

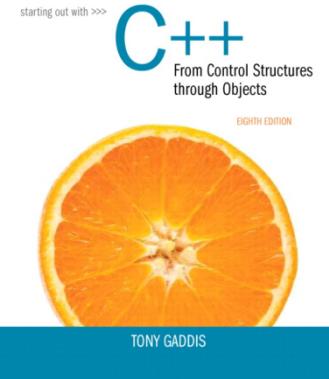
CPSC 5011: Object-Oriented Concepts

Lecture 0A: Basic C++
Gaddis, Chapters 2-7

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.



2.1

The Parts of a C++ Program

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

The Parts of a C++ Program

```
// sample C++ program ← comment
#include <iostream> ← preprocessor directive
using namespace std; ← which namespace to use
int main() ← beginning of function named main
{ ← beginning of block for main
    cout << "Hello, there!" ; ← output statement
    return 0; ← Send 0 to operating system
} ← end of block for main
```

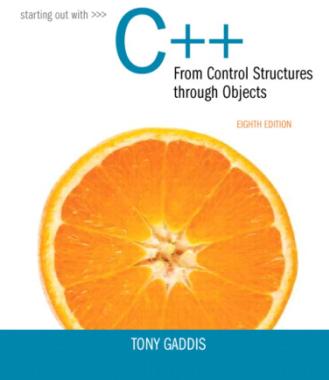
Special Characters

Character	Name	Meaning
//	Double slash	Beginning of a comment
#	Pound sign	Beginning of preprocessor directive
< >	Open/close brackets	Enclose filename in #include
()	Open/close parentheses	Used when naming a function
{ }	Open/close brace	Encloses a group of statements
" "	Open/close quotation marks	Encloses string of characters
;	Semicolon	End of a programming statement

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.



2.2

The cout Object

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

The cout Object

- Displays output on the computer screen
- You use the stream insertion operator << to send output to cout:

```
cout << "Programming is fun!";
```

The cout Object

- Can be used to send more than one item to cout:

```
cout << "Hello " << "there!";
```

Or:

```
cout << "Hello ";
cout << "there!";
```

The cout Object

- This produces one line of output:

```
cout << "Programming is ";  
cout << "fun!";
```

The endl Manipulator

- You can use the `endl` manipulator to start a new line of output. This will produce two lines of output:

```
cout << "Programming is" << endl;  
cout << "fun!";
```

The endl Manipulator

```
cout << "Programming is" << endl;  
cout << "fun!";
```



Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

The endl Manipulator

- You do NOT put quotation marks around `endl`
- The last character in `endl` is a lowercase L, not the number 1.

`endl` ← This is a lowercase L

The `\n` Escape Sequence

- You can also use the `\n` escape sequence to start a new line of output. This will produce two lines of output:

```
cout << "Programming is\n";  
cout << "fun!";
```

Notice that the `\n` is INSIDE
the string.

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

The \n Escape Sequence

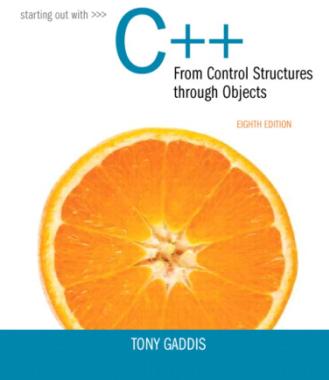
```
cout << "Programming is\n";
cout << "fun!";
```



Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.



2.3

The #include Directive

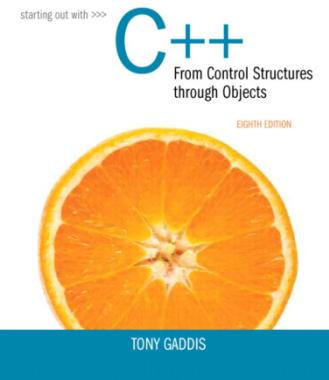
Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

The #include Directive

- Orange circle icon: Inserts the contents of another file into the program
- Orange circle icon: This is a preprocessor directive, not part of C++ language
- Orange circle icon: #include lines not seen by compiler
- Orange circle icon: Do not place a semicolon at end of #include line



2.4

Variables and Literals

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Variables and Literals

- Variable: a storage location in memory

- Has a name and a type of data it can hold
- Must be defined before it can be used:

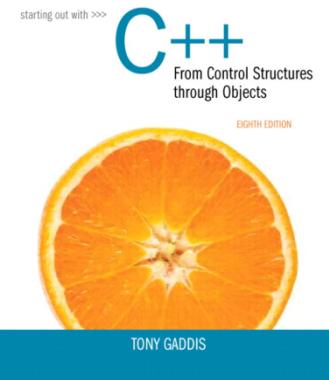
```
int item;
```

Literals

- **Literal:** a value that is written into a program's code.

"hello, there" **(string literal)**

12 **(integer literal)**



2.5

Identifiers

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Identifiers

- An identifier is a programmer-defined name for some part of a program: variables, functions, etc.

C++ Key Words

Table 2-4 The C++ Key Words

alignas	const	for	private	throw
alignof	constexpr	friend	protected	true
and	const_cast	goto	public	try
and_eq	continue	if	register	typedef
asm	decltype	inline	reinterpret_cast	typeid
auto	default	int	return	typename
bitand	delete	long	short	union
bitor	do	mutable	signed	unsigned
bool	double	namespace	sizeof	using
break	dynamic_cast	new	static	virtual
case	else	noexcept	static_assert	void
catch	enum	not	static_cast	volatile
char	explicit	not_eq	struct	wchar_t
char16_t	export	nullptr	switch	while
char32_t	extern	operator	template	xor
class	false	or	this	xor_eq
compl	float	or_eq	thread_local	

Addison-Wesley
is an imprint of

You cannot use any of the C++ key words as an identifier. These words have reserved meaning.



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Variable Names

- A variable name should represent the purpose of the variable. For example:

itemsOrdered

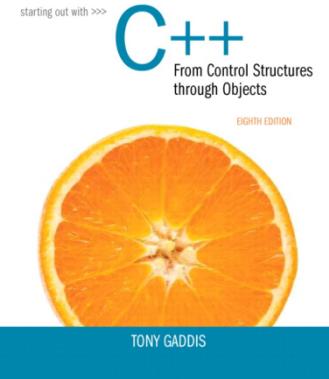
The purpose of this variable is to hold the number of items ordered.

Identifier Rules

- The first character of an identifier must be an alphabetic character or and underscore (_),
- After the first character you may use alphabetic characters, numbers, or underscore characters.
- Upper- and lowercase characters are distinct

Valid and Invalid Identifiers

IDENTIFIER	VALID?	REASON IF INVALID
totalSales	Yes	
total_Sales	Yes	
total.Sales	No	Cannot contain .
4thQtrSales	No	Cannot begin with digit
totalSale\$	No	Cannot contain \$



2.6

Integer Data Types

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Integer Data Types

- Integer variables can hold whole numbers such as 12, 7, and -99.

Table 2-6 Integer Data Types

Data Type	Typical Size	Typical Range
<code>short int</code>	2 bytes	-32,768 to +32,767
<code>unsigned short int</code>	2 bytes	0 to +65,535
<code>int</code>	4 bytes	-2,147,483,648 to +2,147,483,647
<code>unsigned int</code>	4 bytes	0 to 4,294,967,295
<code>long int</code>	4 bytes	-2,147,483,648 to +2,147,483,647
<code>unsigned long int</code>	4 bytes	0 to 4,294,967,295
<code>long long int</code>	8 bytes	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
<code>unsigned long long int</code>	8 bytes	0 to 18,446,744,073,709,551,615

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Defining Variables

- Variables of the same type can be defined
 - On separate lines:

```
int length;  
int width;  
unsigned int area;
```
 - On the same line:

```
int length, width;  
unsigned int area;
```
- Variables of different types must be in different definitions

Integer Literals

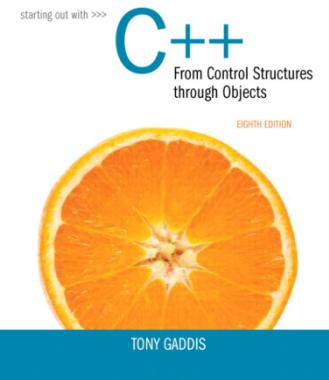
- An integer literal is an integer value that is typed into a program's code. For example:

```
itemsOrdered = 15;
```

In this code, 15 is an integer literal.

