rotate( Math.PI / 10.0 );
46           g2d.setColor(
47              new Color( ( int ) ( Math.random() * 256 ),
48                         ( int ) ( Math.random() * 256 ),
49                         ( int ) ( Math.random() * 256 ) ) );
50           g2d.fill( star );   // draw a filled star
51        } // end for
52     } // end method paint
53
```



© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
54    public static void main( String args[] )
55    {
56       Shapes2 app = new Shapes2();
57
58       app.addWindowListener(
59          new WindowAdapter() {
60             public void windowClosing( WindowEvent e )
61             {
62                System.exit( 0 );
63             } // end method windowClosing
64          } // end anonymous inner class
65       ); // end addWindowListener
66    } // end main
67 } // end class Shapes2
```



Shapes2.java (3 of 3)

Program Output

Drawing 2D Shapes

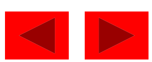
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Chapter 29 - Java Graphical User Interface Components

Outline

29.1 Introduction
29.2 Swing Overview
29.3 `JLabel`
29.4 Event Handling Model
29.5 `JTextField` and `JPasswordField`
  29.5.1 How Event Handling Works
29.6 `JTextArea`
29.7 `JButton`
29.8 `JCheckBox`
29.9 `JComboBox`
29.10 Mouse Event Handling
29.11 Layout Managers
  29.11.1 `FlowLayout`
  29.11.2 `BorderLayout`
  29.11.3 `GridLayout`
29.12 Panels
29.13 Creating a Self-Contained Subclass of `JPanel`
29.14 Windows
29.15 Using Menus with Frames

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Objectives

- In this chapter, you will learn:
  - To understand the design principles of graphical user interfaces.
  - To be able to build graphical user interfaces.
  - To understand the packages containing graphical user interface components and event handling classes and interfaces.
  - To be able to create and manipulate buttons, labels, lists, text fields and panels.
  - To understand mouse events and keyboard events.
  - To understand and be able to use layout managers.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

29.1 Introduction

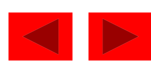
- Graphical User Interface ("Goo-ee")
  - Pictoral interface to a program
    - Distinctive "look" and "feel"
  - Different applications with consistent GUIs improve productivity
- Example: Netscape Communicator
  - Menu bar, text field, label
- GUIs built from components
  - Component: object with which user interacts
  - Examples: Labels, Text fields, Buttons, Checkboxes

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

29.1 Introduction

Button Label Menu Menu bar Text field

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

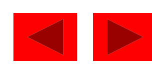


29.1 Introduction

Component	Description
JLabel	An area where uneditable text or icons can be displayed.
JTextField	An area in which the user inputs data from the keyboard. The area can also display information.
JButton	An area that triggers an event when clicked.
JCheckBox	A GUI component that is either selected or not selected.
JComboBox	A drop-down list of items from which the user can make a selection by clicking an item in the list or by typing into the box, if permitted.
JList	An area where a list of items is displayed from which the user can make a selection by clicking once on any element in the list. Double-clicking an element in the list generates an action event. Multiple elements can be selected.
JPanel	A container in which components can be placed.

Fig. 29.2 Some basic GUI components.

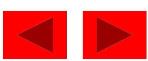
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



29.2 Swing Overview

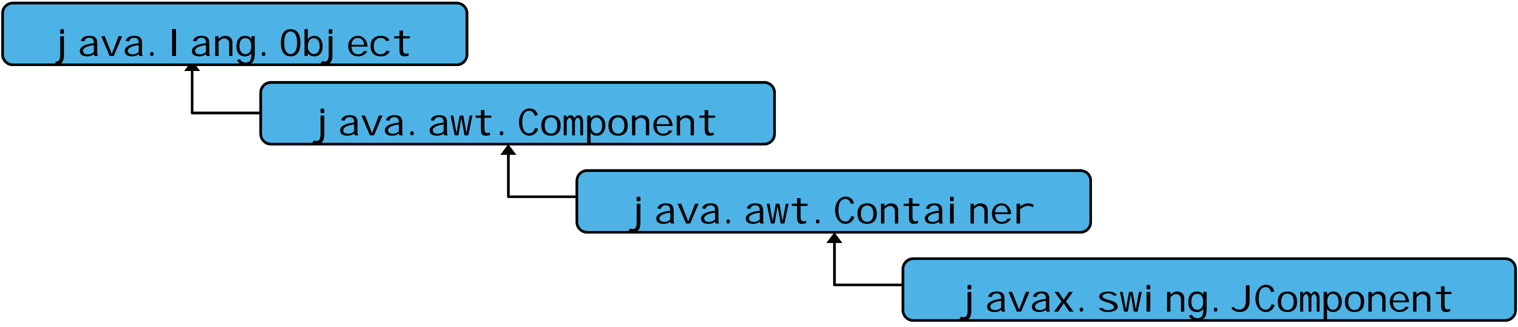
- Swing GUI components
  - Defined in package `javax.swing`
  - Original GUI components from Abstract Windowing Toolkit in `java.awt`
    - Heavyweight components - rely on local platform's windowing system for look and feel
  - Swing components are lightweight
    - Written in Java, not weighed down by complex GUI capabilities of platform
    - More portable than heavyweight components
  - Swing components allow programmer to specify look and feel
    - Can change depending on platform
    - Can be the same across all platforms

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



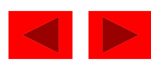
29.2 Swing Overview

- Swing component inheritance hierarchy



- `Component` defines methods that can be used in its subclasses (for example, `paint` and `repaint`)
- `Container` - collection of related components
  - When using `JFrames`, attach components to the content pane (a `Container`)
  - Method `add` to add components to content pane
- `JComponent` - superclass to most Swing components
- Much of a component's functionality inherited from these classes

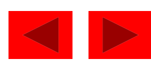
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



29.3 JLabel

- Labels
  - Provide text instructions on a GUI
  - Read-only text
  - Programs rarely change a label's contents
  - Class `JLabel` (subclass of `JComponent`)
- Methods
  - Can define label text in constructor
  - `myLabel.setToolTipText( "Text" )`
    - Displays `"Text"` in a tool tip when mouse over label
  - `myLabel.setText( "Text" )`
  - `myLabel.getText()`

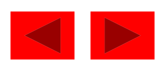
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



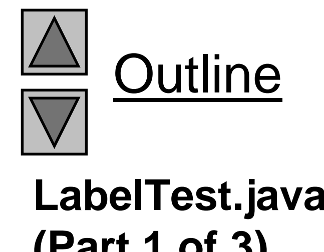
29.3 JLabel

- Icon
  - Object that implements interface Icon
  - One class is ImageIcon (.gif and .jpeg images)
  - Display an icon with setIcon method (of class JLabel)
    - `myLabel.setIcon( myIcon );`
    - `myLabel.getIcon //returns current Icon`
- Alignment
  - Set of integer constants defined in interface SwingConstants (javax.swing)
    - SwingConstants.LEFT
    - Use with JLabel methods setHorizontalTextPosition and setVerticalTextPosition

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



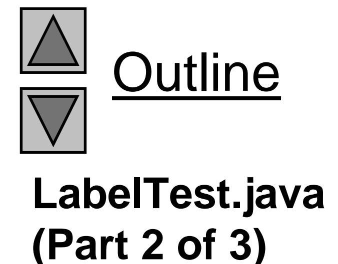
```
1  // Fig. 29.4: LabelTest.java
2  // Demonstrating the JLabel class.
3  import javax.swing.*;
4  import java.awt.*;
5  import java.awt.event.*;
6
7  public class LabelTest extends JFrame {
8     private JLabel label1, label2, label3;
9
10    public LabelTest()
11    {
12       super( "Testing JLabel" );
13
14       Container c = getContentPane();
15       c.setLayout( new FlowLayout() );
16
17       // JLabel constructor with a string argument
18       label1 = new JLabel( "Label with text" );
19       label1.setToolTipText( "This is label1" );
20       c.add( label1 );
21
```



LabelTest.java
(Part 1 of 3)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

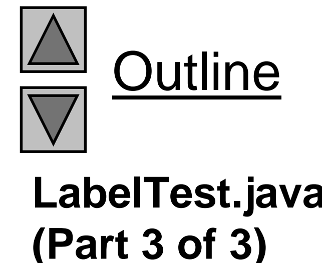
```
22        // JLabel constructor with string, Icon and
23        // alignment arguments
24        Icon bug = new ImageIcon( "bug1.gif" );
25        label2 = new JLabel( "Label with text and icon",
26                             bug, SwingConstants.LEFT );
27        label2.setToolTipText( "This is label2" );
28        c.add( label2 );
29
30        // JLabel constructor no arguments
31        label3 = new JLabel();
32        label3.setText( "Label with icon and text at bottom" );
33        label3.setIcon( bug );
34        label3.setHorizontalTextPosition(
35           SwingConstants.CENTER );
36        label3.setVerticalTextPosition(
37           SwingConstants.BOTTOM );
38        label3.setToolTipText( "This is label3" );
39        c.add( label3 );
40
41        setSize( 275, 170 );
42        show();
43     } // end LabelTest constructor
44
```



LabelTest.java (Part 2 of 3)

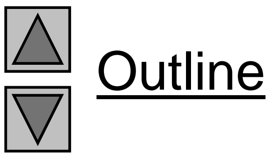
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
45    public static void main( String args[] )
46    {
47       LabelTest app = new LabelTest();
48
49       app.addWindowListener(
50          new WindowAdapter() {
51             public void windowClosing( WindowEvent e )
52             {
53                System.exit( 0 );
54             } // end method windowClosing
55          } // end anonymous inner class
56       ); // end addWindowListener
57    } // end main
58 } // end class LabelTest
```

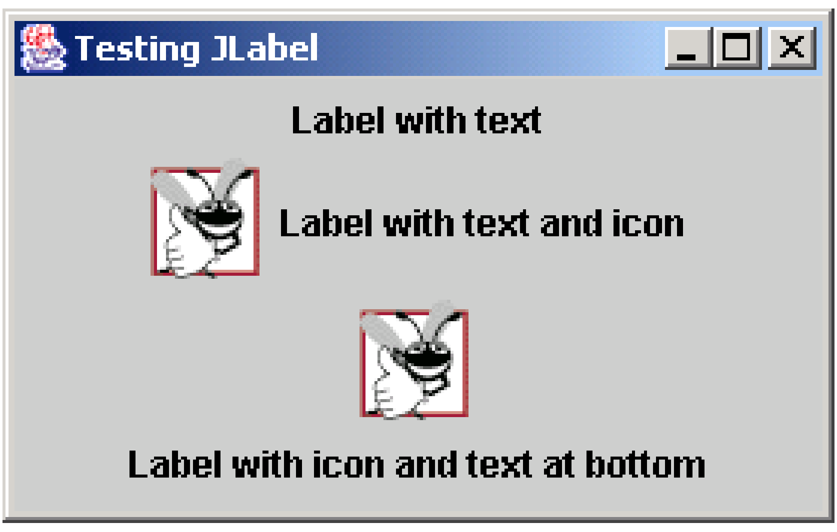


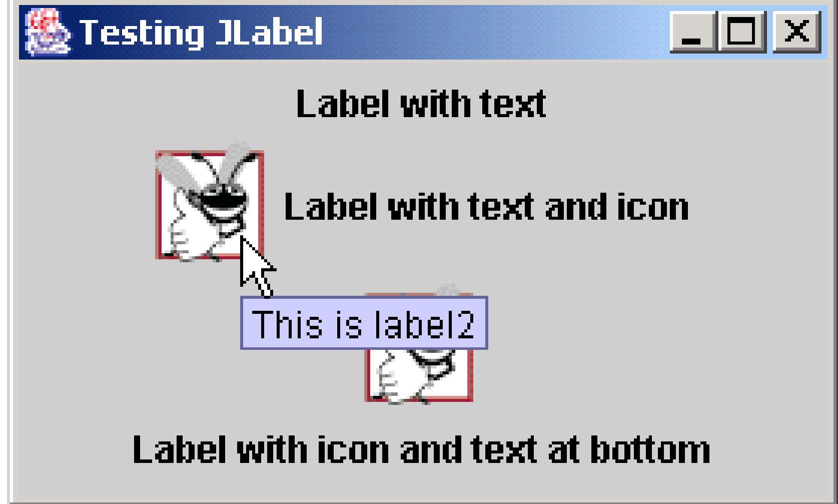
LabelTest.java
(Part 3 of 3)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Program Output



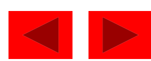


© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

29.4 Event Handling Model

- GUIs are event driven
  - Generate events when user interacts with GUI
    - Mouse movements, mouse clicks, typing in a text field, etc.
  - Event information stored in object that extends `AWTEvent`
- To process an event
  - Register an event listener
    - Object from a class that implements an event-listener interface (from `java.awt.event` or `javax.swing.event`)
    - "Listens" for events
  - Implement event handler
    - Method that is called in response to an event
    - Each event handling interface has one or more event handling methods that must be defined

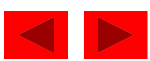
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



29.4 Event Handling Model

- Delegation event model
  - Use of event listeners in event handling
  - Processing of event delegated to particular object
- When an event occurs
  - GUI component notifies its listeners
    - Calls listener's event handling method
- Example:
  - *Enter* pressed in a `JTextField`
  - Method `actionPerformed` called for registered listener
  - Details in following section

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



29.4 Event Handling Model

java.lang.Object

java.util.EventObject

java.awt.AWTEvent

ActionEvent

AdjustmentEvent

ItemEvent

ComponentEvent

ContainerEvent

FocusEvent

PaintEvent

WindowEvent

InputEvent

KeyEvent

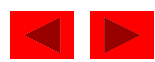
MouseEvent

Key

Class name

Interface Name

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



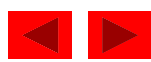
29.4 Event Handling Model

java.util.EventListener

ActionListener
AdjustmentListener
ComponentListener
ContainerListener
FocusListener
ItemListener
KeyListener
MouseListener
MouseMotionListener
TextListener
WindowListener

Key
Class name
Interface Name

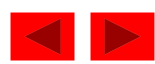
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



29.5 JTextField and JPasswordField

- JTextFields and JPasswordFields
  - Single line areas in which text can be entered or displayed
  - JPasswordFields show inputted text as *
  - JTextField extends JTextComponent
    - JPasswordField extends JTextField
- When Enter pressed
  - ActionEvent occurs
  - Currently active field "has the focus"
- Methods
  - Constructor
    - JTextField(10) - sets textfield with 10 columns of text
    - JTextField("Hi") - sets text, width determined automatically

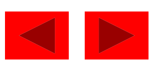
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



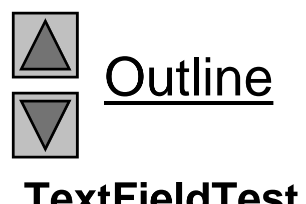
29.5 `JTextField` and `JPasswordField`

- Methods (continued)
  - `setEditable( boolean )`
    - If `true`, user can edit text
  - `getPassword`
    - Class `JPasswordField`
    - Returns password as an **`array`** of type **`char`**
- Example
  - Create `JTextField`s and a `JPasswordField`
  - Create and register an event handler
    - Displays a dialog box when *Enter* pressed

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



```
1  // Fig. 29.7: TextFieldTest.java
2  // Demonstrating the JTextField class.
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class TextFieldTest extends JFrame {
8     private JTextField text1, text2, text3;
9     private JPasswordField password;
10
11     public TextFieldTest()
12     {
13        super( "Testing JTextField and JPasswordField" );
14
15        Container c = getContentPane();
16        c.setLayout( new FlowLayout() );
17
18        // construct textfield with default sizing
19        text1 = new JTextField( 10 );
20        c.add( text1 );
21
22        // construct textfield with default text
23        text2 = new JTextField( "Enter text here" );
24        c.add( text2 );
25
```

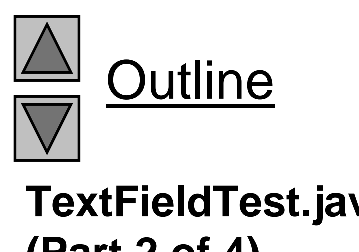


Outline

TextFieldTest.java (Part 1 of 4)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
26       // construct textfield with default text and
27       // 20 visible elements and no event handler
28       text3 = new JTextField( "Uneditable text field", 20 );
29       text3.setEditable( false );
30       c.add( text3 );
31
32       // construct textfield with default text
33       password = new JPasswordField( "Hidden text" );
34       c.add( password );
35
36       TextFieldHandler handler = new TextFieldHandler();
37       text1.addActionListener( handler );
38       text2.addActionListener( handler );
39       text3.addActionListener( handler );
40       password.addActionListener( handler );
41
42       setSize( 325, 100 );
43       show();
44    } // end TextFieldTest constructor
45
46    public static void main( String args[] )
47    {
48       TextFieldTest app = new TextFieldTest();
49
```