Integer Literals

- Integer literals are stored in memory as ints by default
- To store an integer constant in a long memory location, put ‘L’ at the end of the number: 1234L
- To store an integer constant in a long long memory location, put ‘LL’ at the end of the number: 324LL
- Constants that begin with ‘0’ (zero) are base 8: 075
- Constants that begin with ‘0x’ are base 16: 0x75A



2.7

The char Data Type

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

The `char` Data Type

- Used to hold characters or very small integer values
- Usually 1 byte of memory
- Numeric value of character from the character set is stored in memory:

CODE:

```
char letter;  
letter = 'C';
```

MEMORY:

letter

67

Character Literals

- Character literals must be enclosed in single quote marks. Example:

' A '

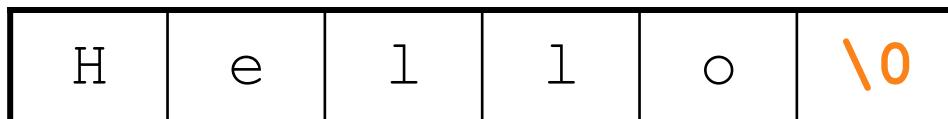
Character Strings

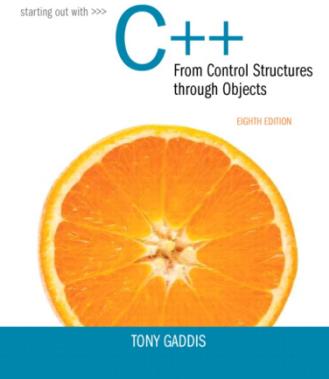
- A series of characters in consecutive memory locations:

"Hello"

- Stored with the null terminator, \0, at the end:

- Comprised of the characters between the " "





2.8

The C++ string Class

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

The C++ string Class

- Special data type supports working with strings

```
#include <string>
```

- Can define string **variables** in programs:

```
string firstName, lastName;
```

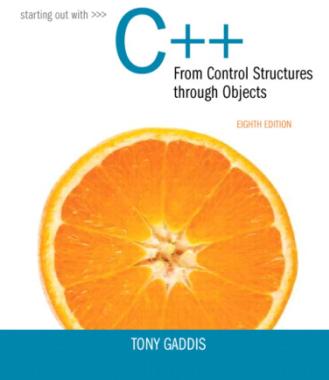
- Can receive values with assignment operator:

```
firstName = "George";
```

```
lastName = "Washington";
```

- Can be displayed via cout

```
cout << firstName << " " << lastName;
```



2.9

Floating-Point Data Types

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Floating-Point Data Types

- The floating-point data types are:

`float`

`double`

`long double`

- They can hold real numbers such as:

12.45

-3.8

- Stored in a form similar to scientific notation
- All floating-point numbers are signed

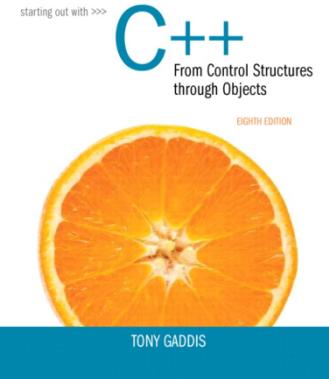
Floating-Point Data Types

Table 2-8 Floating Point Data Types on PCs

Data Type	Key Word	Description
Single precision	<code>float</code>	4 bytes. Numbers between $\pm 3.4\text{E-}38$ and $\pm 3.4\text{E}38$
Double precision	<code>double</code>	8 bytes. Numbers between $\pm 1.7\text{E-}308$ and $\pm 1.7\text{E}308$
Long double precision	<code>long double*</code>	8 bytes. Numbers between $\pm 1.7\text{E-}308$ and $\pm 1.7\text{E}308$

Floating-Point Literals

- Can be represented in
 - Fixed point (decimal) notation:
31.4159 0.0000625
 - E notation:
3.14159E1 6.25e-5
- Are double by default
- Can be forced to be float (3.14159f) or long double (0.0000625L)



2.10

The bool Data Type

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

The `bool` Data Type

- Represents values that are `true` or `false`
- `bool` variables are stored as small integers
- `false` is represented by 0, `true` by 1:

```
bool allDone = true;      allDone    finished
```

```
bool finished = false;
```

1

0

Boolean Variables in Program 2-17

Program 2-17

```
1 // This program demonstrates boolean variables.  
2 #include <iostream>  
3 using namespace std;  
4  
5 int main()  
6 {  
7     bool boolValue;  
8  
9     boolValue = true;  
10    cout << boolValue << endl;  
11    boolValue = false;  
12    cout << boolValue << endl;  
13    return 0;  
14 }
```

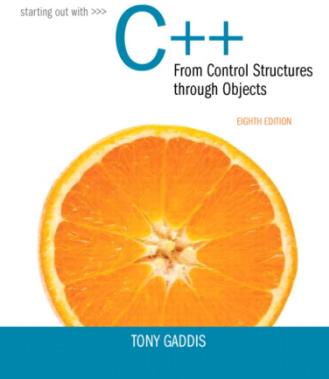
Program Output

```
1  
0
```

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.



2.12

Variable Assignments and Initialization

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Variable Assignments and Initialization

- An assignment statement uses the = operator to store a value in a variable.

```
item = 12;
```

- This statement assigns the value 12 to the item variable.

Assignment

- The variable receiving the value must appear on the left side of the = operator.
- This will NOT work:

```
// ERROR!  
12 = item;
```

Variable Initialization

- To initialize a variable means to assign it a value when it is defined:

```
int length = 12;
```

- Can initialize some or all variables:

```
int length = 12, width = 5, area;
```

Variable Initialization in Program 2-19

Program 2-19

```
1 // This program shows variable initialization.  
2 #include <iostream>  
3 using namespace std;  
4  
5 int main()  
6 {  
7     int month = 2, days = 28;  
8  
9     cout << "Month " << month << " has " << days << " days.\n";  
10    return 0;  
11 }
```

Program Output

Month 2 has 28 days.

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Declaring Variables With the `auto` Key Word

- C++ 11 introduces an alternative way to define variables, using the `auto` key word and an initialization value. Here is an example:

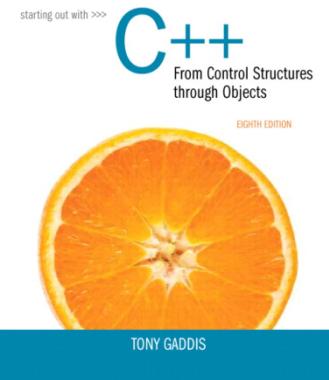
```
auto amount = 100;           ← int
```

- The `auto` key word tells the compiler to determine the variable's data type from the initialization value.

```
auto interestRate= 12.0;      ← double
```

```
auto stockCode = 'D';        ← char
```

```
auto customerNum = 459L;     ← long
```



2.13

Scope

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

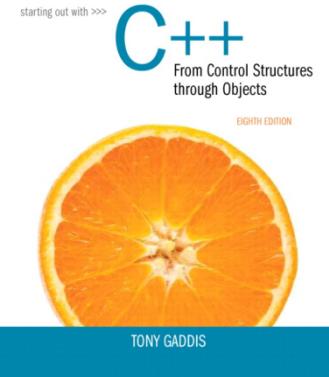
Scope

- The scope of a variable: the part of the program in which the variable can be accessed
- A variable cannot be used before it is defined

Variable Out of Scope in Program 2-20

Program 2-20

```
1 // This program can't find its variable.  
2 #include <iostream>  
3 using namespace std;  
4  
5 int main()  
6 {  
7     cout << value; // ERROR! value not defined yet!  
8  
9     int value = 100;  
10    return 0;  
11 }
```



2.14

Arithmetic Operators

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Arithmetic Operators

- Used for performing numeric calculations
- C++ has unary, binary, and ternary operators:
 - unary (1 operand) -5
 - binary (2 operands) $13 - 7$
 - ternary (3 operands) $\text{exp1} ? \text{exp2} : \text{exp3}$

Binary Arithmetic Operators

SYMBOL	OPERATION	EXAMPLE	VALUE OF ans
+	addition	ans = 7 + 3;	10
-	subtraction	ans = 7 - 3;	4
*	multiplication	ans = 7 * 3;	21
/	division	ans = 7 / 3;	2
%	modulus	ans = 7 % 3;	1

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

A Closer Look at the / Operator

- ➊ / (division) operator performs integer division if both operands are integers

```
cout << 13 / 5;      // displays 2
```

```
cout << 91 / 7;      // displays 13
```

- ➋ If either operand is floating point, the result is floating point

```
cout << 13 / 5.0;    // displays 2.6
```

```
cout << 91.0 / 7;    // displays 13.0
```

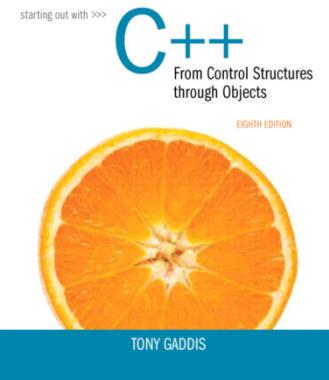
A Closer Look at the % Operator

- % (modulus) operator computes the remainder resulting from integer division

```
cout << 13 % 5; // displays 3
```

- % requires integers for both operands

```
cout << 13 % 5.0; // error
```



2.15

Comments

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Comments

- Used to document parts of the program
- Intended for persons reading the source code of the program:
 - Indicate the purpose of the program
 - Describe the use of variables
 - Explain complex sections of code
- Are ignored by the compiler

Single-Line Comments

- Begin with // through to the end of line:

```
int length = 12; // length in  
inches
```

```
int width = 15; // width in inches  
int area; // calculated area
```

```
// calculate rectangle area  
area = length * width;
```

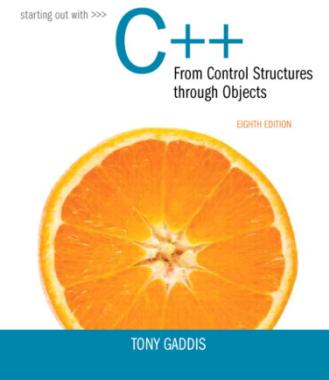
Multi-Line Comments

- Begin with `/*`, end with `*/`
- Can span multiple lines:

```
/* this is a multi-line  
comment  
*/
```

- Can begin and end on the same line:

```
int area; /* calculated area */
```



2.16

Named Constants

Addison-Wesley
is an imprint of



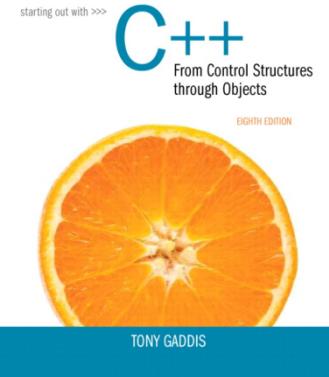
Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Named Constants

- Named constant (constant variable): variable whose content cannot be changed during program execution
- Used for representing constant values with descriptive names:

```
const double TAX_RATE = 0.0675;  
const int NUM_STATES = 50;
```

- Often named in uppercase letters



2.17

Programming Style

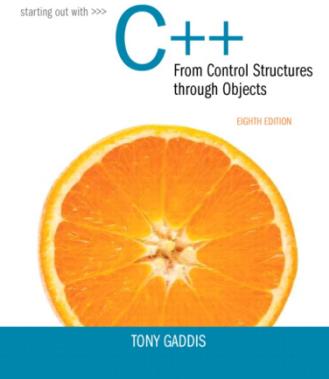
Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Programming Style

- The visual organization of the source code
- Includes the use of spaces, tabs, and blank lines
- Does not affect the syntax of the program
- Affects the readability of the source code



3.1

The `cin` Object

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

The `cin` Object

- Standard input object
- Like `cout`, requires `iostream` file
- Used to read input from keyboard
- Information retrieved from `cin` with `>>`
- Input is stored in one or more variables

The `cin` Object in Program 3-1

Program 3-1

```
1 // This program asks the user to enter the length and width of
2 // a rectangle. It calculates the rectangle's area and displays
3 // the value on the screen.
4 #include <iostream>
5 using namespace std;
6
7 int main()
8 {
9     int length, width, area;
10
11    cout << "This program calculates the area of a ";
12    cout << "rectangle.\n";
13    cout << "What is the length of the rectangle? ";
14    cin >> length;
15    cout << "What is the width of the rectangle? ";
16    cin >> width;
17    area = length * width;
18    cout << "The area of the rectangle is " << area << ".\n";
19
20 }
```

Program Output with Example Input Shown In Bold

This program calculates the area of a rectangle.
What is the length of the rectangle? **10 [Enter]**
What is the width of the rectangle? **20 [Enter]**
The area of the rectangle is 200.

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

The `cin` Object

- ➊ `cin` converts data to the type that matches the variable:

```
int height;  
cout << "How tall is the room? ";  
cin >> height;
```

Displaying a Prompt

- A prompt is a message that instructs the user to enter data.
- You should always use **cout** to display a prompt before each **cin** statement.

```
cout << "How tall is the room? ";
cin >> height;
```

The `cin` Object

- Can be used to input more than one value:

```
cin >> height >> width;
```

- Multiple values from keyboard must be separated by spaces
- Order is important: first value entered goes to first variable, etc.

The `cin` Object Gathers Multiple Values in Program 3-2

Program 3-2

```
1 // This program asks the user to enter the length and width of
2 // a rectangle. It calculates the rectangle's area and displays
3 // the value on the screen.
4 #include <iostream>
5 using namespace std;
6
7 int main()
8 {
9     int length, width, area;
10
11    cout << "This program calculates the area of a ";
12    cout << "rectangle.\n";
13    cout << "Enter the length and width of the rectangle ";
14    cout << "separated by a space.\n";
15    cin >> length >> width;
16    area = length * width;
17    cout << "The area of the rectangle is " << area << endl;
18    return 0;
19 }
```

Program Output with Example Input Shown In Bold

This program calculates the area of a rectangle.

Enter the length and width of the rectangle separated by a space.

10 20 [Enter]

The area of the rectangle is 200

Addison-Wesley
is an imprint of



The `cin` Object Reads Different Data Types in Program 3-3

Program 3-3

```
1 // This program demonstrates how cin can read multiple values
2 // of different data types.
3 #include <iostream>
4 using namespace std;
5
6 int main()
7 {
8     int whole;
9     double fractional;
10    char letter;
11
12    cout << "Enter an integer, a double, and a character: ";
13    cin >> whole >> fractional >> letter;
14    cout << "Whole: " << whole << endl;
15    cout << "Fractional: " << fractional << endl;
16    cout << "Letter: " << letter << endl;
17    return 0;
18 }
```

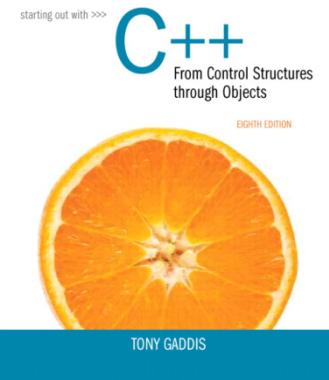
Program Output with Example Input Shown in Bold

Enter an integer, a double, and a character: **4 5.7 b** [Enter]

Whole: 4

Fractional: 5.7

Letter: b



3.2

Mathematical Expressions

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Mathematical Expressions

- Can create complex expressions using multiple mathematical operators
- An expression can be a literal, a variable, or a mathematical combination of constants and variables
- Can be used in assignment, cout, other statements:

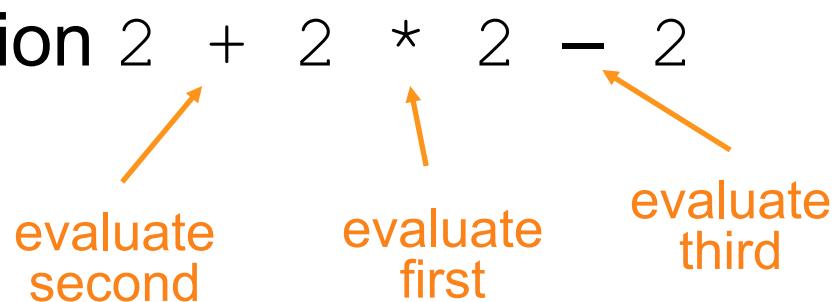
```
area = 2 * PI * radius;  
cout << "border is: " << 2*(l+w);
```

Order of Operations

In an expression with more than one operator, evaluate in this order:

- (unary negation), in order, left to right
- * / %, in order, left to right
- + -, in order, left to right

In the expression $2 + 2 * 2 - 2$



Order of Operations

Table 3-2 Some Simple Expressions and Their Values

Expression	Value
5 + 2 * 4	13
10 / 2 - 3	2
8 + 12 * 2 - 4	28
4 + 17 % 2 - 1	4
6 - 3 * 2 + 7 - 1	6

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Associativity of Operators

- $-$ (unary negation) associates right to left
- $\ast, /, \%, +, -$ associate right to left
- parentheses $()$ can be used to override the order of operations:

$$2 + 2 * 2 - 2 = 4$$

$$(2 + 2) * 2 - 2 = 6$$

$$2 + 2 * (2 - 2) = 2$$

$$(2 + 2) * (2 - 2) = 0$$

Grouping with Parentheses

Table 3-4 More Simple Expressions and Their Values

Expression	Value
(5 + 2) * 4	28
10 / (5 - 3)	5
8 + 12 * (6 - 2)	56
(4 + 17) % 2 - 1	0
(6 - 3) * (2 + 7) / 3	9

Algebraic Expressions

- ➊ Multiplication requires an operator:

$Area = lw$ is written as `Area = l * w;`

- ➋ There is no exponentiation operator:

$Area = s^2$ is written as `Area = pow(s, 2);`

- ➌ Parentheses may be needed to maintain order of operations:

$$m = \frac{y_2 - y_1}{x_2 - x_1}$$

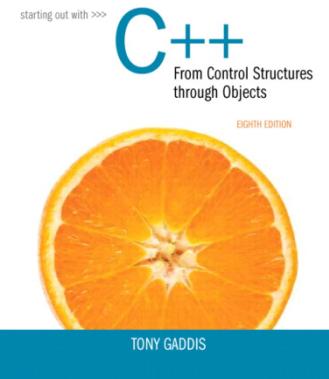
is written as

`m = (y2-y1) / (x2-x1);`

Algebraic Expressions

Table 3-5 Algebraic and C++ Multiplication Expressions

Algebraic Expression	Operation	C++ Equivalent
$6B$	6 times B	<code>6 * B</code>
$(3)(12)$	3 times 12	<code>3 * 12</code>
$4xy$	4 times x times y	<code>4 * x * y</code>



3.6

Multiple Assignment and Combined Assignment

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Multiple Assignment and Combined Assignment

- The = can be used to assign a value to multiple variables:

`x = y = z = 5;`

- Value of = is the value that is assigned
- Associates right to left:

`x = (y = (z = 5)) ;`

The diagram illustrates the associativity of the assignment operator. Three orange arrows originate from the text "value is 5" and point to the three equals signs in the expression. The first arrow points to the innermost equals sign, the second to the middle one, and the third to the outermost one.

Combined Assignment

- Look at the following statement:

```
sum = sum + 1;
```

This adds 1 to the variable **sum**.