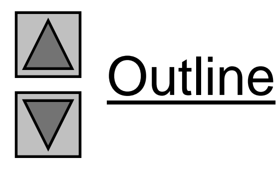
TextFieldTest.java (Part 2 of 4)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
50       app.addWindowListener(
51          new WindowAdapter() {
52             public void windowClosing( WindowEvent e )
53             {
54                System.exit( 0 );
55             } // end method windowClosing
56          } // end anonymous inner class
57       ); // end addWindowListener
58    } // end main
59
60    // inner class for event handling
61    private class TextFieldHandler implements ActionListener {
62       public void actionPerformed( ActionEvent e )
63       {
64          String s = "";
65
66          if ( e.getSource() == text1 )
67             s = "text1: " + e.getActionCommand();
68          else if ( e.getSource() == text2 )
69             s = "text2: " + e.getActionCommand();
70          else if ( e.getSource() == text3 )
71             s = "text3: " + e.getActionCommand();
```

`e.getSource()` returns a `Component` reference, which is cast to a `JPasswordField`

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

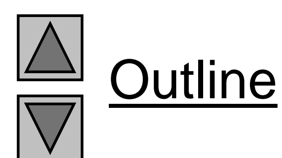


Outline

TextFieldTest.java (Part 4 of 4)

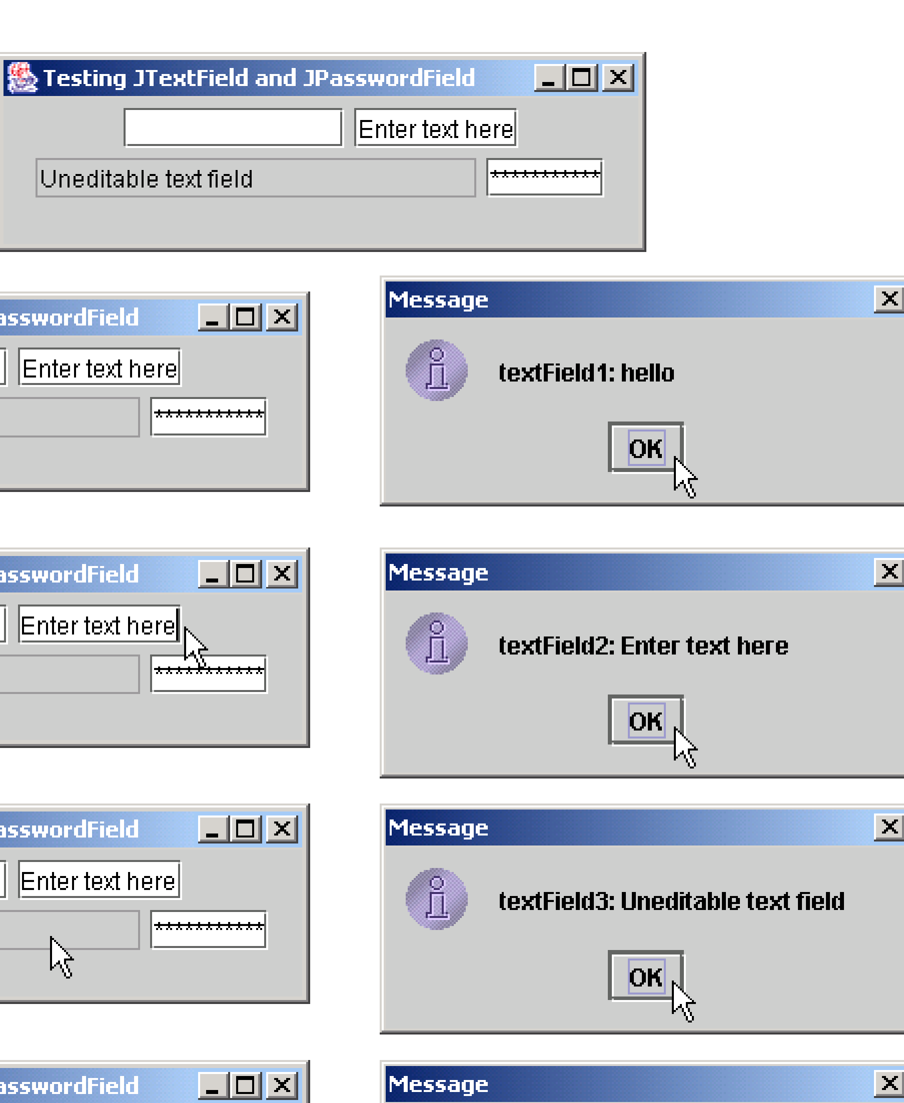
```
72          else if ( e.getSource() == password ) {
73             JPasswordField pwd =
74                (JPasswordField) e.getSource();
75             s = "password: " +
76                 new String( pwd.getPassword() );
77          } // end else if
78
79          JOptionPane.showMessageDialog( null, s );
80       } // end method actionPerformed
81    } // end class TextFieldHandler
82 } // end class TextFieldText
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Outline

Program Output



© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

29.5.1 How Event Handling Works

- Registering event listeners
  - All JComponents contain an object of class EventListenerList called listenerList
  - When text1.addActionListener(handler) executes
    - New entry placed into listenerList
- Handling events
  - When event occurs, has an event ID
    - Component uses this to decide which method to call
    - If ActionEvent, then actionPerformed called (in all registered ActionListeners)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

29.5.1 How Event Handling Works

text1

This is the JTextField object. It contains an instance variable of type EventListenerList called listenerList that it inherited from class JComponent.

listenerList

...

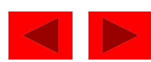
handler

This is the TextFieldHandler object that implements ActionListener and defines method actionPerformed.

```
public void
   actionPerformed(
      ActionEvent e )
{ // event handled here
}
```

This reference is created by the statement
`text1.addActionListener( handler );`

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



29.6 `JTextArea`

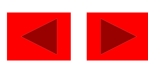
- Area for manipulating multiple lines of text
  - Like `JTextField`, inherits from `JTextComponent`
  - Many of the same methods
- `JScrollPane`
  - Provides scrolling
  - Initialize with component
    - `new JScrollPane( myComponent )`
  - Can set scrolling policies (always, as needed, never)
    - See book for details

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

29.6 JTextArea

- Box container
  - Uses BoxLayout layout manager
  - Arrange GUI components horizontally or vertically
  - `Box b = Box.createHorizontalbox();`
    - Arranges components attached to it from left to right, in order attached

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



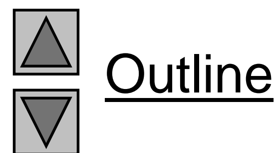
Outline

TextAreaDemo.java (Part 1 of 3)

```
1  // Fig. 29.9: TextAreaDemo.java
2  // Copying selected text from one text area to another.
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class TextAreaDemo extends JFrame {
8     private JTextArea t1, t2;
9     private JButton copy;
10
11     public TextAreaDemo()
12     {
13        super( "TextArea Demo" );
14
15        Box b = Box.createHorizontalBox();
16
17        String s = "This is a demo string to\n" +
18                   "illustrate copying text\n" +
19                   "from one TextArea to \n" +
20                   "another TextArea using an\n"+
21                   "external  event\n";
22
23        t1 = new JTextArea( s, 10, 15 );
24        b.add( new JScrollPane( t1 ) );
25
```

Initialize JScrollPane to t1 and attach to Box b

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



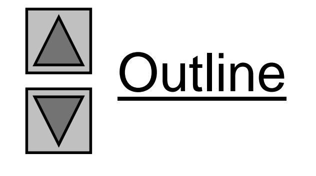
Outline

TextAreaDemo.java (Part 2 of 3)

```
26       copy = new JButton( "Copy >>>" );
27       copy.addActionListener(
28          new ActionListener() {
29             public void actionPerformed( ActionEvent e )
30             {
31                t2.setText( t1.getSelectedText() );
32             } // end method actionPerformed
33          } // end anonymous inner class
34       ); // end addActionListener
35       b.add( copy );
36
37       t2 = new JTextArea( 10, 15 );
38       t2.setEditable( false );
39       b.add( new JScrollPane( t2 ) );
40
41       Container c = getContentPane();
42       c.add( b );
43       setSize( 425, 200 );
44       show();
45    } // end TextAreaDemo constructor
46
```

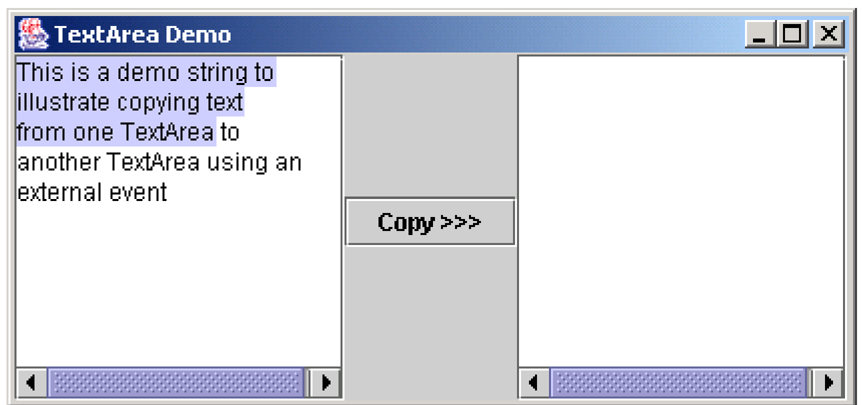
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

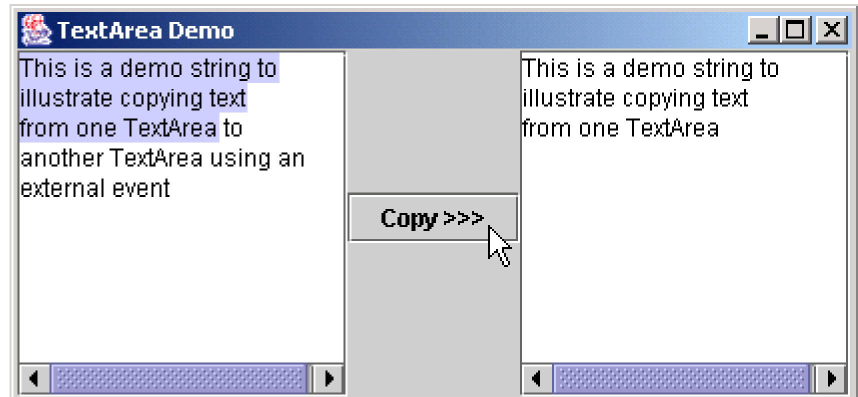
```
47    public static void main( String args[] )
48    {
49       TextAreaDemo app = new TextAreaDemo();
50
51       app.addWindowListener(
52          new WindowAdapter() {
53             public void windowClosing( WindowEvent e )
54             {
55                System.exit( 0 );
56             } // end method windowClosing
57          } // end anonymous inner class
58       ); // end addWindowListener
59    } // end main
60 } // end class TextAreaDemo
```



TextAreaDemo.java (Part 3 of 3)

Program Output



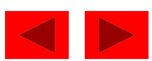


© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

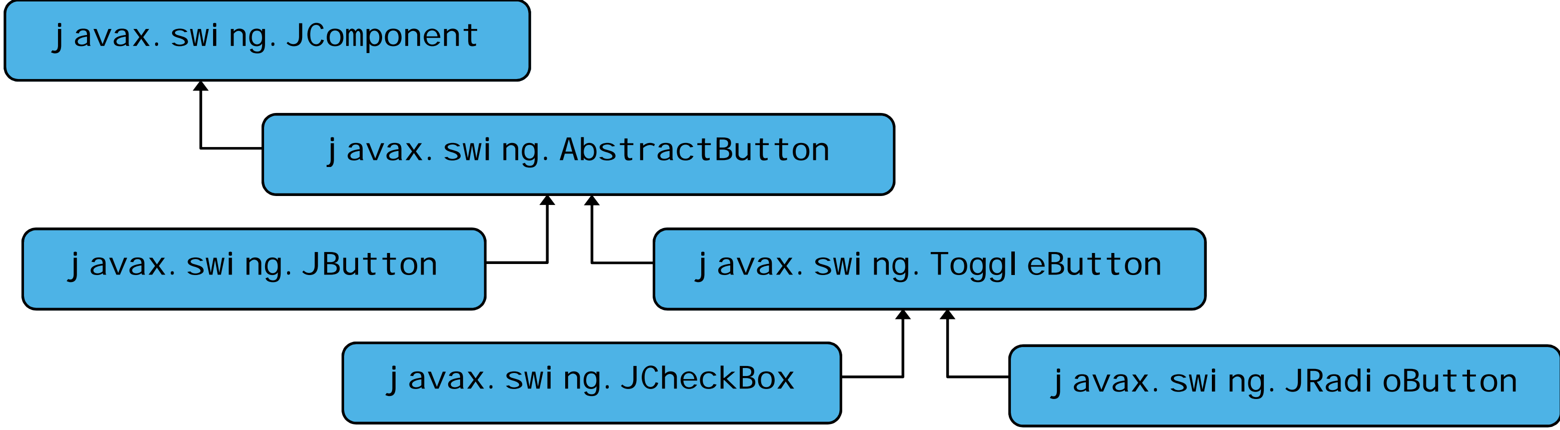
29.7 `JButton`

- Button
  - Component user clicks to trigger an action
  - Several types of buttons
    - Command buttons, toggle buttons, check boxes, radio buttons
- Command button
  - Generates `ActionEvent` when clicked
  - Created with class `JButton`
    - Inherits from class `AbstractButton`
- `Jbutton`
  - Text on face called button label
  - Each button should have a different label
  - Support display of `Icons`

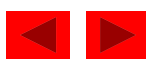
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



29.7 JButton



© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



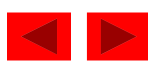
29.7 JButton

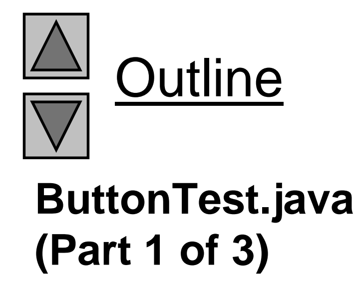
- Constructors

```
Jbutton myButton = new JButton( "Button" );
Jbutton myButton = new JButton( "Button", myIcon );
```

- Method
  - `setRolloverIcon( myIcon )`
    - Sets image to display when mouse over button

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



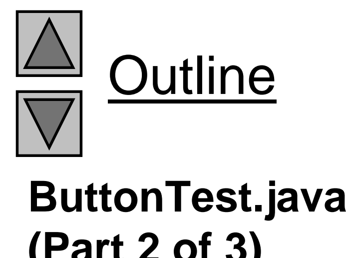


ButtonTest.java (Part 1 of 3)

```
1  // Fig. 29.11: ButtonTest.java
2  // Creating JButtons.
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class ButtonTest extends JFrame {
8     private JButton plainButton, fancyButton;
9
10    public ButtonTest()
11    {
12       super( "Testing Buttons" );
13
14       Container c = getContentPane();
15       c.setLayout( new FlowLayout() );
16
17       // create buttons
18       plainButton = new JButton( "Plain Button" );
19       c.add( plainButton );
20
21       Icon bug1 = new ImageIcon( "bug1.gif" );
22       Icon bug2 = new ImageIcon( "bug2.gif" );
23       fancyButton = new JButton( "Fancy Button", bug1 );
24       fancyButton.setRolloverIcon( bug2 );
25       c.add( fancyButton );
26
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

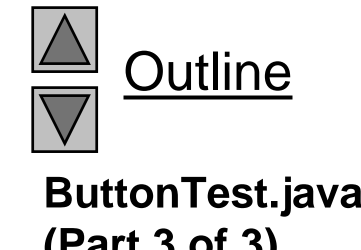
```
27       // create an instance of inner class ButtonHandler
28       // to use for button event handling
29       ButtonHandler handler = new ButtonHandler();
30       fancyButton.addActionListener( handler );
31       plainButton.addActionListener( handler );
32
33       setSize( 275, 100 );
34       show();
35    } // end ButtonTest constructor
36
37    public static void main( String args[] )
38    {
39       ButtonTest app = new ButtonTest();
40
41       app.addWindowListener(
42          new WindowAdapter() {
43             public void windowClosing( WindowEvent e )
44             {
45                System.exit( 0 );
46             } // end method windowClosing
47          } // end anonymous inner class
48       ); // end addWindowListener
49    } // end main
50
```



ButtonTest.java (Part 2 of 3)

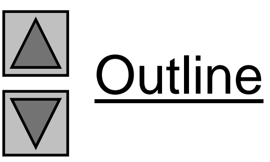
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
51    // inner class for button event handling
52    private class ButtonHandler implements ActionListener {
53       public void actionPerformed( ActionEvent e )
54       {
55          JOptionPane.showMessageDialog( null,
56             "You pressed: " + e.getActionCommand() );
57       } // end method actionPerformed
58    } // end class ButtonHandler
59 } // end class ButtonTest
```