Other Similar Statements

Table 3-8 (Assume $x = 6$)

Statement	What It Does	Value of x After the Statement
$x = x + 4;$	Adds 4 to x	10
$x = x - 3;$	Subtracts 3 from x	3
$x = x * 10;$	Multiplies x by 10	60
$x = x / 2;$	Divides x by 2	3
$x = x \% 4$	Makes x the remainder of $x / 4$	2

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Combined Assignment

- The combined assignment operators provide a shorthand for these types of statements.
- The statement

```
sum = sum + 1;
```

is equivalent to

```
sum += 1;
```

Combined Assignment Operators

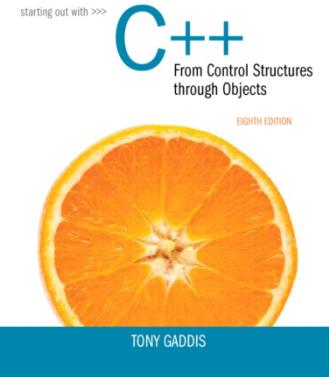
Table 3-9

Operator	Example Usage	Equivalent to
<code>+=</code>	<code>x += 5;</code>	<code>x = x + 5;</code>
<code>-=</code>	<code>y -= 2;</code>	<code>y = y - 2;</code>
<code>*=</code>	<code>z *= 10;</code>	<code>z = z * 10;</code>
<code>/=</code>	<code>a /= b;</code>	<code>a = a / b;</code>
<code>%=</code>	<code>c %= 3;</code>	<code>c = c % 3;</code>

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.



3.8

Working with Characters and string Objects

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Working with Characters and `string` Objects

- Using `cin` with the `>>` operator to input strings can cause problems:
- It passes over and ignores any leading *whitespace characters* (*spaces, tabs, or line breaks*)
- To work around this problem, you can use a C++ function named `getline`.

Using getline in Program 3-19

Program 3-19

```
1 // This program demonstrates using the getline function
2 // to read character data into a string object.
3 #include <iostream>
4 #include <string>
5 using namespace std;
6
7 int main()
8 {
9     string name;
10    string city;
11
12    cout << "Please enter your name: ";
13    getline(cin, name);
14    cout << "Enter the city you live in: ";
15    getline(cin, city);
16
17    cout << "Hello, " << name << endl;
18    cout << "You live in " << city << endl;
19    return 0;
20 }
```

Program Output with Example Input Shown in Bold

Please enter your name: **Kate Smith** [Enter]

Enter the city you live in: **Raleigh** [Enter]

Hello, Kate Smith

You live in Raleigh

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Working with Characters and **string** Objects

- To read a single character:

- Use `cin`:

```
char ch;  
cout << "Strike any key to continue";  
cin >> ch;
```

Problem: will skip over blanks, tabs, <CR>

- Use `cin.get()`:

```
cin.get(ch);
```

Will read the next character entered, even whitespace

Using `cin.get()` in Program 3-21

Program 3-21

```
1 // This program demonstrates three ways
2 // to use cin.get() to pause a program.
3 #include <iostream>
4 using namespace std;
5
6 int main()
7 {
8     char ch;
9
10    cout << "This program has paused. Press Enter to continue.";
11    cin.get(ch);
12    cout << "It has paused a second time. Please press Enter again.";
13    ch = cin.get();
14    cout << "It has paused a third time. Please press Enter again.";
15    cin.get();
16    cout << "Thank you!";
17    return 0;
18 }
```

Program Output with Example Input Shown in Bold

This program has paused. Press Enter to continue. **[Enter]**

It has paused a second time. Please press Enter again. **[Enter]**

It has paused a third time. Please press Enter again. **[Enter]**

Thank you!

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Working with Characters and *string* Objects

- Mixing `cin >>` and `cin.get()` in the same program can cause input errors that are hard to detect
- To skip over unneeded characters that are still in the keyboard buffer, use `cin.ignore()`:

```
cin.ignore(); // skip next char  
cin.ignore(10, '\n'); // skip the next  
// 10 char. or until a '\n'
```

string Member Functions and Operators

- To find the length of a string:

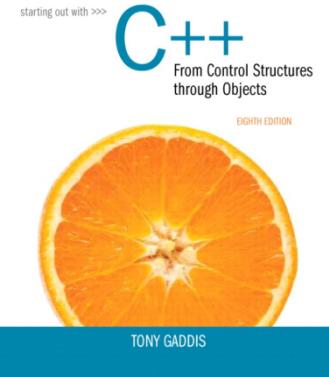
```
string state = "Texas";  
int size = state.length();
```

- To concatenate (join) multiple strings:

```
greeting2 = greeting1 + name1;  
greeting1 = greeting1 + name2;
```

Or using the `+=` combined assignment operator:

```
greeting1 += name2;
```



3.9

More Mathematical Library Functions

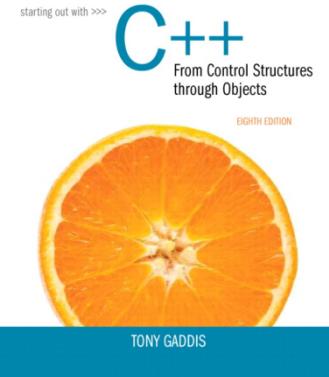
Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

More Mathematical Library Functions

- These require `cstdlib` header file
- `rand()` : returns a random number (`int`) between 0 and the largest int the computer holds. Yields same sequence of numbers each time program is run.
- `srand(x)` : initializes random number generator with `unsigned int x`



4.1

Relational Operators

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Relational Operators

- Used to compare numbers to determine relative order
- Operators:

>	Greater than
<	Less than
\geq	Greater than or equal to
\leq	Less than or equal to
$=$	Equal to
\neq	Not equal to

Relational Expressions

- Boolean expressions – true or false
- Examples:

`12 > 5` is true

`7 <= 5` is false

`if x is 10, then`

`x == 10` is true,

`x != 8` is true, and

`x == 8` is false

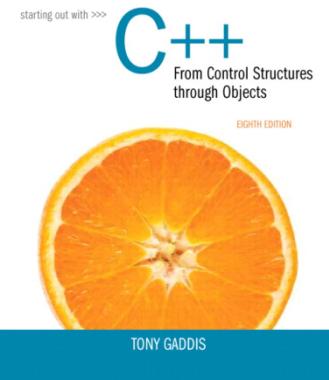
Relational Expressions

- Can be assigned to a variable:

```
result = x <= y;
```

- Assigns 0 for false, 1 for true

- Do not confuse = and ==



4.2

The if Statement

Addison-Wesley
is an imprint of

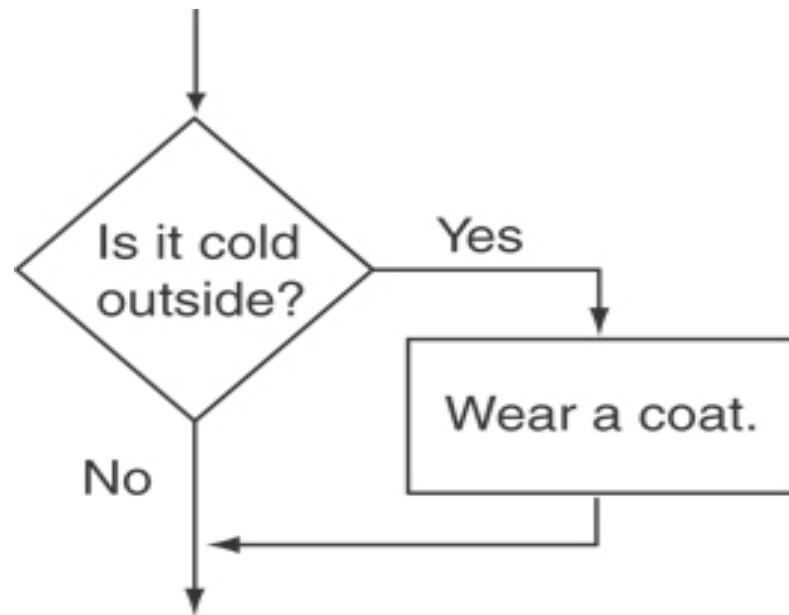


Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

The if Statement

- Allows statements to be conditionally executed or skipped over
- Models the way we mentally evaluate situations:
 - "If it is raining, take an umbrella."
 - "If it is cold outside, wear a coat."

Flowchart for Evaluating a Decision

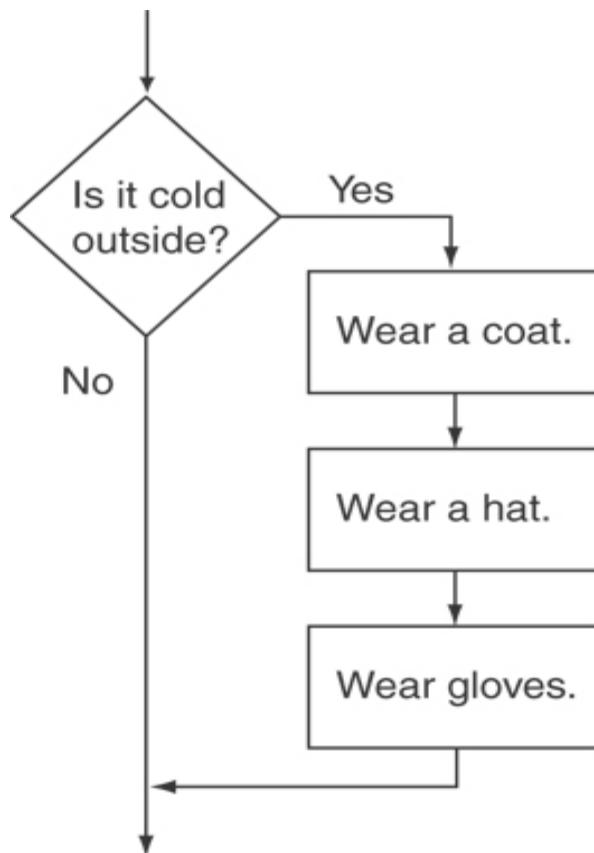


Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Flowchart for Evaluating a Decision



Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

The if Statement

- General Format:

```
if (expression)
    statement;
```

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

The if Statement-What Happens

To evaluate:

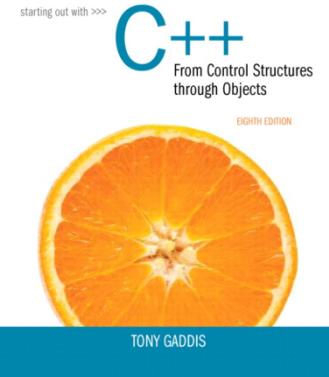
```
if (expression)
    statement;
```

- If the *expression* is true, then *statement* is executed.
- If the *expression* is false, then *statement* is skipped.

if Statement Notes

- Do not place ; after *(expression)*
- Place *statement;* on a separate line after *(expression)*, indented:

```
if (score > 90)
    grade = 'A';
```
- Be careful testing floats and doubles for equality
- 0 is false; any other value is true



4.3

Expanding the if Statement

Addison-Wesley
is an imprint of



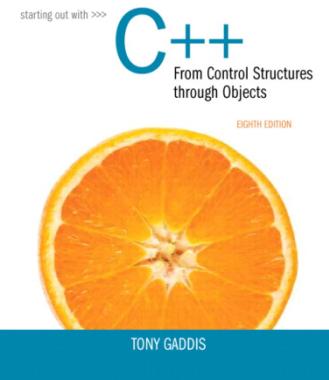
Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Expanding the if Statement

- To execute more than one statement as part of an if statement, enclose them in { }:

```
if (score > 90)
{
    grade = 'A';
    cout << "Good Job! \n";
}
```

- { } creates a block of code



4.4

The if/else Statement

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

The if/else statement

- Provides two possible paths of execution
- Performs one statement or block if the *expression* is true, otherwise performs another statement or block.

The if/else statement

General Format:

```
if (expression)
    statement1; // or block
else
    statement2; // or block
```

if/else-What Happens

To evaluate:

```
if (expression)
    statement1;
else
    statement2;
```

- If the *expression* is true, then *statement1* is executed and *statement2* is skipped.
- If the *expression* is false, then *statement1* is skipped and *statement2* is executed.

The `if/else` statement and Modulus Operator in Program 4-8

Program 4-8

```
1 // This program uses the modulus operator to determine
2 // if a number is odd or even. If the number is evenly divisible
3 // by 2, it is an even number. A remainder indicates it is odd.
4 #include <iostream>
5 using namespace std;
6
7 int main()
8 {
9     int number;
10
11    cout << "Enter an integer and I will tell you if it\n";
12    cout << "is odd or even. ";
13    cin >> number;
14    if (number % 2 == 0)
15        cout << number << " is even.\n";
16    else
17        cout << number << " is odd.\n";
18    return 0;
19 }
```

Program Output with Example Input Shown in Bold

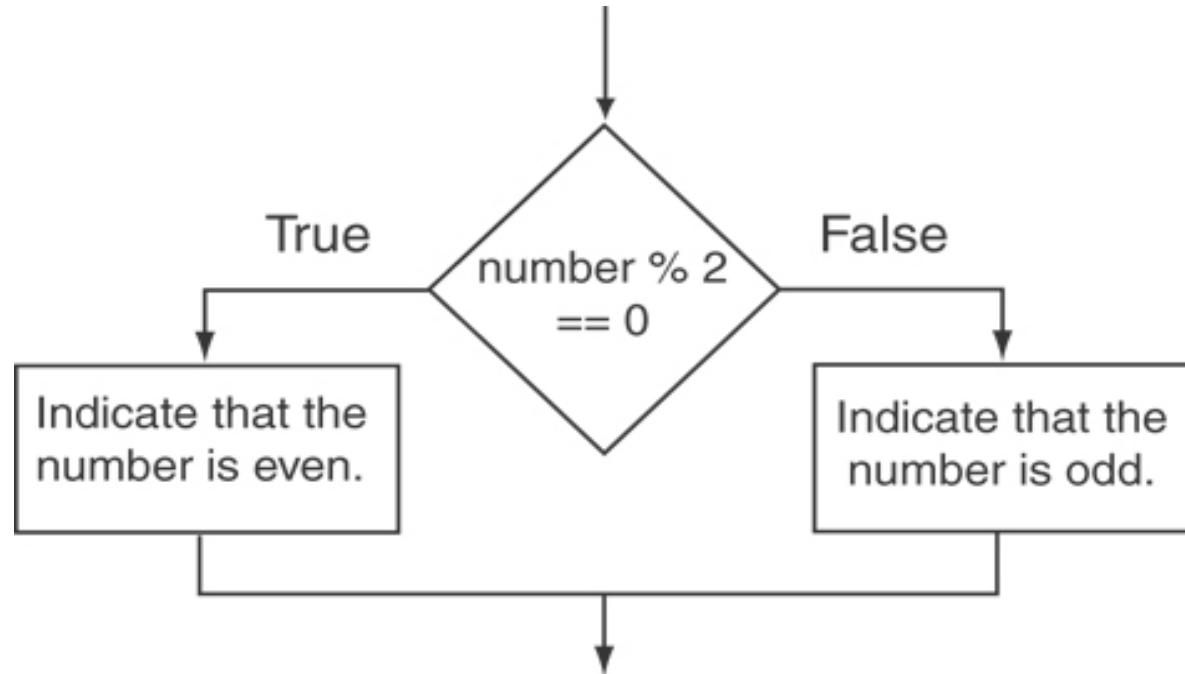
Enter an integer and I will tell you if it
is odd or even. **17 [Enter]**
17 is odd.

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Flowchart for Program 4-8 Lines 14 through 18



Testing the Divisor in Program 4-9

Program 4-9

```
1 // This program asks the user for two numbers, num1 and num2.  
2 // num1 is divided by num2 and the result is displayed.  
3 // Before the division operation, however, num2 is tested  
4 // for the value 0. If it contains 0, the division does not  
5 // take place.  
6 #include <iostream>  
7 using namespace std;  
8  
9 int main()  
10 {  
11     double num1, num2, quotient;  
12 }
```

Testing the Divisor in Program 4-9

Program 4-9 *(continued)*

```
13     // Get the first number.  
14     cout << "Enter a number: ";  
15     cin >> num1;  
16  
17     // Get the second number.  
18     cout << "Enter another number: ";  
19     cin >> num2;  
20  
21     // If num2 is not zero, perform the division.  
22     if (num2 == 0)  
23     {  
24         cout << "Division by zero is not possible.\n";  
25         cout << "Please run the program again and enter\n";  
26         cout << "a number other than zero.\n";  
27     }  
28     else  
29     {  
30         quotient = num1 / num2;  
31         cout << "The quotient of " << num1 << " divided by "  
32         cout << num2 << " is " << quotient << ".\n";  
33     }  
34     return 0;  
35 }
```

Program Output with Example Input Shown in Bold

(When the user enters 0 for num2)

Enter a number: **10 [Enter]**

Enter another number: **0 [Enter]**

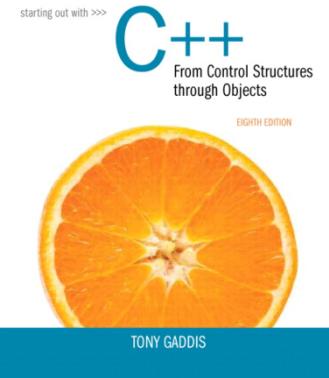
Division by zero is not possible.

Please run the program again and enter
a number other than zero.

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.