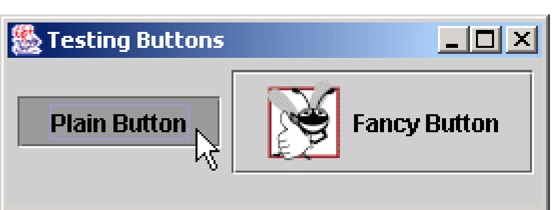


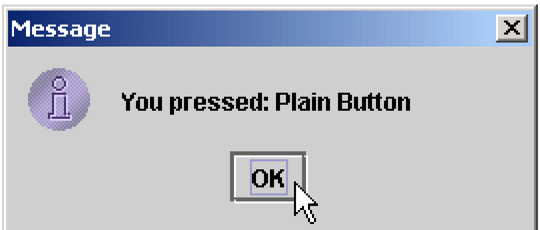
ButtonTest.java (Part 3 of 3)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Program Output

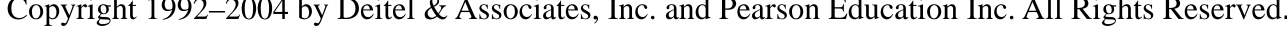


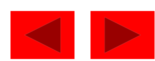


© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

29.8 JCheckBox

- State buttons
  - JToggleButton
    - Subclasses JCheckBox, JRadioButton
  - Have on/off (true/false) values
  - We discuss JCheckBox in this section
- Initialization
  - `JCheckBox myBox = new JCheckBox( "Title" );`
- When JCheckBox changes
  - ItemEvent generated
    - Handled by an ItemListener, which must define itemStateChanged
  - Register with addItemListener

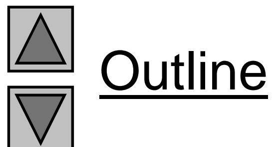

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



29.8 JCheckBox

- ItemEvent methods
  - getStateChange
    - Returns ItemEvent.SELECTED or ItemEvent.DESELECTED

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

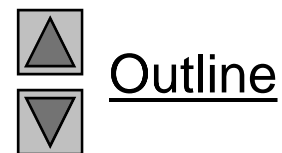


CheckBoxTest.java (Part 1 of 3)

```
1  // Fig. 29.12: CheckBoxTest.java
2  // Creating Checkbox buttons.
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class CheckBoxTest extends JFrame {
8     private JTextField t;
9     private JCheckBox bold, italic;
10
11     public CheckBoxTest()
12     {
13        super( "JCheckBox Test" );
14
15        Container c = getContentPane();
16        c.setLayout(new FlowLayout());
17
18        t = new JTextField( "Watch the font style change", 20 );
19        t.setFont( new Font( "TimesRoman", Font.PLAIN, 14 ) );
20        c.add( t );
21
22        // create checkbox objects
23        bold = new JCheckBox( "Bold" );
24        c.add( bold );
25
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
26       italic = new JCheckBox( "Italic" );
27       c.add( italic );
28
29       CheckBoxHandler handler = new CheckBoxHandler();
30       bold.addItemListener( handler );
31       italic.addItemListener( handler );
32
33       addWindowListener(
34          new WindowAdapter() {
35             public void windowClosing( WindowEvent e )
36             {
37                System.exit( 0 );
38             } // end method windowClosing
39          } // end anonymous inner class
40       ); // end addWindowListener
41
42       setSize( 275, 100 );
43       show();
44    } // end CheckBoxTest constructor
45
46    public static void main( String args[] )
47    {
48       new CheckBoxTest();
49    }
50
```



CheckBoxTest.java (Part 2 of 3)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

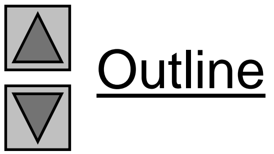
```
51    private class CheckBoxHandler implements ItemListener {
52       private int valBold = Font.PLAIN;
53       private int valItalic = Font.PLAIN;
54
55       public void itemStateChanged( ItemEvent e )
56       {
57          if ( e.getSource() == bold )
58             if ( e.getStateChange() == ItemEvent.SELECTED )
59                valBold = Font.BOLD;
60             else
61                valBold = Font.PLAIN;
62
63          if ( e.getSource() == italic )
64             if ( e.getStateChange() == ItemEvent.SELECTED )
65                valItalic = Font.ITALIC;
66             else
67                valItalic = Font.PLAIN;
68
69          t.setFont(
70             new Font( "TimesRoman", valBold + valItalic, 14 ) );
71          t.repaint();
72       } // end method itemStateChanged
73    } // end inner class CheckBoxHandler
74 } // end class CheckBoxTest
```

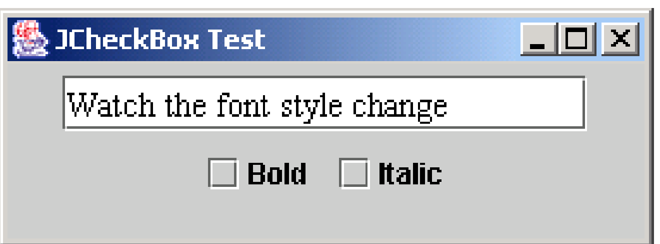
Because **CheckBoxHandler** implements **ItemListener**, it must define method **itemStateChanged**

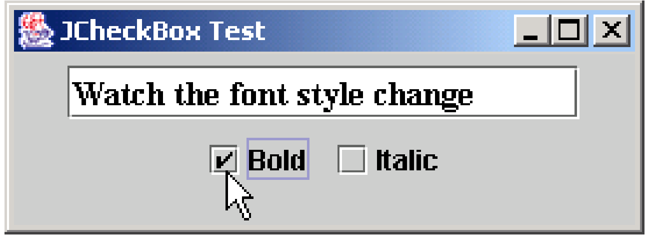
Outline

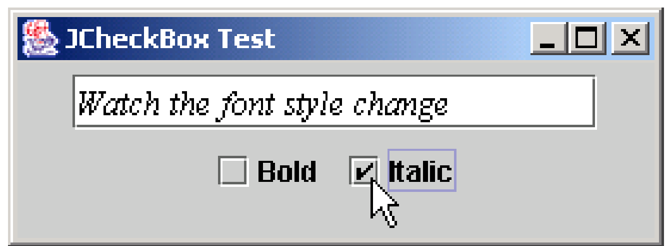
CheckBoxTest.java (Part 3 of 3)

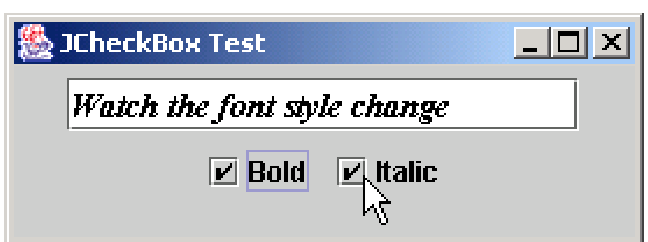
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.









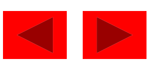


© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

29.9 JComboBox

- Combo box (drop down list)
  - List of items, user makes a selection
  - Class JComboBox
    - Generate ItemEvents
- JComboBox
  - Numeric index keeps track of elements
    - First element added at index 0
    - First item added is appears as currently selected item when combo box appears

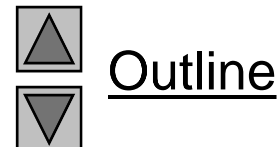
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



29.9 JComboBox

- Methods
  - `getSelectedIndex`
    - Returns the index of the currently selected item
  - `setMaximumRowCount( n )`
    - Set the maximum number of elements to display when user clicks combo box
    - Scrollbar automatically provided

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

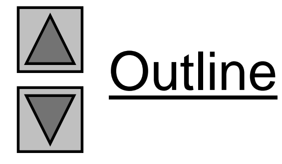


Outline

ComboBoxTest.java (Part 1 of 3)

```
1  // Fig. 29.13: ComboBoxTest.java
2  // Using a JComboBox to select an image to display.
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class ComboBoxTest extends JFrame {
8     private JComboBox images;
9     private JLabel label;
10    private String names[] =
11       { "bug1.gif", "bug2.gif",
12         "travelbug.gif", "buganim.gif" };
13    private Icon icons[] =
14       { new ImageIcon( names[ 0 ] ),
15         new ImageIcon( names[ 1 ] ),
16         new ImageIcon( names[ 2 ] ),
17         new ImageIcon( names[ 3 ] ) };
18
19    public ComboBoxTest()
20    {
21       super( "Testing JComboBox" );
22
23       Container c = getContentPane();
24       c.setLayout( new FlowLayout() );
25
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

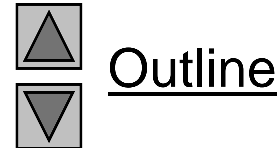


ComboBoxTest.java (Part 2 of 3)

```
26       images = new JComboBox( names );
27       images.setMaximumRowCount( 3 );
28
29       images.addItemListener(
30          new ItemListener() {
31             public void itemStateChanged( ItemEvent e )
32             {
33                label.setIcon(
34                   icons[ images.getSelectedIndex() ] );
35             } // end method itemStateChanged
36          } // end anonymous inner class
37       ); // end addItemListener
38
39       c.add( images );
40
41       label  = new JLabel( icons[ 0 ] );
42       c.add( label  );
43
44       setSize( 350, 100 );
45       show();
46    } // end ComboBoxText constructor
47
48    public static void main( String args[] )
49    {
50       ComboBoxTest app = new ComboBoxTest();
51
```

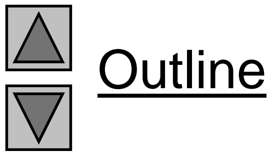
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
52       app.addWindowListener(
53          new WindowAdapter() {
54             public void windowClosing( WindowEvent e )
55             {
56                System.exit( 0 );
57             } // end method windowClosing
58          } // end anonymous inner class
59       ); // end addWindowListener
60    } // end main
61 } // end class ComboBoxTest
```

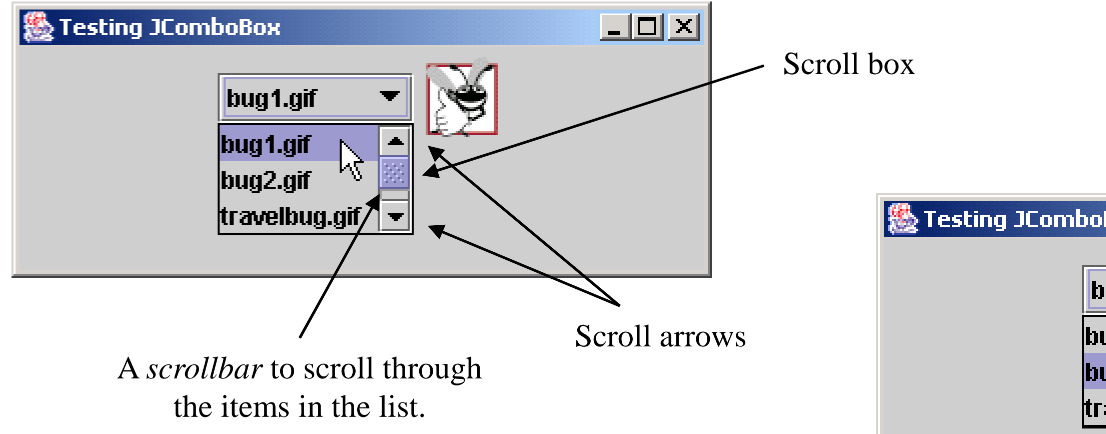


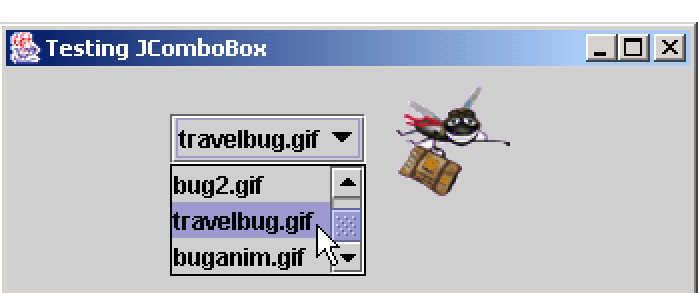
ComboBoxTest.java (Part 3 of 3)

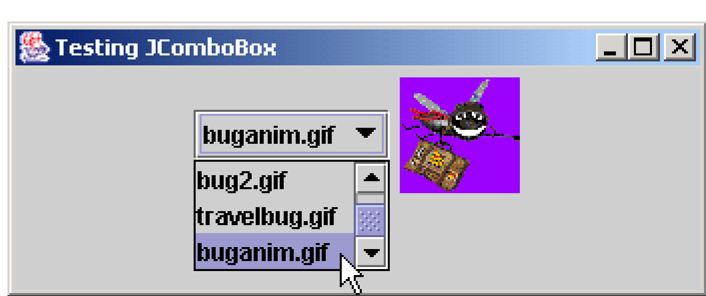
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Program Output







© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

29.10 Mouse Event Handling

- Mouse events
  - Can be trapped for any GUI component derived from `java.awt.Component`
  - Mouse event handling methods
    - Take a `MouseEvent` object
      - Contains info about event, including `x` and `y` coordinates
      - Methods `getX` and `getY`
  - `MouseListener` and `MouseMotionListener` methods called automatically (if component is registered)
    - `addMouseListener`
    - `addMouseMotionListener`

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

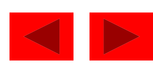
29.10 Mouse Event Handling

- Interface methods for MouseListener and MouseMotionListener

MouseListener and MouseMotionListener interface methods
`public void mousePressed( MouseEvent e )  // MouseListener`
Called when a mouse button is pressed with the mouse cursor on a component.
`public void mouseClicked( MouseEvent e )  // MouseListener`
Called when a mouse button is pressed and released on a component without moving the mouse cursor.
`public void mouseReleased( MouseEvent e ) // MouseListener`
Called when a mouse button is released after being pressed. This event is always preceded by a mousePressed event.
`public void mouseEntered( MouseEvent e )  // MouseListener`
Called when the mouse cursor enters the bounds of a component.
`public void mouseExited( MouseEvent e )   // MouseListener`
Called when the mouse cursor leaves the bounds of a component.
`public void mouseDragged( MouseEvent e )  // MouseMotionListener`
Called when the mouse button is pressed and the mouse is moved. This event is always preceded by a call to mousePressed.
`public void mouseMoved( MouseEvent e )    // MouseMotionListener`
Called when the mouse is moved with the mouse cursor on a component.