4.5

Nested if Statements

Addison-Wesley
is an imprint of

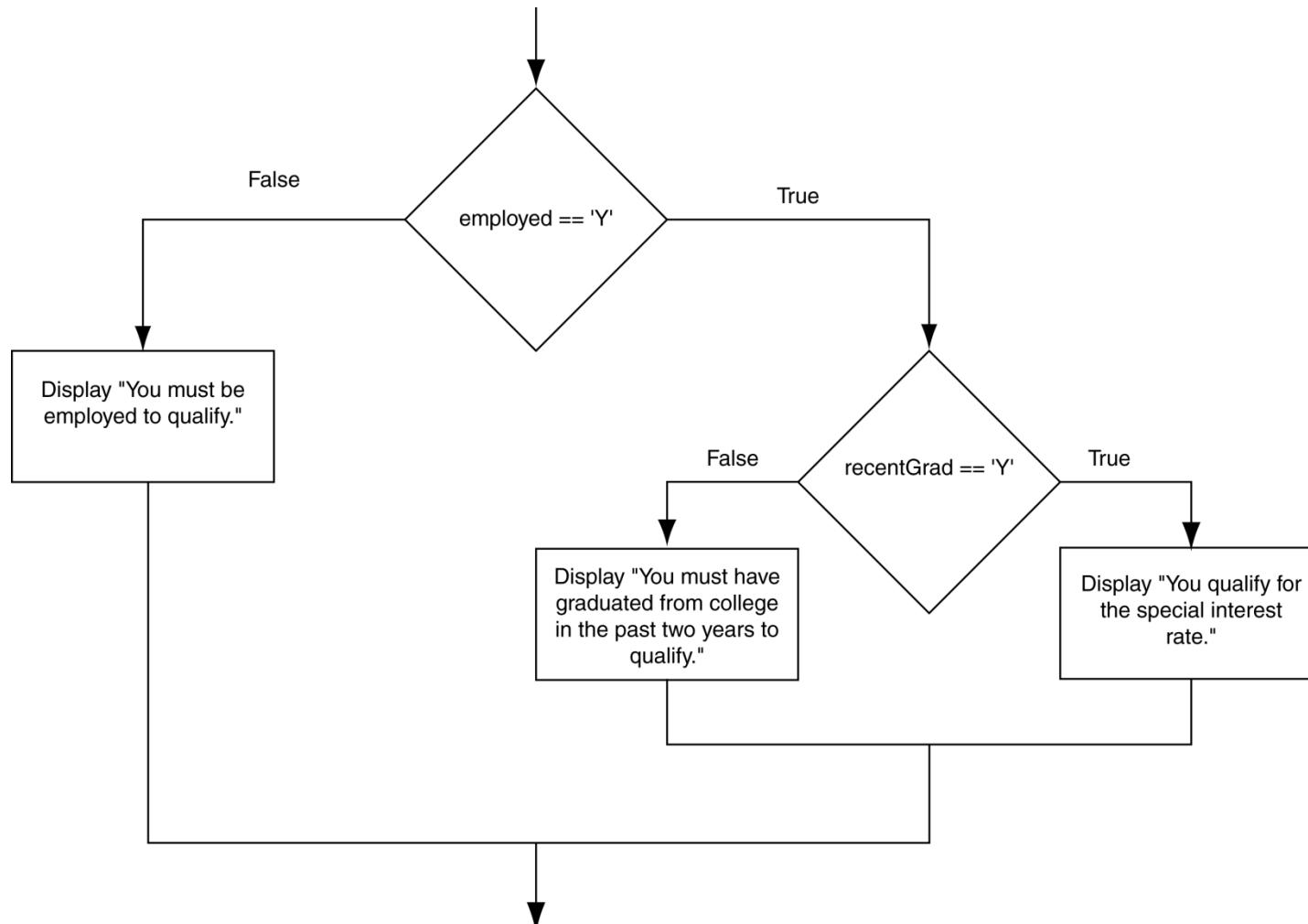


Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Nested if Statements

- An if statement that is nested inside another if statement
- Nested if statements can be used to test more than one condition

Flowchart for a Nested if Statement



Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Nested if Statements

From Program 4-10

```
20     // Determine the user's loan qualifications.  
21     if (employed == 'Y')  
22     {  
23         if (recentGrad == 'Y') //Nested if  
24         {  
25             cout << "You qualify for the special "  
26             cout << "interest rate.\n";  
27         }  
28     }
```

Nested if Statements

Another example, from Program 4-1

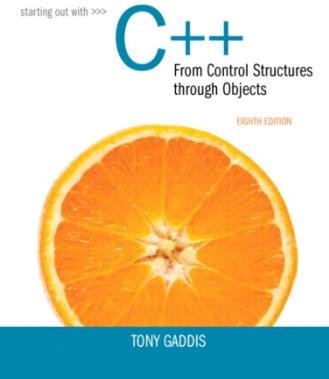
```
20     // Determine the user's loan qualifications.  
21     if (employed == 'Y')  
22     {  
23         if (recentGrad == 'Y') // Nested if  
24         {  
25             cout << "You qualify for the special ";  
26             cout << "interest rate.\n";  
27         }  
28     else // Not a recent grad, but employed  
29     {  
30         cout << "You must have graduated from ";  
31         cout << "college in the past two\n";  
32         cout << "years to qualify.\n";  
33     }  
34 }  
35 else // Not employed  
36 {  
37     cout << "You must be employed to qualify.\n";  
38 }
```

Use Proper Indentation!

```
if (employed == 'Y')
{
    if (recentGrad == 'Y') // Nested if
    {
        cout << "You qualify for the special ";
        cout << "interest rate.\n";
    }
    else // Not a recent grad, but employed
    {
        cout << "You must have graduated from ";
        cout << "college in the past two\n";
        cout << "years to qualify.\n";
    }
}
else // Not employed
{
    cout << "You must be employed to qualify.\n";
}
```

This if and else go together.

This if and else go together.



4.6

The if/else if Statement

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

The `if/else if` Statement

- Tests a series of conditions until one is found to be true
- Often simpler than using nested `if/else` statements
- Can be used to model thought processes such as:

"If it is raining, take an umbrella,
else, if it is windy, take a hat,
else, take sunglasses"

if/else if Format

```
if (expression)
    statement1; // or block
else if (expression)
    statement2; // or block
.
.
.
// other else ifs
.
.
.
else if (expression)
    statementn; // or block
```

The **if/else if** Statement in Program 4-13

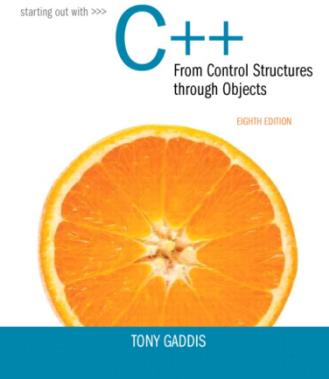
```
21     // Determine the letter grade.  
22     if (testScore >= A_SCORE)  
23         cout << "Your grade is A.\n";  
24     else if (testScore >= B_SCORE)  
25         cout << "Your grade is B.\n";  
26     else if (testScore >= C_SCORE)  
27         cout << "Your grade is C.\n";  
28     else if (testScore >= D_SCORE)  
29         cout << "Your grade is D.\n";  
30     else  
31         cout << "Your grade is F.\n";
```

Using a Trailing `else` to Catch Errors in Program 4-14

- The trailing `else` clause is optional, but it is best used to catch errors.

```
21     // Determine the letter grade.  
22     if (testScore >= A_SCORE)  
23         cout << "Your grade is A.\n";  
24     else if (testScore >= B_SCORE)  
25         cout << "Your grade is B.\n";  
26     else if (testScore >= C_SCORE)  
27         cout << "Your grade is C.\n";  
28     else if (testScore >= D_SCORE)  
29         cout << "Your grade is D.\n";  
30     else if (testScore >= 0)  
31         cout << "Your grade is F.\n";  
32     else  
33         cout << "Invalid test score.\n";
```

This trailing
else
catches
invalid test
scores



4.8

Logical Operators

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Logical Operators

- Used to create relational expressions from other relational expressions
- Operators, meaning, and explanation:

&&	AND	New relational expression is true if both expressions are true
	OR	New relational expression is true if either expression is true
!	NOT	Reverses the value of an expression – true expression becomes false, and false becomes true

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Logical Operators-Examples

```
int x = 12, y = 5, z = -4;
```

(x > y) && (y > z)	true
(x > y) && (z > y)	false
(x <= z) (y == z)	false
(x <= z) (y != z)	true
! (x >= z)	false

The logical && operator in Program 4-15

```
21 // Determine the user's loan qualifications.  
22 if (employed == 'Y' && recentGrad == 'Y')  
23 {  
24     cout << "You qualify for the special "  
25         << "interest rate.\n";  
26 }  
27 else  
28 {  
29     cout << "You must be employed and have\n"  
30         << "graduated from college in the\n"  
31         << "past two years to qualify.\n";  
32 }
```

The logical || Operator in Program 4-16

```
23     // Determine the user's loan qualifications.  
24     if (income >= MIN_INCOME || years > MIN_YEARS)  
25         cout << "You qualify.\n";  
26     else  
27     {  
28         cout << "You must earn at least $"  
29                     << MIN_INCOME << " or have been "  
30                     << "employed more than " << MIN_YEARS  
31                     << " years.\n";  
32     }
```

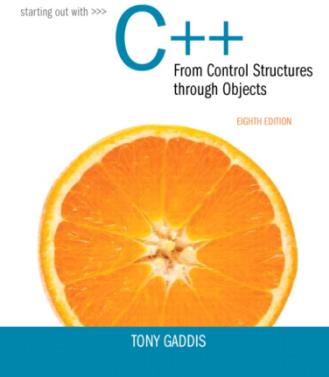
The logical ! Operator in Program

4-17

```
23 // Determine the user's loan qualifications.  
24 if (!(income >= MIN_INCOME || years > MIN_YEARS))  
25 {  
26     cout << "You must earn at least $"  
27             << MIN_INCOME << " or have been "  
28             << "employed more than " << MIN_YEARS  
29             << " years.\n";  
30 }  
31 else  
32     cout << "You qualify.\n";
```

Logical Operator-Notes

- ! has highest precedence, followed by &&, then |||
- If the value of an expression can be determined by evaluating just the sub-expression on left side of a logical operator, then the sub-expression on the right side will not be evaluated (*short circuit evaluation*)



4.9

Checking Numeric Ranges with Logical Operators

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Checking Numeric Ranges with Logical Operators

- Used to test to see if a value falls **inside** a range:

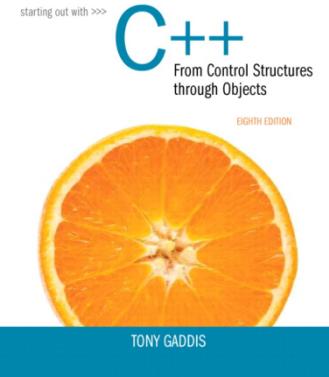
```
if (grade >= 0 && grade <= 100)  
    cout << "Valid grade";
```

- Can also test to see if value falls **outside** of range:

```
if (grade <= 0 || grade >= 100)  
    cout << "Invalid grade";
```

- Cannot use mathematical notation:

```
if (0 <= grade <= 100) //doesn't work!
```



4.12

Comparing Characters and Strings

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Comparing Characters

- Characters are compared using their ASCII values
- 'A' < 'B'
 - The ASCII value of 'A' (65) is less than the ASCII value of 'B'(66)
- '1' < '2'
 - The ASCII value of '1' (49) is less than the ASCII value of '2' (50)
- Lowercase letters have higher ASCII codes than uppercase letters, so 'a' > 'Z'