Fig. 29.14 MouseListener and MouseMotionListener interface methods.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

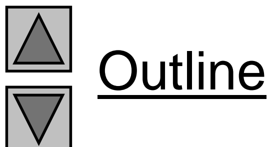


MouseTracker.java (Part 1 of 4)

```
1  // Fig. 29.15: MouseTracker.java
2  // Demonstrating mouse events.
3
4  import java.awt.*;
5  import java.awt.event.*;
6  import javax.swing.*;
7
8  public class MouseTracker extends JFrame
9              implements MouseListener, MouseMotionListener {
10    private JLabel statusBar;
11
12    public MouseTracker()
13    {
14       super( "Demonstrating Mouse Events" );
15
16       statusBar = new JLabel();
17       getContentPane().add( statusBar, BorderLayout.SOUTH );
18
19       // application listens to its own mouse events
20       addMouseListener( this );
21       addMouseMotionListener( this );
22
23       setSize( 275, 100 );
24       show();
25    } // end MouseTracker constructor
26
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

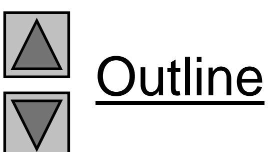
```
27    // MouseListener event handlers
28    public void mouseClicked( MouseEvent e )
29    {
30       statusBar.setText( "Clicked at [" + e.getX() +
31                          ", " + e.getY() + "]" );
32    } // end method mouseClicked
33
34    public void mousePressed( MouseEvent e )
35    {
36       statusBar.setText( "Pressed at [" + e.getX() +
37                          ", " + e.getY() + "]" );
38    } // end method mousePressed
39
40    public void mouseReleased( MouseEvent e )
41    {
42       statusBar.setText( "Released at [" + e.getX() +
43                          ", " + e.getY() + "]" );
44    } // end method mouseReleased
45
46    public void mouseEntered( MouseEvent e )
47    {
48       statusBar.setText( "Mouse in window" );
49    } // end method mouseEntered
50
```



MouseTracker.java (Part 2 of 4)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

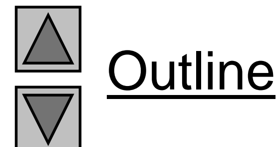
```
51    public void mouseExited( MouseEvent e )
52    {
53       statusBar.setText( "Mouse outside window" );
54    } // end method mouseExited
55
56    // MouseMotionListener event handlers
57    public void mouseDragged( MouseEvent e )
58    {
59       statusBar.setText( "Dragged at [" + e.getX() +
60                          ", " + e.getY() + "]" );
61    } // end method mouseDragged
62
63    public void mouseMoved( MouseEvent e )
64    {
65       statusBar.setText( "Moved at [" + e.getX() +
66                          ", " + e.getY() + "]" );
67    } // end method mouseMoved
68
69    public static void main( String args[] )
70    {
71       MouseTracker app = new MouseTracker();
72
```



Outline

MouseTracker.java (Part 3 of 4)

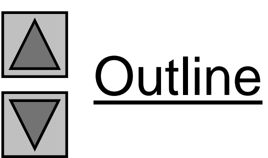
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



MouseTracker.java (Part 4 of 4)

```
73       app.addWindowListener(
74          new WindowAdapter() {
75             public void windowClosing( WindowEvent e )
76             {
77                System.exit( 0 );
78             } // end method windowClosing
79          } // end anonymous inner class
80       ); // end addWindowListener
81    } // end main
82 } // end class MouseTracker
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Program Output

Demonstrating Mouse Events

Mouse outside window

Demonstrating Mouse Events

Mouse in window

Demonstrating Mouse Events

Moved at [69, 31]

Demonstrating Mouse Events

Pressed at [127, 47]

Demonstrating Mouse Events

Dragged at [171, 43]

Demonstrating Mouse Events

Released at [158, 47]

Demonstrating Mouse Events

Clicked at [4, 26]

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

29.11 Layout Managers

- Layout managers
  - Arrange GUI components on a container
  - Provide basic layout capabilities
    - Easier to use than determining exact size and position of every component
    - Programmer concentrates on "look and feel" rather than details

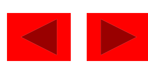
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

29.11 Layout Managers

Layout manager	Description
`FlowLayout`	Default for `java.awt.Applet`, `java.awt.Panel` and `javax.swing.JPanel`. Places components sequentially (left to right) in the order they were added. It is also possible to specify the order of the components using the `Container` method `add` that takes a `Component` and an integer index position as arguments.
`BorderLayout`	Default for the content panes of `JFrames` (and other windows) and `JApplets`. Arranges the components into five areas: North, South, East, West and Center.
`GridLayout`	Arranges the components into rows and columns.

Fig. 29.16 Layout managers.

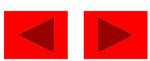
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



29.11.1 `FlowLayout`

- Most basic layout manager
  - Components placed left to right in order added
  - When edge of container reached, continues on next line
  - Components can be left-aligned, centered (default), or right-aligned
- Method
  - `setAlignment`
    - `FlowLayout.LEFT`, `FlowLayout.CENTER`, `FlowLayout.RIGHT`
  - `layoutContainer( Container )`
    - Update `Container` specified with layout

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

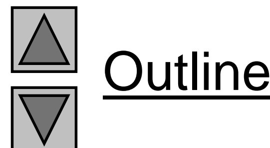


FlowLayout-Demo.java (1 of 4)

```
1  // Fig. 29.17: FlowLayoutDemo.java
2  // Demonstrating FlowLayout alignments.
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class FlowLayoutDemo extends JFrame {
8     private JButton left, center, right;
9     private Container c;
10     private FlowLayout layout;
11
12     public FlowLayoutDemo()
13     {
14        super( "FlowLayout Demo" );
15
16        layout = new FlowLayout();
17
18        c = getContentPane();
19        c.setLayout( layout );
20
21        left = new JButton( "Left" );
22        left.addActionListener(
23           new ActionListener() {
24              public void actionPerformed( ActionEvent e )
25              {
26                 layout.setAlignment( FlowLayout.LEFT );
27
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

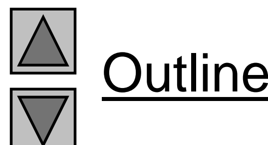
```
28                  // re-align attached components
29                  layout.layoutContainer( c );
30               } // end method actionPerformed
31            } // end anonymous inner class
32         ); // end addActionListener
33         c.add( left );
34
35         center = new JButton( "Center" );
36         center.addActionListener(
37            new ActionListener() {
38               public void actionPerformed( ActionEvent e )
39               {
40                  layout.setAlignment( FlowLayout.CENTER );
41
42                  // re-align attached components
43                  layout.layoutContainer( c );
44               } // end method actionPerformed
45            } // end anonymous inner class
46         ); // end addActionListener
47         c.add( center );
48
```



FlowLayout-Demo.java
(2 of 4)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

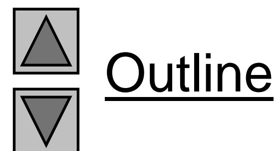
```
49       right = new JButton( "Right" );
50       right.addActionListener(
51          new ActionListener() {
52             public void actionPerformed( ActionEvent e )
53             {
54                layout.setAlignment( FlowLayout.RIGHT );
55
56                // re-align attached components
57                layout.layoutContainer( c );
58             } // end method actionPerformed
59          } // end anonymous inner class
60       ); // end addActionListener
61       c.add( right );
62
63       setSize( 300, 75 );
64       show();
65    } // end FlowLayoutDemo constructor
66
```



FlowLayout-Demo.java
(3 of 4)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

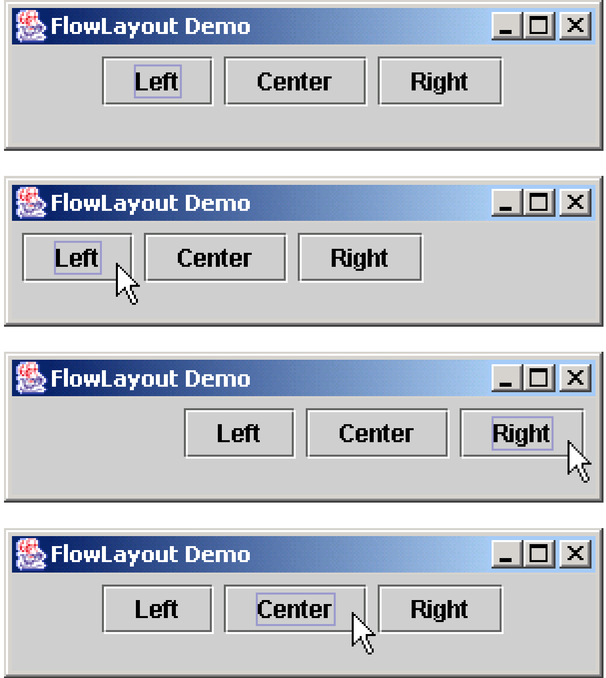
```
67    public static void main( String args[] )
68    {
69       FlowLayoutDemo app = new FlowLayoutDemo();
70
71       app.addWindowListener(
72          new WindowAdapter() {
73             public void windowClosing( WindowEvent e )
74             {
75                System.exit( 0 );
76             } // end method windowClosing
77          } // end anonymous inner class
78       ); // end addWindowListener
79    } // end main
80 } // end class FlowLayoutDemo
```



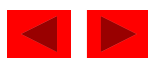
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

29.11.1 FlowLayout

Fig. 29.17 Program that demonstrates components in FlowLayout.



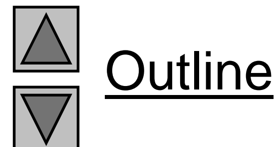
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



29.11.2 `BorderLayout`

- `BorderLayout`
  - Default manager for content pane
  - Arrange components into 5 regions
    - North, south, east, west, center
  - Up to 5 components can be added directly
    - One for each region
  - Components placed in
    - North/South - Region is as tall as component
    - East/West - Region is as wide as component
    - Center - Region expands to take all remaining space

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

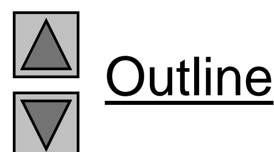


Outline

BorderLayout-Demo.java (1 of 3)

```
1  // Fig. 29.18: BorderLayoutDemo.java
2  // Demonstrating BorderLayout.
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class BorderLayoutDemo extends JFrame
8                               implements ActionListener {
9     private JButton b[];
10    private String names[] =
11       { "Hide North", "Hide South", "Hide East",
12         "Hide West", "Hide Center" };
13    private BorderLayout layout;
14
15    public BorderLayoutDemo()
16    {
17       super( "BorderLayout Demo" );
18
19       layout = new BorderLayout( 5, 5 );
20
21       Container c = getContentPane();
22       c.setLayout( layout );
23
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



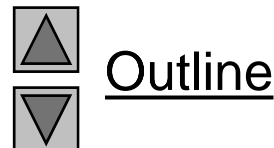
Outline

BorderLayout-Demo.java
(2 of 3)

```
24      // instantiate button objects
25      b = new JButton[ names.length ];
26
27      for ( int i = 0; i < names.length; i++ ) {
28         b[ i ] = new JButton( names[ i ] );
29         b[ i ].addActionListener( this );
30      } // end for
31
32      // order not important
33      c.add( b[ 0 ], BorderLayout.NORTH );  // North position
34      c.add( b[ 1 ], BorderLayout.SOUTH );  // South position
35      c.add( b[ 2 ], BorderLayout.EAST );   // East position
36      c.add( b[ 3 ], BorderLayout.WEST );   // West position
37      c.add( b[ 4 ], BorderLayout.CENTER ); // Center position
38
39      setSize( 300, 200 );
40      show();
41   } // end BorderLayoutDemo constructor
42
43   public void actionPerformed( ActionEvent e )
44   {
45      for ( int i = 0; i < b.length; i++ )
46         if ( e.getSource() == b[ i ] )
47            b[ i ].setVisible( false );
48         else
49            b[ i ].setVisible( true );
50
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
51       // re-layout the content pane
52       layout.layoutContainer( getContentPane() );
53    } // end method actionPerformed
54
55    public static void main( String args[] )
56    {
57       BorderLayoutDemo app = new BorderLayoutDemo();
58
59       app.addWindowListener(
60          new WindowAdapter() {
61             public void windowClosing( WindowEvent e )
62             {
63                System.exit( 0 );
64             } // end method windowClosing
65          } // end anonymous inner class
66       ); // end addWindowListener
67    } // end main
68 } // end class BorderLayoutDemo
```

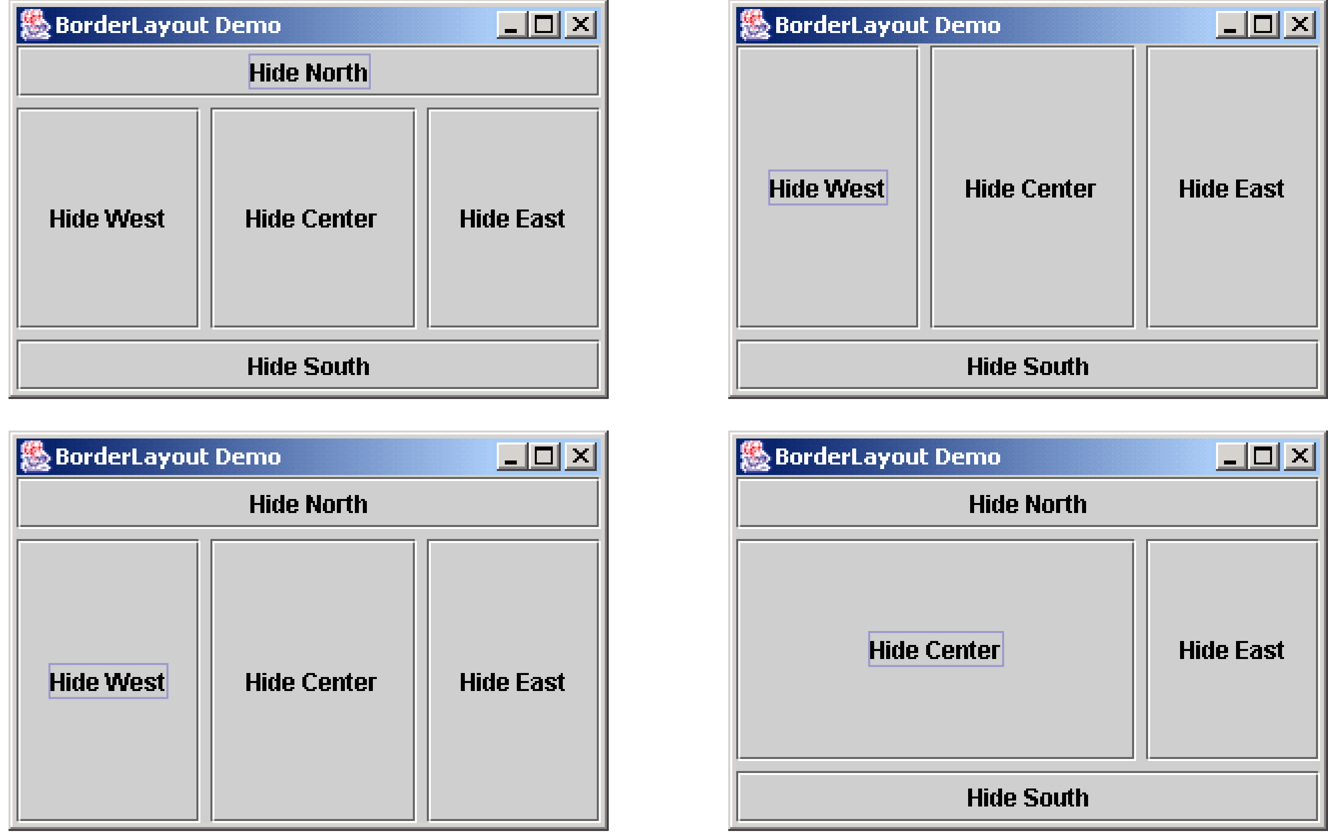


BorderLayoutDemo.java
(3 of 3)

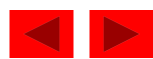
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

29.11.2 BorderLayout

Fig. 29.18 Demonstrating components in BorderLayout.

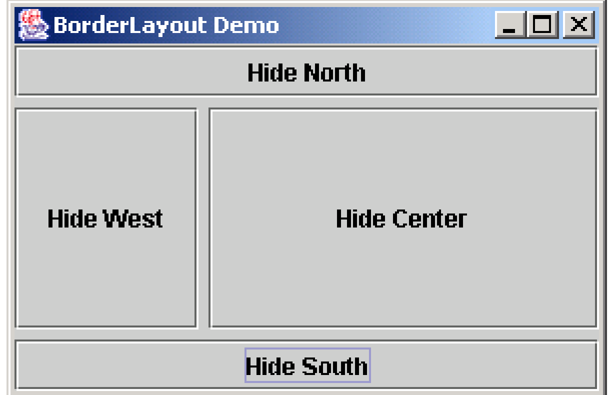


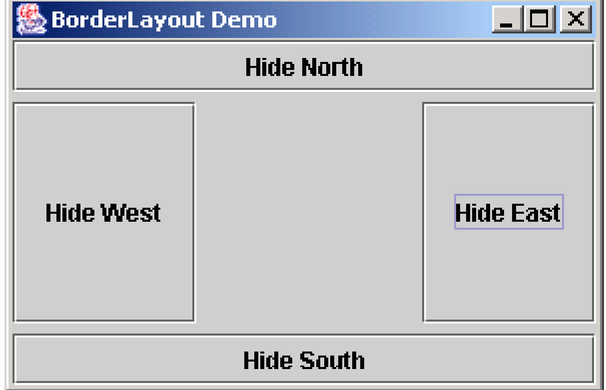
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



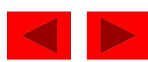
29.11.2 BorderLayout

Fig. 29.18 Demonstrating components in BorderLayout.





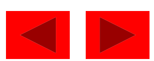
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



29.11.2 BorderLayout

- Methods
  - Constructor: `BorderLayout( hGap, vGap );`
    - `hGap` - horizontal gap space between regions
    - `vGap` - vertical gap space between regions
    - Default is `0` for both
  - Adding components
    - `myLayout.add( component, position )`
    - `component` - component to add
    - `position` - `BorderLayout.NORTH`
      - `SOUTH`, `EAST`, `WEST`, `CENTER` similar
  - `setVisible( boolean )` (in class `JButton`)
    - If `false`, hides component
  - `layoutContainer( container )` - updates container, as before

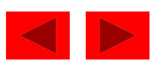
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



29.11.3 GridLayout

- `GridLayout`
  - Divides container into a grid
  - Components placed in rows and columns
  - All components have same width and height
    - Added starting from top left, then from left to right
    - When row full, continues on next row, left to right
- Constructors
  - `GridLayout( rows, columns, hGap, vGap );`
    - Specify number of rows and columns, and horizontal and vertical gaps between elements (in pixels)
  - `GridLayout( rows, columns );`
    - Default `0` for `hGap` and `vGap`

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



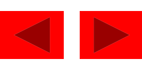
29.11.3 `GridLayout`

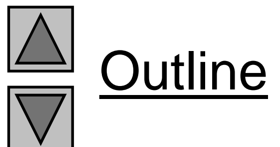
- Updating containers
  - `Container` method `validate`
    - Re-computes the layout for a `Container`
  - Example:

```
c.validate();
```

    - Changes layout and updates `c` if condition met

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.





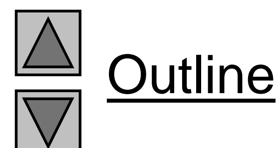
Outline

GridLayoutDemo.java (1 of 3)

```
1  // Fig. 29.19: GridLayoutDemo.java
2  // Demonstrating GridLayout.
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class GridLayoutDemo extends JFrame
8                              implements ActionListener {
9     private JButton b[];
10    private String names[] =
11       { "one", "two", "three", "four", "five", "six" };
12    private boolean toggle = true;
13    private Container c;
14    private GridLayout grid1, grid2;
15
16    public GridLayoutDemo()
17    {
18       super( "GridLayout Demo" );
19
20       grid1 = new GridLayout( 2, 3, 5, 5 );
21       grid2 = new GridLayout( 3, 2 );
22
23       c = getContentPane();
24       c.setLayout( grid1 );
25
```

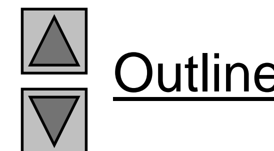
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
26       // create and add buttons
27       b = new JButton[ names.length ];
28
29       for (int i = 0; i < names.length; i++ ) {
30          b[ i ] = new JButton( names[ i ] );
31          b[ i ].addActionListener( this );
32          c.add( b[ i ] );
33       }
34
35       setSize( 300, 150 );
36       show();
37    } // end GridLayoutDemo constructor
38
39    public void actionPerformed( ActionEvent e )
40    {
41       if ( toggle )
42          c.setLayout( grid2 );
43       else
44          c.setLayout( grid1 );
45
46       toggle = !toggle;
47       c.validate();
48    } // end method actionPerformed
49
```



© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
50    public static void main( String args[] )
51    {
52       GridLayoutDemo app = new GridLayoutDemo();
53
54       app.addWindowListener(
55          new WindowAdapter() {
56             public void windowClosing( WindowEvent e )
57             {
58                System.exit( 0 );
59             } // end method windowClosing
60          } // end anonymous inner class
61       ); // end addWindowListener
62    } // end main
63 } // end class GridLayoutDemo
```

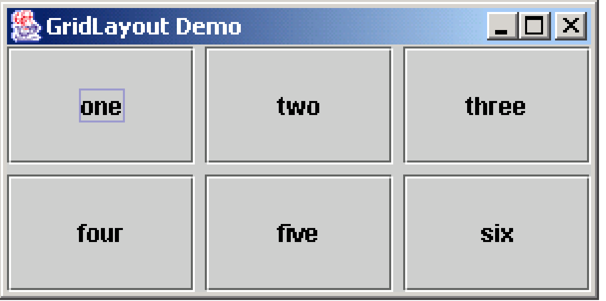


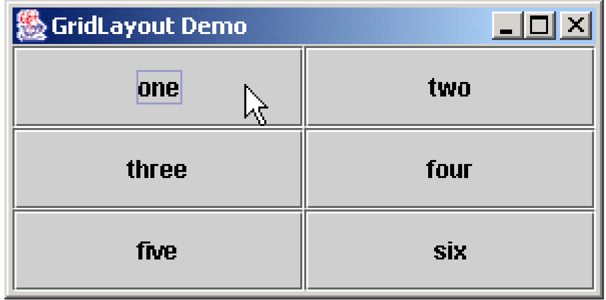
GridLayout-Demo.java (3 of 3)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

29.11.3 GridLayout

Fig. 29.19 Program that demonstrates components in GridLayout.



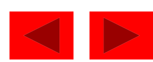


© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

29.12 Panels

- Complex GUIs
  - Each component needs to be placed in an exact location
  - Can use multiple panels
    - Each panel's components arranged in a specific layout
- Panels
  - Class `JPanel` inherits from `JComponent`, which inherits from `java.awt.Container`
    - Every `JPanel` is a `Container`
  - `JPanel`s can have components (and other `JPanel`s) added to them
    - `JPanel` sized to components it contains
    - Grows to accommodate components as they are added

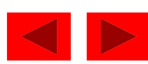
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

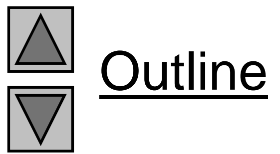


29.12 Panels

- Usage
  - Create panels, and set the layout for each
  - Add components to the panels as needed
  - Add the panels to the content pane (default `BorderLayout`)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

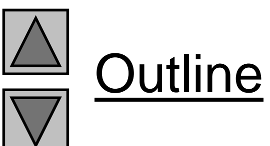





```
1  // Fig. 29.20: PanelDemo.java
2  // Using a JPanel to help lay out components.
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class PanelDemo extends JFrame {
8     private JPanel buttonPanel;
9     private JButton buttons[];
10
11     public PanelDemo()
12     {
13        super( "Panel Demo" );
14
15        Container c = getContentPane();
16        buttonPanel = new JPanel();
17        buttons = new JButton[ 5 ];
18
19        buttonPanel.setLayout(
20           new GridLayout( 1, buttons.length ) );
21
22        for ( int i = 0; i < buttons.length; i++ ) {
23           buttons[ i ] = new JButton( "Button " + (i + 1) );
24           buttonPanel.add( buttons[ i ] );
25        }
26
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
27       c.add( buttonPanel, BorderLayout.SOUTH );
28
29       setSize( 425, 150 );
30       show();
31    } // end PanelDemo constructor
32
33    public static void main( String args[] )
34    {
35       PanelDemo app = new PanelDemo();
36
37       app.addWindowListener(
38          new WindowAdapter() {
39             public void windowClosing( WindowEvent e )
40             {
41                System.exit( 0 );
42             } // end method windowClosing
43          } // end anonymous inner class
44       ); // end addWindowListener
45    } // end main
46 } // end class PanelDemo
```

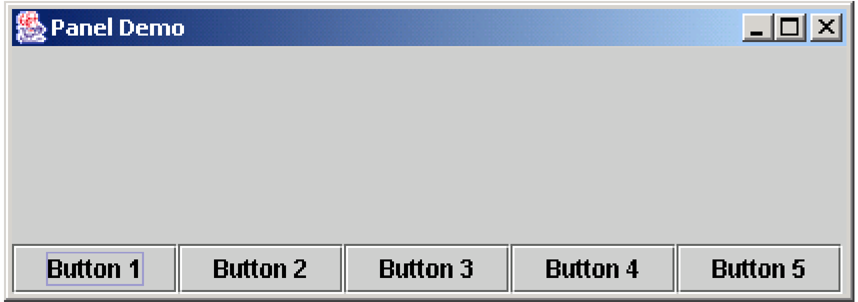


PanelDemo.java (2 of 2)

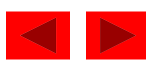
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

29.12 Panels

Fig. 29.20 A `JPanel` with five `JButtons` in a `GridLayout` attached to the `SOUTH` region of a `BorderLayout`.



© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



29.13 Creating a Self-Contained Subclass of JPanel

- JPanel
  - Can be used as a dedicated drawing area
    - Receive mouse events
    - Can extend to create new components
  - Combining Swing GUI components and drawing can lead to improper display
    - GUI may cover drawing, or may be able to draw over GUI components
  - Solution: separate the GUI and graphics
    - Create dedicated drawing areas as subclasses of JPanel

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

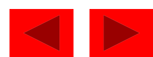
29.13 Creating a Self-Contained Subclass of JPanel

- Swing components inheriting from JComponent
  - Contain method paintComponent
    - Helps to draw properly in a Swing GUI
  - When customizing a JPanel, override paintComponent

```
public void paintComponent( Graphics g )
{
    super.paintComponent( g );
    //additional drawing code
}
```

  - Call to superclass **paintComponent** ensures painting occurs in proper order
    - The call should be the first statement - otherwise, it will erase any drawings before it

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



29.13 Creating a Self-Contained Subclass of JPanel

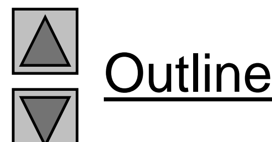
- JFrame and JApplet
  - Not subclasses of JComponent
    - Do not contain paintComponent
  - Override paint to draw directly on subclasses
- Events
  - JPanels do not create events like buttons
  - Can recognize lower-level events
    - Mouse and key events

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

29.13 Creating a Self-Contained Subclass of JPanel

- Example
  - Create a subclass of JPanel named SelfContainedPanel that listens for its own mouse events
    - Draws an oval on itself (overrides paintComponent)
  - Import SelfContainedPanel into another class
    - The other class contains its own mouse handlers
  - Add an instance of SelfContainedPanel to the content pane

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

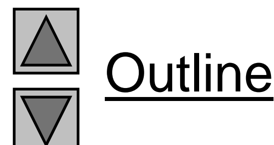


SelfContained-PanelTest .java (1 of 3)

```
1  // Fig. 29.21: SelfContainedPanelTest.java
2  // Creating a self-contained subclass of JPanel
3  // that processes its own mouse events.
4  import java.awt.*;
5  import java.awt.event.*;
6  import javax.swing.*;
7  import com.deitel.chtp3.ch29.SelfContainedPanel;
8
9  public class SelfContainedPanelTest extends JFrame {
10    private SelfContainedPanel myPanel;
11
12    public SelfContainedPanelTest()
13    {
14       myPanel = new SelfContainedPanel();
15       myPanel.setBackground( Color.yellow );
16
17       Container c = getContentPane();
18       c.setLayout( new FlowLayout() );
19       c.add( myPanel );
20
```

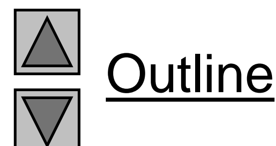
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
21       addMouseMotionListener(
22          new MouseMotionListener() {
23             public void mouseDragged( MouseEvent e )
24             {
25                setTitle( "Dragging: x=" + e.getX() +
26                          "; y=" + e.getY() );
27             } // end method mouseDragged
28
29             public void mouseMoved( MouseEvent e )
30             {
31                setTitle( "Moving: x=" + e.getX() +
32                          "; y=" + e.getY() );
33             } // end method mouseMoved
34          } // end anonymous inner class
35       ); // end addMouseMotionListener
36
37       setSize( 300, 200 );
38       show();
39    } // end SelfContainedPanelTest constructor
40
```



SelfContained-PanelTest.java (2 of 3)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



SelfContained-PanelTest.java (3 of 3)

```
41    public static void main( String args[] )
42    {
43       SelfContainedPanelTest app =
44          new SelfContainedPanelTest();
45
46       app.addWindowListener(
47          new WindowAdapter() {
48             public void windowClosing( WindowEvent e )
49             {
50                System.exit( 0 );
51             } // end method windowClosing
52          } // end anonymous inner class
53       ); // end addWindowListener
54    } // end main
55 } // end class SelfContainedPanelTest
```

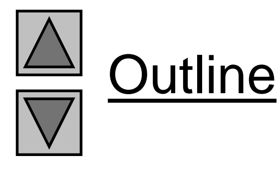
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

SelfContained-Panel.java (1 of 3)

```
56 // Fig. 29.21: SelfContainedPanel.java
57 // A self-contained JPanel class that
58 // handles its own mouse events.
59 package com.deitel.chtp3.ch29;
60
61 import java.awt.*;
62 import java.awt.event.*;
63 import javax.swing.*;
64
65 public class SelfContainedPanel extends JPanel {
66    private int x1, y1, x2, y2;
67
68    public SelfContainedPanel()
69    {
70       addMouseListener(
71          new MouseAdapter() {
72             public void mousePressed( MouseEvent e )
73             {
74                x1 = e.getX();
75                y1 = e.getY();
76             } // end method mousePressed
77
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

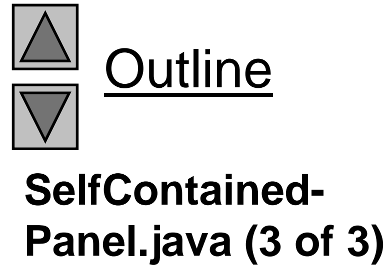
```
78             public void mouseReleased( MouseEvent e )
79             {
80                x2 = e.getX();
81                y2 = e.getY();
82                repaint();
83             } // end method mouseReleased
84          } // end anonymous inner class
85       ); // end addMouseListener
86
87       addMouseMotionListener(
88          new MouseMotionAdapter() {
89             public void mouseDragged( MouseEvent e )
90             {
91                x2 = e.getX();
92                y2 = e.getY();
93                repaint();
94             } // end method mouseDragged
95          } // end anonymous inner class
96       ); // end addMouseMotionListener
97    } // end SelfContainedPanel constructor
98
```