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Relational Operators Compare Characters in Program 4-20

```
10     // Get a character from the user.  
11     cout << "Enter a digit or a letter: ";  
12     ch = cin.get();  
13  
14     // Determine what the user entered.  
15     if (ch >= '0' && ch <= '9')  
16         cout << "You entered a digit.\n";  
17     else if (ch >= 'A' && ch <= 'Z')  
18         cout << "You entered an uppercase letter.\n";  
19     else if (ch >= 'a' && ch <= 'z')  
20         cout << "You entered a lowercase letter.\n";  
21     else  
22         cout << "That is not a digit or a letter.\n";
```

Comparing **string** Objects

- Like characters, strings are compared using their ASCII values

```
string name1 = "Mary";  
string name2 = "Mark";
```

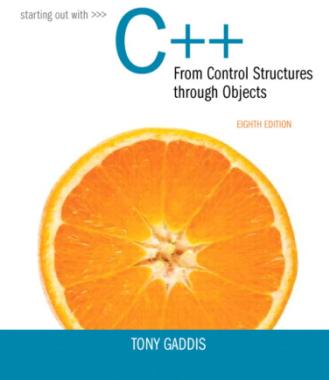
The characters in each string must match before they are equal

```
name1 > name2 // true  
name1 <= name2 // false  
name1 != name2 // true
```

```
name1 < "Mary Jane" // true
```

Relational Operators Compare Strings in Program 4-21

```
26 // Determine and display the correct price
27 if (partNum == "S-29A")
28     cout << "The price is $" << PRICE_A << endl;
29 else if (partNum == "S-29B")
30     cout << "The price is $" << PRICE_B << endl;
31 else
32     cout << partNum << " is not a valid part number.\n";
```



4.13

The Conditional Operator

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

The Conditional Operator

- Can use to create short if/else statements
- Format: expr ? expr : expr;

```
x<0 ? y=10 : z=20;
```

First Expression:
Expression to be tested

2nd Expression:
Executes if first expression is true

3rd Expression:
Executes if the first expression is false

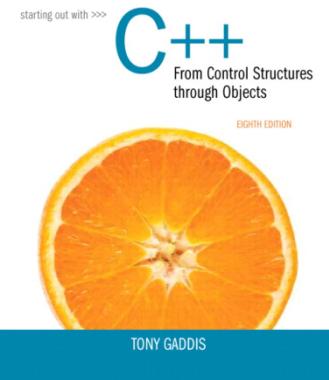
The diagram illustrates the structure of the conditional operator expression `x<0 ? y=10 : z=20;`. It features three orange arrows pointing from descriptive text below to specific parts of the code above. The first arrow points to the condition `x<0`, which is grouped by a brace under the text "First Expression: Expression to be tested". The second arrow points to the assignment `y=10`, which is grouped by a brace under the text "2nd Expression: Executes if first expression is true". The third arrow points to the assignment `z=20`, which is grouped by a brace under the text "3rd Expression: Executes if the first expression is false".

The Conditional Operator

- The value of a conditional expression is
 - The value of the second expression if the first expression is true
 - The value of the third expression if the first expression is false
- Parentheses () may be needed in an expression due to precedence of conditional operator

The Conditional Operator in Program 4-22

```
1 // This program calculates a consultant's charges at $50
2 // per hour, for a minimum of 5 hours. The ?: operator
3 // adjusts hours to 5 if less than 5 hours were worked.
4 #include <iostream>
5 #include <iomanip>
6 using namespace std;
7
8 int main()
9 {
10    const double PAY_RATE = 50.0;    // Hourly pay rate
11    const int MIN_HOURS = 5;        // Minimum billable hours
12    double hours,                 // Hours worked
13        charges;                  // Total charges
14
15    // Get the hours worked.
16    cout << "How many hours were worked? ";
17    cin >> hours;
18
19    // Determine the hours to charge for.
20    hours = hours < MIN_HOURS ? MIN_HOURS : hours;
21
22    // Calculate and display the charges.
23    charges = PAY_RATE * hours;
24    cout << fixed << showpoint << setprecision(2)
25        << "The charges are $" << charges << endl;
26    return 0;
27 }
```



4.14

The switch Statement

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

The `switch` Statement

- Used to select among statements from several alternatives
- In some cases, can be used instead of `if/else if` statements

switch Statement Format

```
switch (expression) //integer  
{  
    case exp1: statement1;  
    case exp2: statement2;  
    . . .  
    case expn: statementn;  
    default:     statementn+1;  
}
```

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

The **switch** Statement in Program 4-23

Program 4-23

```
1 // The switch statement in this program tells the user something
2 // he or she already knows: the data just entered!
3 #include <iostream>
4 using namespace std;
5
6 int main()
7 {
8     char choice;
9
10    cout << "Enter A, B, or C: ";
11    cin >> choice;
12    switch (choice)
13    {
14        case 'A': cout << "You entered A.\n";
15                    break;
16        case 'B': cout << "You entered B.\n";
17                    break;
18        case 'C': cout << "You entered C.\n";
19                    break;
20        default: cout << "You did not enter A, B, or C!\n";
21    }
22    return 0;
23 }
```

Program Output with Example Input Shown in Bold

Enter A, B, or C: **B** [Enter]
You entered B.

Program Output with Example Input Shown in Bold

Enter A, B, or C: **F** [Enter]
You did not enter A, B, or C!

switch Statement Requirements

- 1) *expression* must be an integer variable or an expression that evaluates to an integer value
- 2) *exp₁* through *exp_n* must be constant integer expressions or literals, and must be unique in the **switch** statement
- 3) default is optional but recommended

switch Statement-How it Works

- 1) *expression* is evaluated
- 2) The value of *expression* is compared against *exp₁* through *exp_n*.
- 3) If *expression* matches value *exp_i*, the program branches to the statement following *exp_i* and continues to the end of the switch
- 4) If no matching value is found, the program branches to the statement after default:

break Statement

- Used to exit a switch statement
- If it is left out, the program "falls through" the remaining statements in the switch statement

break and default statements in Program 4-25

Program 4-25

```
1 // This program is carefully constructed to use the "fall through"
2 // feature of the switch statement.
3 #include <iostream>
4 using namespace std;
5
6 int main()
7 {
8     int modelNum; // Model number
9
10    // Get a model number from the user.
11    cout << "Our TVs come in three models:\n";
12    cout << "The 100, 200, and 300. Which do you want? ";
13    cin >> modelNum;
14
15    // Display the model's features.
16    cout << "That model has the following features:\n";
17    switch (modelNum)
18    {
19        case 300: cout << "\tPicture-in-a-picture.\n";
20        case 200: cout << "\tStereo sound.\n";
21        case 100: cout << "\tRemote control.\n";
22            break;
23        default: cout << "You can only choose the 100,";
24            cout << "200, or 300.\n";
25    }
26    return 0;
27 }
```

Addison-Wesley
is an imprint of

Continued...



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

break and default statements in Program 4-25

Program Output with Example Input Shown in Bold

Our TVs come in three models:

The 100, 200, and 300. Which do you want? **100 [Enter]**

That model has the following features:

 Remote control.

Program Output with Example Input Shown in Bold

Our TVs come in three models:

The 100, 200, and 300. Which do you want? **200 [Enter]**

That model has the following features:

 Stereo sound.

 Remote control.

Program Output with Example Input Shown in Bold

Our TVs come in three models:

The 100, 200, and 300. Which do you want? **300 [Enter]**

That model has the following features:

 Picture-in-a-picture.

 Stereo sound.

 Remote control.

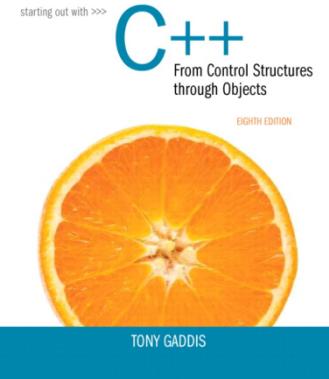
Program Output with Example Input Shown in Bold

Our TVs come in three models:

The 100, 200, and 300. Which do you want? **500 [Enter]**

That model has the following features:

You can only choose the 100, 200, or 300.



4.15

More About Blocks and Scope

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

More About Blocks and Scope

- Scope of a variable is the block in which it is defined, from the point of definition to the end of the block
- Usually defined at beginning of function
- May be defined close to first use

Inner Block Variable Definition in Program 4-29

```
16     if (income >= MIN_INCOME)
17     {
18         // Get the number of years at the current job.
19         cout << "How many years have you worked at "
20             << "your current job? ";
21         int years;      // Variable definition
22         cin >> years;
23
24         if (years > MIN_YEARS)
25             cout << "You qualify.\n";
26         else
27         {
28             cout << "You must have been employed for\n"
29                 << "more than " << MIN_YEARS
30                 << " years to qualify.\n";
31         }
32     }
```

Variables with the Same Name

- Variables defined inside { } have local or block scope
- When inside a block within another block, can define variables with the same name as in the outer block.
 - When in inner block, outer definition is not available
 - Not a good idea

Two Variables with the Same Name in Program 4-30

Program 4-30

```
1 // This program uses two variables with the name number.  
2 #include <iostream>  
3 using namespace std;  
4  
5 int main()  
6 {  
7     // Define a variable named number.  
8     int number;  
9  
10    cout << "Enter a number greater than 0: ";  
11    cin >> number;  
12    if (number > 0)  
13    {  
14        int number; // Another variable named number.  
15        cout << "Now enter another number: ";  
16        cin >> number;  
17        cout << "The second number you entered was "  
18            << number << endl;  
19    }  
20    cout << "Your first number was " << number << endl;  
21    return 0;  
22 }
```

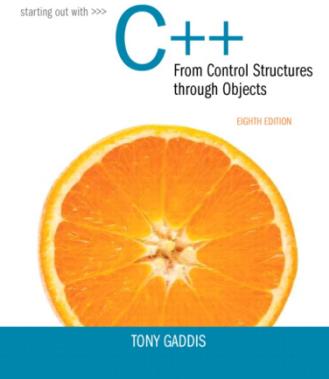
Program Output with Example Input Shown in Bold

Enter a number greater than 0: **2** [Enter]

Now enter another number: **7** [Enter]

The second number you entered was 7

Your first number was 2



5.1

The Increment and Decrement Operators

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

The Increment and Decrement Operators

- `++` is the increment operator.