SelfContained-Panel.java (2 of 3)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
99    public Dimension getPreferredSize()
100   {
101      return new Dimension( 150, 100 );
102   } // end method getPreferredSize
103
104   public void paintComponent( Graphics g )
105   {
106      super.paintComponent( g );
107
108      g.drawOval( Math.min( x1, x2 ), Math.min( y1, y2 ),
109                  Math.abs( x1 - x2 ), Math.abs( y1 - y2 ) );
110   } // end method paintComponent
111 } // end class SelfContainedPanel
```

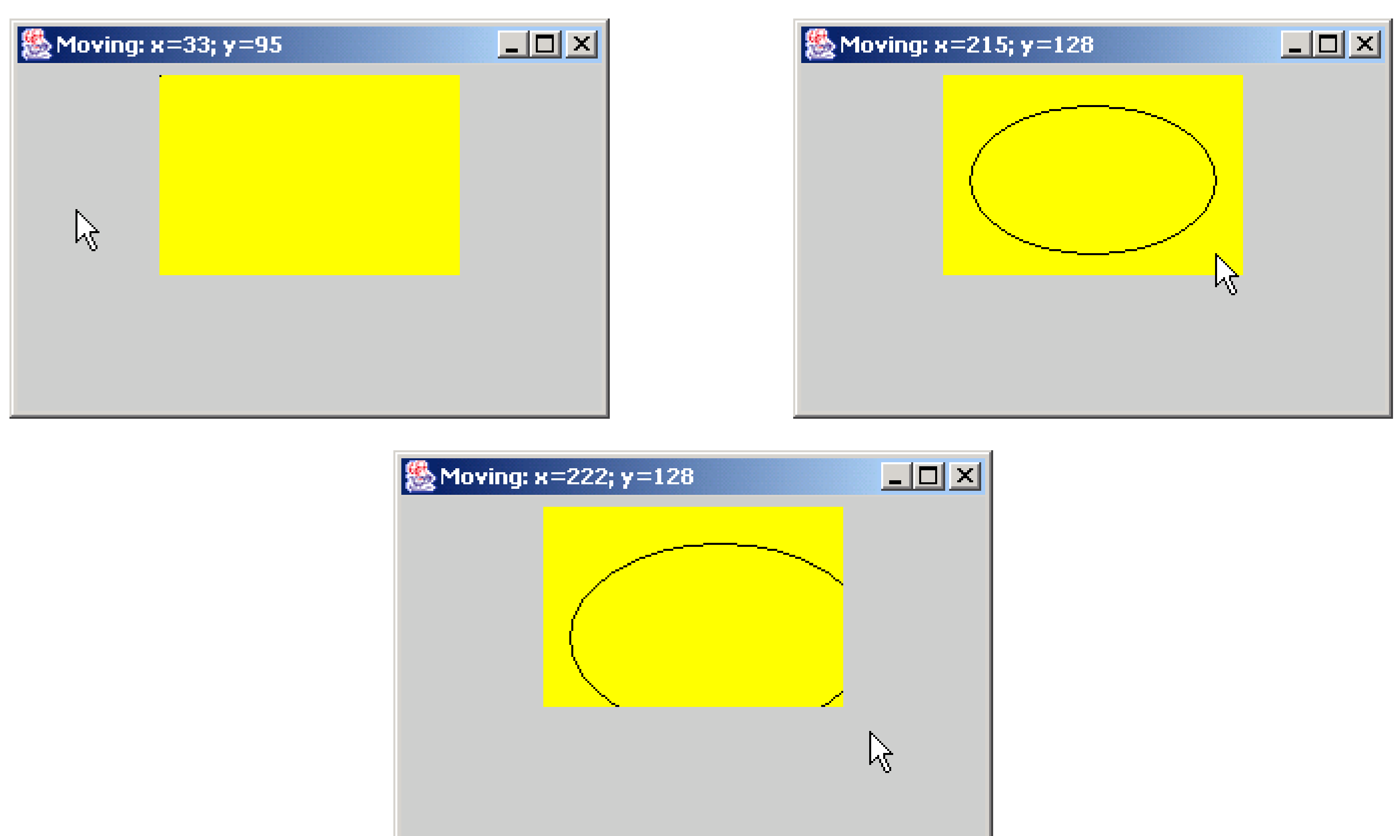


SelfContained-Panel.java (3 of 3)

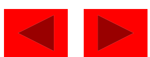
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

29.13 Creating a Self-Contained Subclass of JPanel

Fig. 29.21 Capturing mouse events with a JPanel—SelfContainedPanel.java.



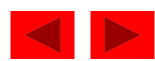
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



29.14 Windows

- `JFrame`
  - Inherits from `java.awt.Frame`, which inherits from `java.awt.Window`
  - `JFrame` is a window with a title bar and a border
    - Not a lightweight component - not written completely in Java
    - Window part of local platform's GUI components
      - Different for Windows, Macintosh, and UNIX
- `JFrame` operations when user closes window
  - Controlled with method `setDefaultCloseOperation`
    - Interface `WindowConstants` (`javax.swing`) has three constants to use
    - `DISPOSE_ON_CLOSE`, `DO_NOTHING_ON_CLOSE`, `HIDE_ON_CLOSE` (default)

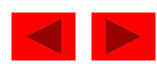
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



29.14 Windows

- Windows take up valuable resources
  - Explicitly remove windows when not needed with method `dispose` (of class `Window`, indirect superclass of `JFrame`)
    - Or, use `setDefaultCloseOperation`
  - `DO_NOTHING_ON_CLOSE` - you determine what happens when user wants to close window
- Display
  - By default, window not displayed until method `show` called
  - Can display by calling method `setVisible( true )`
  - Method `setSize` - make sure to set a window's size, otherwise only the title bar will appear

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



29.14 Windows

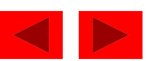
- All windows generate window events
  - `addWindowListener`
  - `WindowListener` interface has 7 methods
    - `windowActivated`
    - `windowClosed` (called after window closed)
    - `windowClosing` (called when user initiates closing)
    - `windowDeactivated`
    - `windowIconified` (minimized)
    - `windowDeiconified`
    - `windowOpened`

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

29.15 Using Menus with Frames

- Menus
  - Important part of GUIs
  - Perform actions without cluttering GUI
  - Attached to objects of classes that have method **`setJMenuBar`**
    - **`JFrame`** and **`JApplet`**
- Classes used to define menus
  - **`JMenuBar`** - container for menus, manages menu bar
  - **`JMenuItem`** - manages menu items
    - Menu items - GUI components inside a menu
    - Can initiate an action or be a submenu

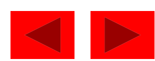
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



29.15 Using Menus with Frames

- Classes used to define menus (continued)
  - `JMenu` - manages menus
    - Menus contain menu items, and are added to menu bars
    - Can be added to other menus as submenus
    - When clicked, expands to show list of menu items
  - `JCheckBoxMenuItem`
    - Manages menu items that can be toggled
    - When selected, check appears to left of item
  - `JRadioButtonMenuItem`
    - Manages menu items that can be toggled
    - When multiple `JRadioButtonMenuItem`s are part of a group, only one can be selected at a time
    - When selected, filled circle appears to left of item

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



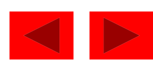
29.15 Using Menus with Frames

- Mnemonics
  - Provide quick access to menu items (File)
    - Can be used with classes that have subclass `javax.swing.AbstractButton`
  - Use method `setMnemonic`

```
JMenu fileMenu = new JMenu( "File" )
fileMenu.setMnemonic( 'F' );
```

    - Press `Alt + F` to access menu
- Methods
  - `setSelected( true )`
    - Of class `AbstractButton`
    - Sets button/item to selected state

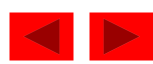
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



29.15 Using Menus with Frames

- Methods (continued)
  - `addSeparator()`
    - Class `JMenu`
    - Inserts separator line into menu
- Dialog boxes
  - Modal - No other window can be accessed while it is open (default)
    - Modeless - other windows can be accessed
  - `JOptionPane.showMessageDialog( parentWindow, String, title, messageType )`
  - `parentWindow` - determines where dialog box appears
    - `null` - displayed at center of screen
    - window specified - dialog box centered horizontally over parent

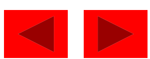
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



29.15 Using Menus with Frames

- Using menus
  - Create menu bar
    - Set menu bar for `JFrame` (`setJMenuBar( myBar );`
  - Create menus
    - Set Mnemonics
  - Create menu items
    - Set Mnemonics
    - Set event handlers
  - If using `JRadioButtonMenuItems`
    - Create a group: `myGroup = new ButtonGroup();`
    - Add `JRadioButtonMenuItems` to the group

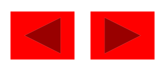
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



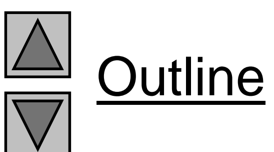
29.15 Using Menus with Frames

- Using menus (continued)
  - Add menu items to appropriate menus
    - `myMenu.add( myItem );`
    - Insert separators if necessary: `myMenu.addSeparator();`
  - If creating submenus, add submenu to menu
    - `myMenu.add( mySubMenu );`
  - Add menus to menu bar
    - `myMenuBar.add( myMenu );`
- Example
  - Use menus to alter text in a `JLabel`
  - Change color, font, style
  - Have a "File" menu with a "About" and "Exit" items

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



```
1  // Fig. 29.22: MenuTest.java
2  // Demonstrating menus
3  import javax.swing.*;
4  import java.awt.event.*;
5  import java.awt.*;
6
7  public class MenuTest extends JFrame {
8     private Color colorValues[] =
9        { Color.black, Color.blue, Color.red, Color.green };
10    private JRadioButtonMenuItem colorItems[], fonts[];
11    private JCheckBoxMenuItem styleItems[];
12    private JLabel display;
13    private ButtonGroup fontGroup, colorGroup;
14    private int style;
15
16    public MenuTest()
17    {
18       super( "Using JMenus" );
19
20       JMenuBar bar = new JMenuBar();  // create menubar
21       setJMenuBar( bar );  // set the menubar for the JFrame
22
```

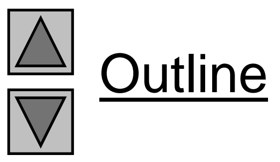


Outline

MenuTest.java (1 of 7)

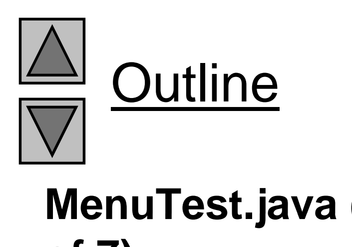
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
23       // create File menu and Exit menu item
24       JMenu fileMenu = new JMenu( "File" );
25       fileMenu.setMnemonic( 'F' );
26       JMenuItem aboutItem = new JMenuItem( "About..." );
27       aboutItem.setMnemonic( 'A' );
28       aboutItem.addActionListener(
29          new ActionListener() {
30             public void actionPerformed( ActionEvent e )
31             {
32                JOptionPane.showMessageDialog( MenuTest.this,
33                   "This is an example\nof using menus",
34                   "About", JOptionPane.PLAIN_MESSAGE );
35             } // end method actionPerformed
36          } // end anonymous inner class
37       ); // end addActionListener
38       fileMenu.add( aboutItem );
39
40       JMenuItem exitItem = new JMenuItem( "Exit" );
41       exitItem.setMnemonic( 'x' );
42       exitItem.addActionListener(
43          new ActionListener() {
44             public void actionPerformed( ActionEvent e )
45             {
46                System.exit( 0 );
47             } // end method actionPerformed
```



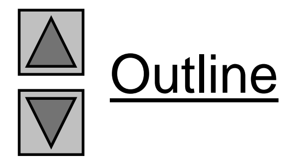
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
48          } // end anonymous inner class
49       ); // end addActionListener
50       fileMenu.add( exitItem );
51       bar.add( fileMenu );    // add File menu
52
53       // create the Format menu, its submenus and menu items
54       JMenu formatMenu = new JMenu( "Format" );
55       formatMenu.setMnemonic( 'r' );
56
57       // create Color submenu
58       String colors[] =
59          { "Black", "Blue", "Red", "Green" };
60       JMenu colorMenu = new JMenu( "Color" );
61       colorMenu.setMnemonic( 'C' );
62       colorItems = new JRadioButtonMenuItem[ colors.length ];
63       colorGroup = new ButtonGroup();
64       ItemHandler itemHandler = new ItemHandler();
65
66       for ( int i = 0; i < colors.length; i++ ) {
67          colorItems[ i ] =
68             new JRadioButtonMenuItem( colors[ i ] );
69          colorMenu.add( colorItems[ i ] );
70          colorGroup.add( colorItems[ i ] );
71          colorItems[ i ].addActionListener( itemHandler );
72       } // end for
73
```



MenuTest.java (3 of 7)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



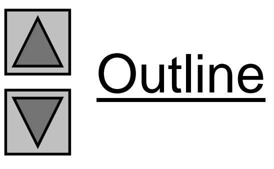
Outline

MenuTest.java (4 of 7)

```
74       colorItems[ 0 ].setSelected( true );
75       formatMenu.add( colorMenu );
76       formatMenu.addSeparator();
77
78       // create Font submenu
79       String fontNames[] =
80          { "TimesRoman", "Courier", "Helvetica" };
81       JMenu fontMenu = new JMenu( "Font" );
82       fontMenu.setMnemonic( 'n' );
83       fonts = new JRadioButtonMenuItem[ fontNames.length ];
84       fontGroup = new ButtonGroup();
85
86       for ( int i = 0; i < fonts.length; i++ ) {
87          fonts[ i ] =
88             new JRadioButtonMenuItem( fontNames[ i ] );
89          fontMenu.add( fonts[ i ] );
90          fontGroup.add( fonts[ i ] );
91          fonts[ i ].addActionListener( itemHandler );
92       } // end for
93
94       fonts[ 0 ].setSelected( true );
95       fontMenu.addSeparator();
96
97       String styleNames[] = { "Bold", "Italic" };
98       styleItems = new JCheckBoxMenuItem[ styleNames.length ];
99       StyleHandler styleHandler = new StyleHandler();
100
```

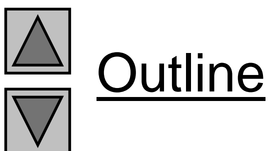
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
101      for ( int i = 0; i < styleNames.length; i++ ) {
102         styleItems[ i ] =
103            new JCheckBoxMenuItem( styleNames[ i ] );
104         fontMenu.add( styleItems[ i ] );
105         styleItems[ i ].addItemListener( styleHandler );
106      } // end for
107
108      formatMenu.add( fontMenu );
109      bar.add( formatMenu );  // add Format menu
110
111      display = new JLabel(
112         "Sample Text", SwingConstants.CENTER );
113      display.setForeground( colorValues[ 0 ] );
114      display.setFont(
115         new Font( "TimesRoman", Font.PLAIN, 72 ) );
116
117      getContentPane().setBackground( Color.cyan );
118      getContentPane().add( display, BorderLayout.CENTER );
119
120      setSize( 500, 200 );
121      show();
122   } // end MenuTest constructor
123
```

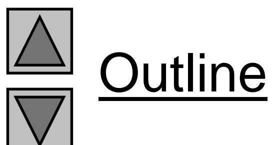


© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
124    public static void main( String args[] )
125    {
126       MenuTest app = new MenuTest();
127
128       app.addWindowListener(
129          new WindowAdapter() {
130             public void windowClosing( WindowEvent e )
131             {
132                System.exit( 0 );
133             } // end method windowClosing
134          } // end anonymous inner class
135       ); // end addWindowListener
136    } // end main
137
138    class ItemHandler implements ActionListener {
139       public void actionPerformed( ActionEvent e )
140       {
141          for ( int i = 0; i < colorItems.length; i++ )
142             if ( colorItems[ i ].isSelected() ) {
143                display.setForeground( colorValues[ i ] );
144                break;
145             }
146
```



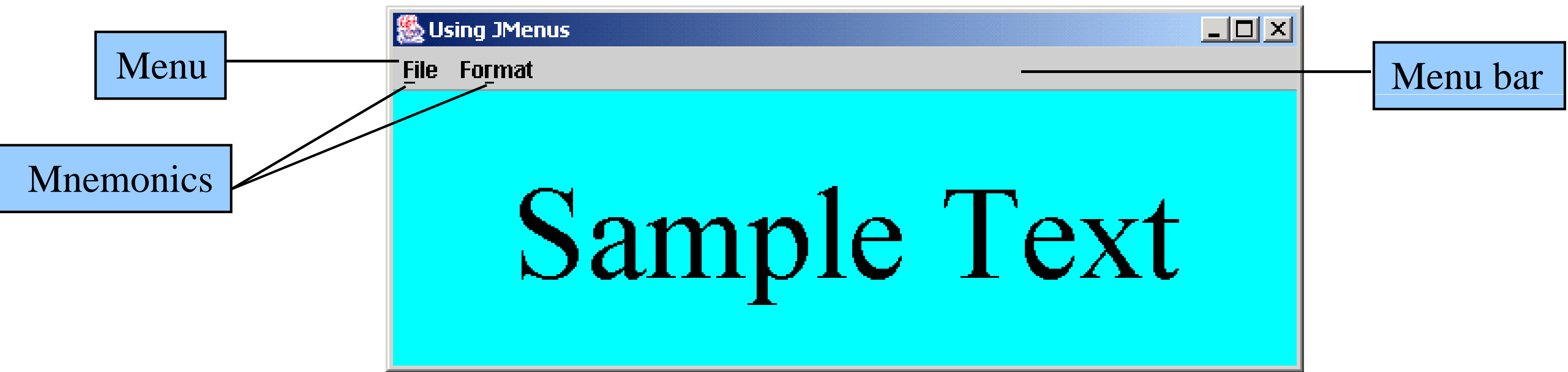
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



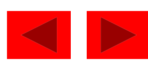

```
147          for ( int i = 0; i < fonts.length; i++ )
148             if ( e.getSource() == fonts[ i ] ) {
149                display.setFont( new Font(
150                   fonts[ i ].getText(), style, 72 ) );
151                break;
152             }
153
154          repaint();
155       } // end method actionPerformed
156    } // end inner class ItemHandler
157
158    class StyleHandler implements ItemListener {
159       public void itemStateChanged( ItemEvent e )
160       {
161          style = 0;
162
163          if ( styleItems[ 0 ].isSelected() )
164             style += Font.BOLD;
165
166          if ( styleItems[ 1 ].isSelected() )
167             style += Font.ITALIC;
168
169          display.setFont( new Font(
170             display.getFont().getName(), style, 72 ) );
171
172          repaint();
173       } // end method itemStateChanged
174    } // end inner class StyleHandler
175 } // end class MenuTest
```

29.15 Using Menus with Frames

Fig. 29.22 Using JMenus and mnemonics.

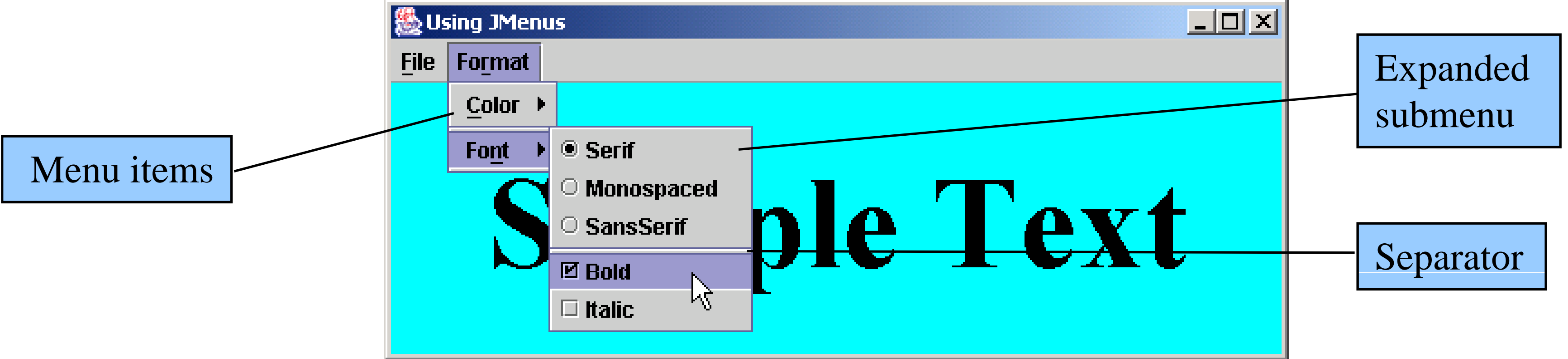


© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

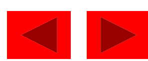


29.15 Using Menus with Frames

Fig. 29.22 Using JMenus and mnemonics.



© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

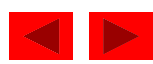


Java Multimedia: Images, Animation, Audio and Video

Outline

30.1 Introduction
30.2 Loading, Displaying and Scaling Images
30.3 Loading and Playing Audio Clips
30.4 Animating a Series of Images
30.5 Animation Issues
30.6 Customizing Applets via the HTML param Tag
30.7 Image Maps
30.8 Internet and World Wide Web Resources

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Objectives

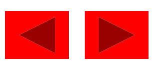
- In this chapter, you will learn:
  - To understand how to get and display images.
  - To be able to create animations from sequences of images; to control animation speed and flicker.
  - To be able to get, play, loop and stop sounds.
  - To be able to monitor the loading of images with class `MediaTracker`; to create image maps.
  - To customize applets with the `param` tag.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

30.1 Introduction

- Revolution in computer industry
  - Before, computers used for high-speed calculations
  - Now, data manipulation important
- Multimedia
  - "sizzle" of Java - images, sound, video
  - CDs, DVDs, video cards
  - Demands extraordinary computing power
    - Fast processors making multimedia possible
- Java
  - Has built-in multimedia capabilities
    - Most programming languages do not
  - Develop powerful multimedia applications

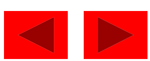
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



30.2 Loading, Displaying and Scaling Images

- Java Multimedia
  - Graphics, images, animations, sounds, and video
    - Begin with images
- Class `Image` (`java.awt`)
  - Abstract class, cannot create an object directly
    - Must request that an `Image` be loaded and returned to you
  - Class `Applet` (superclass of `JApplet`) has this method
    - `getImage( imageLocation, filename );`
    - `imageLocation - getDocumentBase() - URL` (address) of HTML file
    - `filename` - Java supports `.gif` and `.jpg` (`.jpeg`)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



30.2 Loading, Displaying and Scaling Images (II)

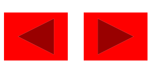
- Displaying Images with `drawImage`
  - Many overloaded versions

```
g.drawImage( myImage, x, y, ImageObserver );
```

  - `myImage` - Image object
  - `x, y` - coordinates to display image
  - `ImageObserver` - object on which image is displayed
    - Use `"this"` to indicate the applet
    - Can be any object that implements `ImageObserver` interface

  - ```
    g.drawImage( myImage, x, y, width, height,
              ImageObserver );
    ```
    - `width` and `height` - dimensions of image (automatically scaled)
      - `getWidth(), getHeight()` - return dimensions of applet

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.


    ```

30.2 Loading, Displaying and Scaling Images (III)

- Class `ImageIcon`
  - Not an abstract class (can create objects)
  - Example constructor

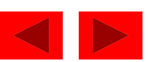
```
private ImageIcon myIcon;
myIcon = new ImageIcon( "myIcon.gif" );
```

- Displaying `Icons` with method `paintIcon`

```
myIcon.paintIcon( Component, Graphics, x, y )
```

  - `Component` - `Component` object on which to display image (`this`)
  - `Graphics` - `Graphics` object used to render image (g)
  - `x`, `y` - coordinates of `Icon`

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



30.2 Loading, Displaying and Scaling Images (IV)

- Usage
  - `ImageIcons` are simpler than `Images`
    - Create objects directly
    - No need for `ImageObserver` reference
  - However, cannot scale `ImageIcons`
- Scaling
  - Use `ImageIcon` method `getImage`
    - Returns `Image` reference
    - This can be used with `drawImage` and be scaled

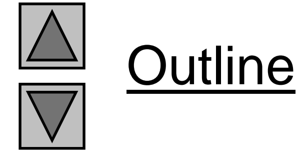
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
1  // Fig. 30.1: LoadImageAndScale.java
2  // Load an image and display it in its original size
3  // and scale it to twice its original width and height.
4  // Load and display the same image as an ImageIcon.
5  import java.applet.Applet;
6  import java.awt.*;
7  import javax.swing.*;
8
9  public class LoadImageAndScale extends JApplet {
10    private Image logo1;
11    private ImageIcon logo2;
12
13    // load the image when the applet is loaded
14    public void init()
15    {
16       logo1 = getImage( getDocumentBase(), "logo.gif" );
17       logo2 = new ImageIcon( "logo.gif" );
18    } // end method init
19
20    // display the image
21    public void paint( Graphics g )
22    {
23       // draw the original image
24       g.drawImage( logo1, 0, 0, this );
25
```

LoadImage-AndScale.java (Part 1 of 2)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
26        // draw the image scaled to fit the width of the applet
27        // and the height of the applet minus 120 pixels
28        g.drawImage( logo1, 0, 120,
29                     getWidth(), getHeight() - 120, this );
30
31        // draw the icon using its paintIcon method
32        logo2.paintIcon( this, g, 180, 0 );
33     } // end method pain
34  } // end class LoadImageAndSacle
```



LoadImage-AndScale.java (Part 2 of 2)

Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

30.3 Loading and Playing Audio Clips

- Audio clips
  - Require speakers and a sound board
  - Sound engine - plays audio clips
    - Supports `.au`, `.wav`, `.aif`, `.mid`
    - Java Media Framework supports additional formats
- Playing audio clips
  - `play` method in `Applet`
  - Plays clip once, marked for garbage collection when finished

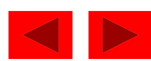
```
play( location, soundFileName );
```

`location` - `getDocumentBase` (URL of HTML file)

```
play( soundURL );
```

`soundURL` - URL that contains location and filename of clip

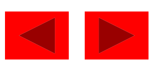
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



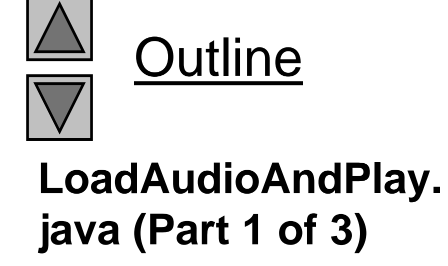
30.3 Loading and Playing Audio Clips (II)

- Playing audio clips
  - Method `play` from `AudioClip` interface
  - More flexible than `Applet` method `play`
    - Audio stored in program, can be reused
  - `getAudioClip`
    - Returns reference to an `AudioClip`
    - Same format as `Applet` method `play`
      - `getAudioClip( location, filename )`
      - `getAudioClip( soundURL )`
  - Once `AudioClip` loaded, use methods
    - `play` - plays audio once
    - `loop` - continuous loops audio in background
    - `stop` - terminates clip that is currently playing

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



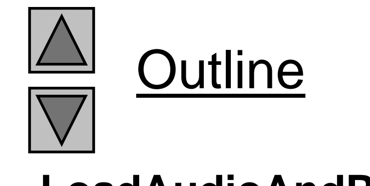
```
1  // Fig. 30.2: LoadAudioAndPlay.java
2  // Load an audio clip and play it.
3  import java.applet.*;
4  import java.awt.*;
5  import java.awt.event.*;
6  import javax.swing.*;
7
8  public class LoadAudioAndPlay extends JApplet {
9     private AudioClip sound1, sound2, currentSound;
10    private JButton playSound, loopSound, stopSound;
11    private JComboBox chooseSound;
12
13    // load the image when the applet begins executing
14    public void init()
15    {
16       Container c = getContentPane();
17       c.setLayout( new FlowLayout() );
18
19       String choices[] = { "Welcome", "Hi" };
20       chooseSound = new JComboBox( choices );
21       chooseSound.addItemListener(
22          new ItemListener() {
23             public void itemStateChanged( ItemEvent e )
24             {
25                currentSound.stop();
26
```



LoadAudioAndPlay.java (Part 1 of 3)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

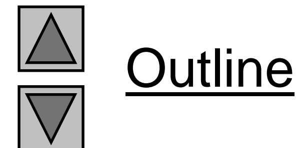
```
27                currentSound =
28                   chooseSound.getSelectedIndex() == 0 ?
29                      sound1 : sound2;
30             } // end method itemStateChanged
31          } // end anonymous inner class
32       ); // end addItemListener
33       c.add( chooseSound );
34 
35       ButtonHandler handler = new ButtonHandler();
36       playSound = new JButton( "Play" );
37       playSound.addActionListener( handler );
38       c.add( playSound );
39       loopSound = new JButton( "Loop" );
40       loopSound.addActionListener( handler );
41       c.add( loopSound );
42       stopSound = new JButton( "Stop" );
43       stopSound.addActionListener( handler );
44       c.add( stopSound );
45 
46       sound1 = getAudioClip(
47                   getDocumentBase(), "welcome.wav" );
48       sound2 = getAudioClip(
49                   getDocumentBase(), "hi.au" );
50       currentSound = sound1;
51    } // end method init
52 
```



LoadAudioAndPlay.java (Part 2 of 3)

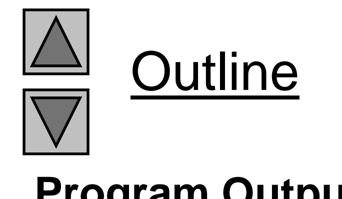
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
53    // stop the sound when the user switches Web pages
54    // (i.e., be polite to the user)
55    public void stop()
56    {
57       currentSound.stop();
58    } // end method stop
59
60    private class ButtonHandler implements ActionListener {
61       public void actionPerformed( ActionEvent e )
62       {
63          if ( e.getSource() == playSound )
64             currentSound.play();
65          else if ( e.getSource() == loopSound )
66             currentSound.loop();
67          else if ( e.getSource() == stopSound )
68             currentSound.stop();
69       } // end method actionPerformed
70    } // end inner class ButtonHandler
71 } // end class LoadAudioAndPlay
```



LoadAudioAndPlay.java (Part 3 of 3)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

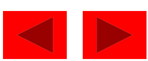
30.4 Animating a Series of Images

- Following example
  - Use a series of images stored in an array
  - Use same techniques to load and display `ImageIcons`
- Class `Timer`
  - Generates `ActionEvents` at a fixed interval in milliseconds

    `Timer ( animationDelay, ActionListener );`

    `ActionListener` - `ActionListener` that will respond to `ActionEvents`
  - Methods
    - `start`
    - `stop`
    - `restart`
    - `isRunning`

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

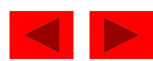


30.4 Animating a Series of Images (II)

- Method `repaint`
  - Calls `update`, which calls `paintComponent`
    - Subclasses of `JComponent` should draw in method `paintComponent`
    - Call superclass's `paintComponent` to make sure Swing components displayed properly
- View area
  - Width and height specify entire window, not client area
  - `Dimension` objects
    - Contain `width` and `height` values

```
myDimObject = new Dimension( 100, 200 );
myDimObject.width
```

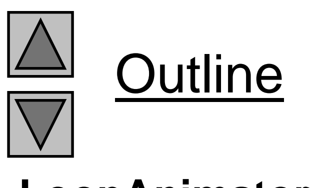
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



30.4 Animating a Series of Images (III)

- `getImageLoadStatus`
  - `ImageIcon` method
    - Determines if image is completely loaded into memory
    - Only complete images should be displayed (smooth animation)
  - If loaded, returns `MediaTracker.COMPLETE`
  - `MediaTracker`
    - Can determine when images are loaded, or force program to wait if not
    - `ImageIcon` creates our `MediaTracker` for us

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

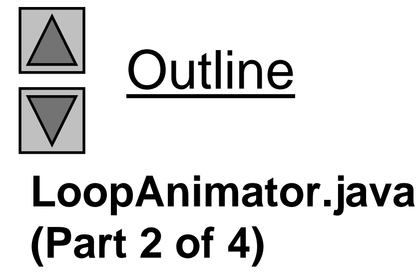


LoopAnimator.java (Part 1 of 4)

```
1  // Fig. 30.3: LogoAnimator.java
2  // Animation a series of images
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class LogoAnimator extends JPanel
8                            implements ActionListener {
9     protected ImageIcon images[];
10    protected int totalImages = 30,
11                  currentImage = 0,
12                  animationDelay = 50; // 50 millisecond delay
13    protected Timer animationTimer;
14
15    public LogoAnimator()
16    {
17       setSize( getPreferredSize() );
18
19       images = new ImageIcon[ totalImages ];
20
21       for ( int i = 0; i < images.length; ++i )
22          images[ i ] =
23             new ImageIcon( "images/deitel" + i + ".gif" );
24
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
25       startAnimation();
26    } // end LogoAnimator constructor
27
28    public void paintComponent( Graphics g )
29    {
30       super.paintComponent( g );
31
32       if ( images[ currentImage ].getImageLoadStatus() ==
33            MediaTracker.COMPLETE ) {
34          images[ currentImage ].paintIcon( this, g, 0, 0 );
35          currentImage = ( currentImage + 1 ) % totalImages;
36       }
37    } // end method paintComponent
38
39    public void actionPerformed( ActionEvent e )
40    {
41       repaint();
42    } // end method actionPerformed
43
44    public void startAnimation()
45    {
46       if ( animationTimer == null ) {
47          currentImage = 0;
48          animationTimer = new Timer( animationDelay, this );
49          animationTimer.start();
50       }
```



LoopAnimator.java (Part 2 of 4)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
51       else  // continue from last image displayed
52          if ( ! animationTimer.isRunning() )
53             animationTimer.restart();
54    } // end method startAnimation
55
56    public void stopAnimation()
57    {
58       animationTimer.stop();
59    } // end method stopAnimation
60
61    public Dimension getMinimumSize()
62    {
63       return getPreferredSize();
64    } // end method getMinimumSize
65
66    public Dimension getPreferredSize()
67    {
68       return new Dimension( 160, 80 );
69    } // end method getPreferredSize
70
71    public static void main( String args[] )
72    {
73       LogoAnimator anim = new LogoAnimator();
74
```

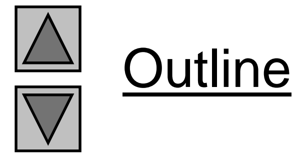
LoopAnimator.java (Part 3 of 4)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

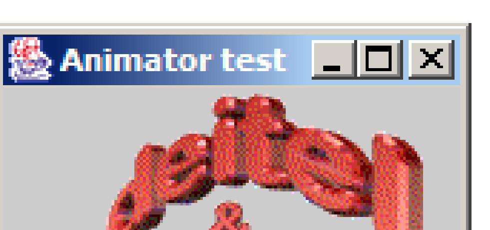
```
75        JFrame app = new JFrame( "Animator test" );
76        app.getContentPane().add( anim, BorderLayout.CENTER );
77
78        app.addWindowListener(
79           new WindowAdapter() {
80              public void windowClosing( WindowEvent e )
81              {
82                 System.exit( 0 );
83              } // end method windowClosing
84           } // end anonymous inner class
85        ); // end addWindowListener
86
87        // The constants 10 and 30 are used below to size the
88        // window 10 pixels wider than the animation and
89        // 30 pixels taller than the animation.
90        app.setSize( anim.getPreferredSize().width + 10,
91                     anim.getPreferredSize().height + 30 );
92        app.show();
93     } // end main
94  } // end class LogoAnimator
```

LoopAnimator.java (Part 4 of 4)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Program Output

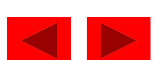


© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

30.5 Animation Issues

- Storing images
  - Interlaced/non-interlaced formats
    - Specifies order in which pixels are stored
    - Non-interlaced - pixels stored in order they appear on screen
      - Image appears in chunks from top to bottom as it is loaded
    - Interlaced - pixels stored in rows, but out of order
      - Image appears to fade in and become more clear
- Animation flickers
  - Due to `update` being called in response to `repaint`
  - In AWT GUI components
    - Draws filled rectangle in background color where image was
    - Draw image, sleep, clear background (flicker), draw next image...
  - Swing's `JPanel` overrides `update` to avoid this

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



30.5 Animation Issues (II)

- Double buffering
  - Used to smooth animations
  - Program renders one image on screen
    - Builds next image in off-screen buffer
  - When time to display next image, done smoothly
    - Partial images user would have seen (while image loads) are hidden
    - All pixels for next image displayed at once
  - Space/Time tradeoff
    - Reduces flicker, but can slow animation speed and uses more memory
  - Used by Swing GUI components by default

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

30.6 Customizing Applets via the HTML `param` Tag

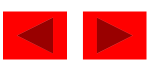
- Applets
  - Customize through parameters in HTML file that invokes it

```
<html>
<applet code="LogoApplet.class" width=400
height=400>
<param name="totalimages" value="30">
<param name="imagename" value="deitel">
<param name="animationdelay" value="200">
</applet>
</html>
```

  - Invokes applet `LogoApplet`
  - `param` tags
    - Each has a `name` and a `value`
    - Use Applet method `getParameter` (returns a `String`)

```
parameter = getParameter( "animationdelay" );
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



30.6 Customizing Applets via the HTML `param` Tag (II)

- Following example
  - Use the `LogoAnimator` class as before, but modified slightly
  - Create Applet `LogoApplet`
    - Takes parameters
    - Creates `LogoAnimator` object using the parameters
    - Plays animation

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

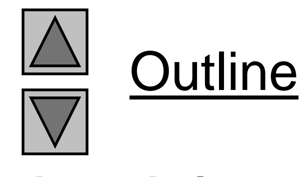
```java
1  // Fig. 30.4: LogoAnimator.java
2  // Animating a series of images
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class LogoAnimator extends JPanel
8                            implements ActionListener {
9     protected ImageIcon images[];
10    protected int totalImages = 30,
11                  currentImage = 0,
12                  animationDelay = 50; // 50 millisecond delay
13    protected String imageName = "deitel";
14    protected Timer animationTimer;
15
16    public LogoAnimator()
17    {
18       initializeAnim();
19    } // end LogoAnimator constructor
20
21    // new constructor to support customization
22    public LogoAnimator( int num, int delay, String name )
23    {
24       totalImages = num;
25       animationDelay = delay;
26       imageName = name;
27
```

Outline

LogoAnimator.java (Part 1 of 5)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

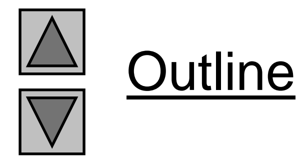
```
28          initializeAnim();
29       } // end LogoAnimator constructor
30
31       private void initializeAnim()
32       {
33          images = new ImageIcon[ totalImages ];
34
35          for ( int i = 0; i < images.length; ++i )
36             images[ i ] = new ImageIcon( "images/" +
37                                 imageName + i + ".gif" );
38
39          // moved here so getPreferredSize can check the size of
40          // the first loaded image.
41          setSize( getPreferredSize() );
42
43          startAnimation();
44       } // end method initializeAnim
45
46       public void paintComponent( Graphics g )
47       {
48          super.paintComponent( g );
49
```



Outline

LogoAnimator.java (Part 2 of 5)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Outline

LogoAnimator.java (Part 3 of 5)

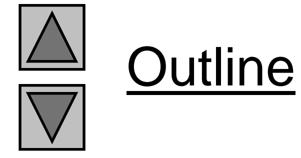
```
50        if ( images[ currentImage ].getImageLoadStatus() ==
51             MediaTracker.COMPLETE ) {
52           images[ currentImage ].paintIcon( this, g, 0, 0 );
53           currentImage = ( currentImage + 1 ) % totalImages;
54        } // end if
55     } // end method paintComponent
56
57     public void actionPerformed( ActionEvent e )
58     {
59        repaint();
60     } // end method actionPerformed
61
62     public void startAnimation()
63     {
64        if ( animationTimer == null ) {
65           currentImage = 0;
66           animationTimer = new Timer( animationDelay, this );
67           animationTimer.start();
68        }
69        else  // continue from last image displayed
70           if ( ! animationTimer.isRunning() )
71              animationTimer.restart();
72     } // end method startAnimation
73
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
74    public void stopAnimation()
75    {
76       animationTimer.stop();
77    } // end method stopAnimation
78
79    public Dimension getMinimumSize()
80    {
81       return getPreferredSize();
82    } // end method getMinimumSize
83
84    public Dimension getPreferredSize()
85    {
86       return new Dimension( images[ 0 ].getIconWidth(),
87                             images[ 0 ].getIconHeight() );
88    } // end method getPreferredSize
89
90    public static void main( String args[] )
91    {
92       LogoAnimator anim = new LogoAnimator();
93
94       JFrame app = new JFrame( "Animator test" );
95       app.getContentPane().add( anim, BorderLayout.CENTER );
96
```

LogoAnimator.java (Part 4 of 5)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



LogoAnimator.java (Part 5 of 5)

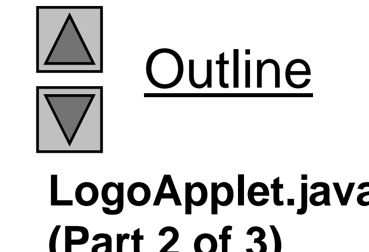
```
97         app.addWindowListener(
98            new WindowAdapter() {
99               public void windowClosing( WindowEvent e )
100              {
101                 System.exit( 0 );
102              } // end method windowClosing
103           } // end anonymous inner class
104        ); // end addWindowListener
105
106        app.setSize( anim.getPreferredSize().width + 10,
107                     anim.getPreferredSize().height + 30 );
108        app.show();
109     } // end main
110 } // end class LogoAnimator
```

LogoApplet.java (Part 1 of 3)

```
111 // Fig. 30.4: LogoApplet.java
112 // Customizing an applet via HTML parameters
113 //
114 // HTML parameter "animationdelay" is an int indicating
115 // milliseconds to sleep between images (default 50).
116 //
117 // HTML parameter "imagename" is the base name of the images
118 // that will be displayed (i.e., "deitel" is the base name
119 // for images "deitel0.gif," "deitel1.gif," etc.). The applet
120 // assumes that images are in an "images" subdirectory of
121 // the directory in which the applet resides.
122 //
```

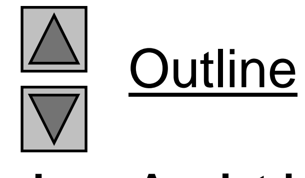
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
127 import java.awt.*;
128 import javax.swing.*;
129
130 public class LogoApplet extends JApplet{
131    public void init()
132    {
133       String parameter;
134
135       parameter = getParameter( "animationdelay" );
136       int animationDelay = ( parameter == null ? 50 :
137                              Integer.parseInt( parameter ) );
138
139       String imageName = getParameter( "imagename" );
140
141       parameter = getParameter( "totalimages" );
142       int totalImages = ( parameter == null ? 0 :
143                           Integer.parseInt( parameter ) );
144
145       // Create an instance of LogoAnimator
146       LogoAnimator animator;
147
148       if ( imageName == null || totalImages == 0 )
149          animator = new LogoAnimator();
```



LogoApplet.java (Part 2 of 3)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



[Outline](#)

LogoApplet.java (Part 3 of 3)

```
150         else
151            animator = new LogoAnimator( totalImages,
152                           animationDelay, imageName );
153
154         setSize( animator.getPreferredSize().width,
155                  animator.getPreferredSize().height );
156         getContentPane().add( animator, BorderLayout.CENTER );
157
158         animator.startAnimation();
159      } // end method init
160   } // end class LogoApplet
```

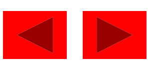
Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

30.7 Image Maps

- Image map
  - Image that has hot areas
    - User can click to accomplish a task
  - Bubble help
    - When mouse over particular point in screen, small message displayed in status bar
- In the following example
  - Load several images
  - Use event handler `mouseMoved` to find x-coordinate
  - Based on the x-coordinate, display a message

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

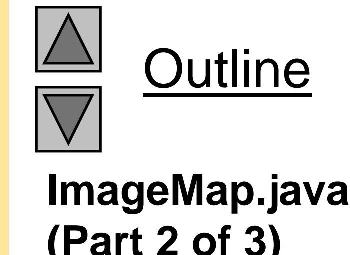


ImageMap.java (Part 1 of 3)

```
1  // Fig. 30.5: ImageMap.java
2  // Demonstrating an image map.
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class ImageMap extends JApplet {
8     private ImageIcon mapImage;
9     private int width, height;
10
11     public void init()
12     {
13        addMouseListener(
14           new MouseAdapter() {
15              public void mouseExited( MouseEvent e )
16              {
17                 showStatus( "Pointer outside applet" );
18              } // end method mouseExited
19           } // end anonymous inner class
20        ); // end addMouseListener
21
```

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

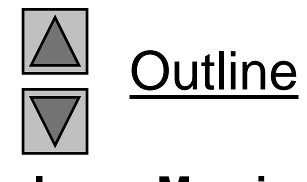
```
22       addMouseMotionListener(
23          new MouseMotionAdapter() {
24             public void mouseMoved( MouseEvent e )
25             {
26                showStatus( translateLocation( e.getX() ) );
27             } // end method mouseMoved
28          } // end anonymous inner class
29       ); // end addMouseMotionListener
30
31       mapImage = new ImageIcon( "icons2.gif" );
32       width = mapImage.getIconWidth();
33       height = mapImage.getIconHeight();
34       setSize( width, height );
35    } // end method init
36
37    public void paint( Graphics g )
38    {
39       mapImage.paintIcon( this, g, 0, 0 );
40    } // end method paint
41
42    public String translateLocation( int x )
43    {
44       // determine width of each icon (there are 6)
45       int iconWidth = width / 6;
46
```



ImageMap.java
(Part 2 of 3)

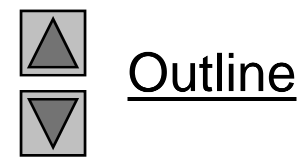
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
47       if ( x >= 0 && x <= iconWidth )
48          return "Common Programming Error";
49       else if ( x > iconWidth && x <= iconWidth * 2 )
50          return "Good Programming Practice";
51       else if ( x > iconWidth * 2 && x <= iconWidth * 3 )
52          return "Performance Tip";
53       else if ( x > iconWidth * 3 && x <= iconWidth * 4 )
54          return "Portability Tip";
55       else if ( x > iconWidth * 4 && x <= iconWidth * 5 )
56          return "Software Engineering Observation";
57       else if ( x > iconWidth * 5 && x <= iconWidth * 6 )
58          return "Testing and Debugging Tip";
59
60       return "";
61    } // end method translateLocation
62 } // end class ImageMap
```

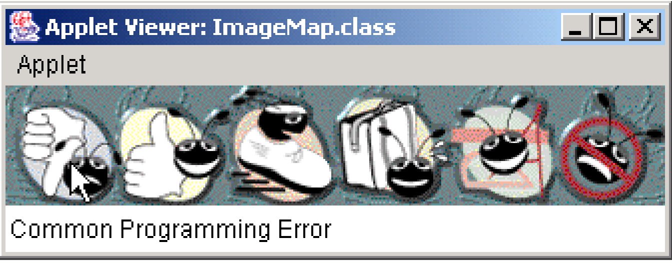


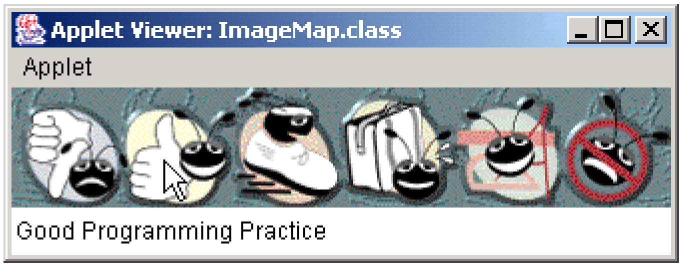
ImageMap.java (Part 3 of 3)

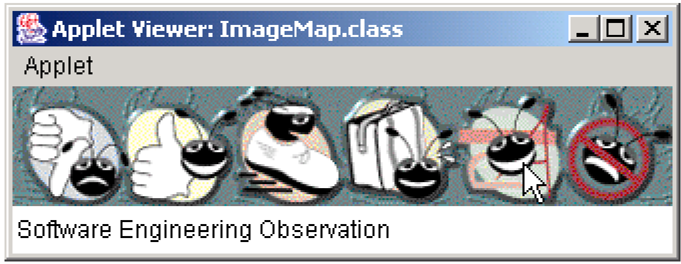
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

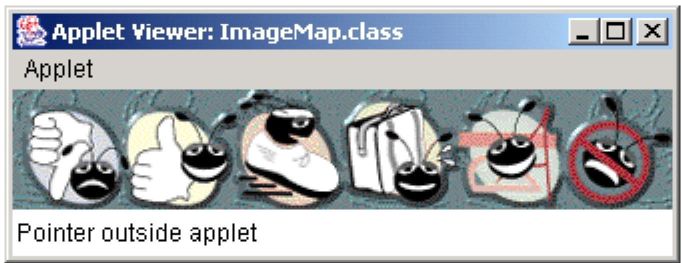


Program Output









© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

30.8 Internet and World Wide Web Resources

- Internet and web resources for multimedia related sites.

  `http://www.nasa.gov/gallery/index.html`

  - The *NASA multimedia gallery*

  `http://sunsite.sut.ac.jp/multimed/`

  - The *Sunsite Japan Multimedia Collection*

  `http://www.anbg.gov.au/anbg/index.html`

  - The *Australian National Botanic Gardens* Web site provides links to sounds of many animals.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.