It adds one to a variable.

`val++;` is the same as `val = val + 1;`

- `++` can be used before (prefix) or after (postfix) a variable:

`++val;` `val++;`

The Increment and Decrement Operators

- -- is the decrement operator.

It subtracts one from a variable.

`val--;` is the same as `val = val - 1;`

- -- can be also used before (prefix) or after (postfix) a variable:

`--val;` `val--;`

Increment and Decrement Operators in Program 5-1

Program 5-1

```
1 // This program demonstrates the ++ and -- operators.
2 #include <iostream>
3 using namespace std;
4
5 int main()
6 {
7     int num = 4;    // num starts out with 4.
8
9     // Display the value in num.
10    cout << "The variable num is " << num << endl;
11    cout << "I will now increment num.\n\n";
12
13    // Use postfix ++ to increment num.
14    num++;
15    cout << "Now the variable num is " << num << endl;
16    cout << "I will increment num again.\n\n";
17
18    // Use prefix ++ to increment num.
19    ++num;
20    cout << "Now the variable num is " << num << endl;
21    cout << "I will now decrement num.\n\n";
22
23    // Use postfix -- to decrement num.
24    num--;
25    cout << "Now the variable num is " << num << endl;
26    cout << "I will decrement num again.\n\n";
27
```

Addison-Wesley
is an imprint of

Continued...



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Increment and Decrement Operators in Program 5-1

Program 5-1 *(continued)*

```
28     // Use prefix -- to increment num.  
29     --num;  
30     cout << "Now the variable num is " << num << endl;  
31     return 0;  
32 }
```

Program Output

```
The variable num is 4  
I will now increment num.
```

```
Now the variable num is 5  
I will increment num again.
```

```
Now the variable num is 6  
I will now decrement num.
```

```
Now the variable num is 5  
I will decrement num again.
```

```
Now the variable num is 4
```

Prefix vs. Postfix

- `++` and `--` operators can be used in complex statements and expressions
- In prefix mode (`++val`, `--val`) the operator increments or decrements, *then* returns the value of the variable
- In postfix mode (`val++`, `val--`) the operator returns the value of the variable, *then* increments or decrements

Prefix vs. Postfix - Examples

```
int num, val = 12;  
cout << val++; // displays 12,  
                  // val is now 13;  
cout << ++val; // sets val to 14,  
                  // then displays it  
num = --val;    // sets val to 13,  
                  // stores 13 in num  
num = val--;   // stores 13 in num,  
                  // sets val to 12
```

Notes on Increment and Decrement

- Can be used in expressions:

```
result = num1++ + --num2;
```

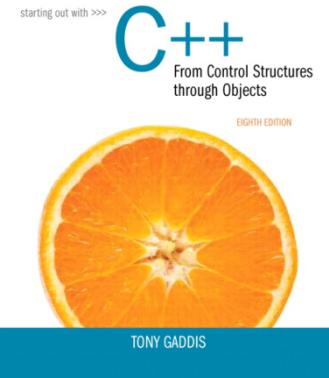
- Must be applied to something that has a location in memory. Cannot have:

```
result = (num1 + num2)++;
```

- Can be used in relational expressions:

```
if (++num > limit)
```

pre- and post-operations will cause different comparisons



5.2

Introduction to Loops: The `while` Loop

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Introduction to Loops: The while Loop

- Loop: a control structure that causes a statement or statements to repeat
- General format of the while loop:

while (*expression*)

statement;

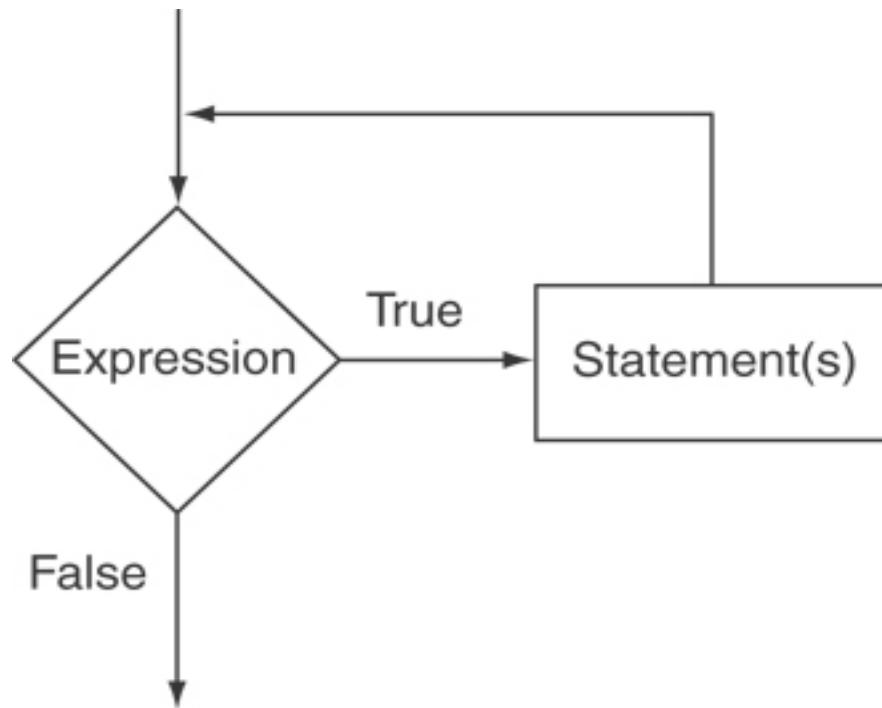
- *statement*; can also be a block of statements enclosed in { }

The `while` Loop – How It Works

```
while (expression)
      statement;
```

- *expression* is evaluated
 - if true, then *statement* is executed, and *expression* is evaluated again
 - if false, then the loop is finished and program statements following *statement* execute

The Logic of a while Loop



Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

The while loop in Program 5-3

Program 5-3

```
1 // This program demonstrates a simple while loop.  
2 #include <iostream>  
3 using namespace std;  
4  
5 int main()  
6 {  
7     int number = 1;  
8  
9     while (number <= 5)  
10    {  
11        cout << "Hello\n";  
12        number++;  
13    }  
14    cout << "That's all!\n";  
15    return 0;  
16 }
```

Program Output

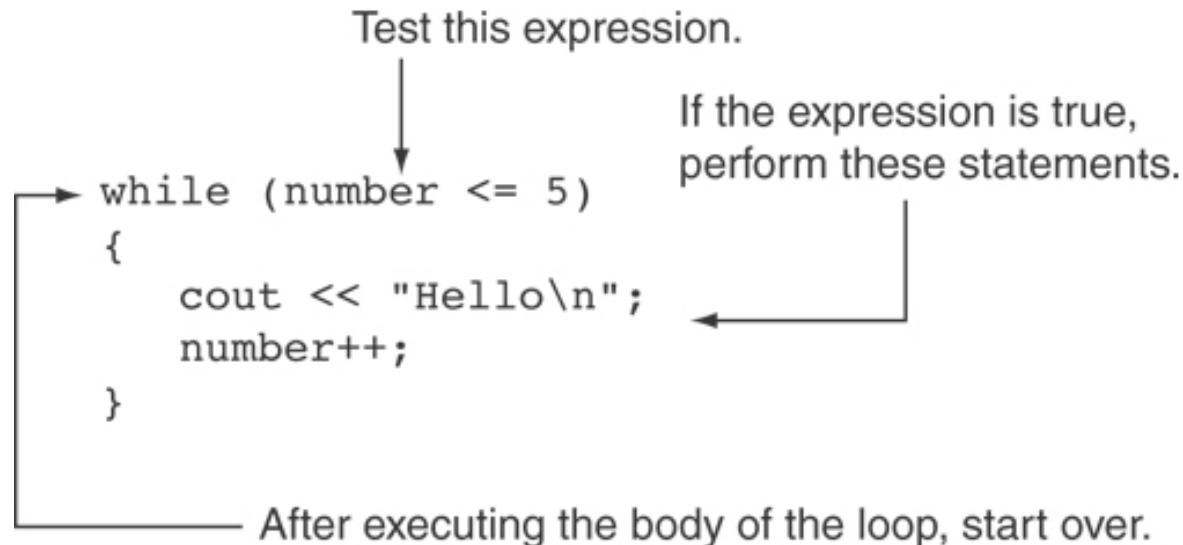
```
Hello  
Hello  
Hello  
Hello  
Hello  
That's all!
```

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

How the `while` Loop in Program 5-3 Lines 9 through 13 Works

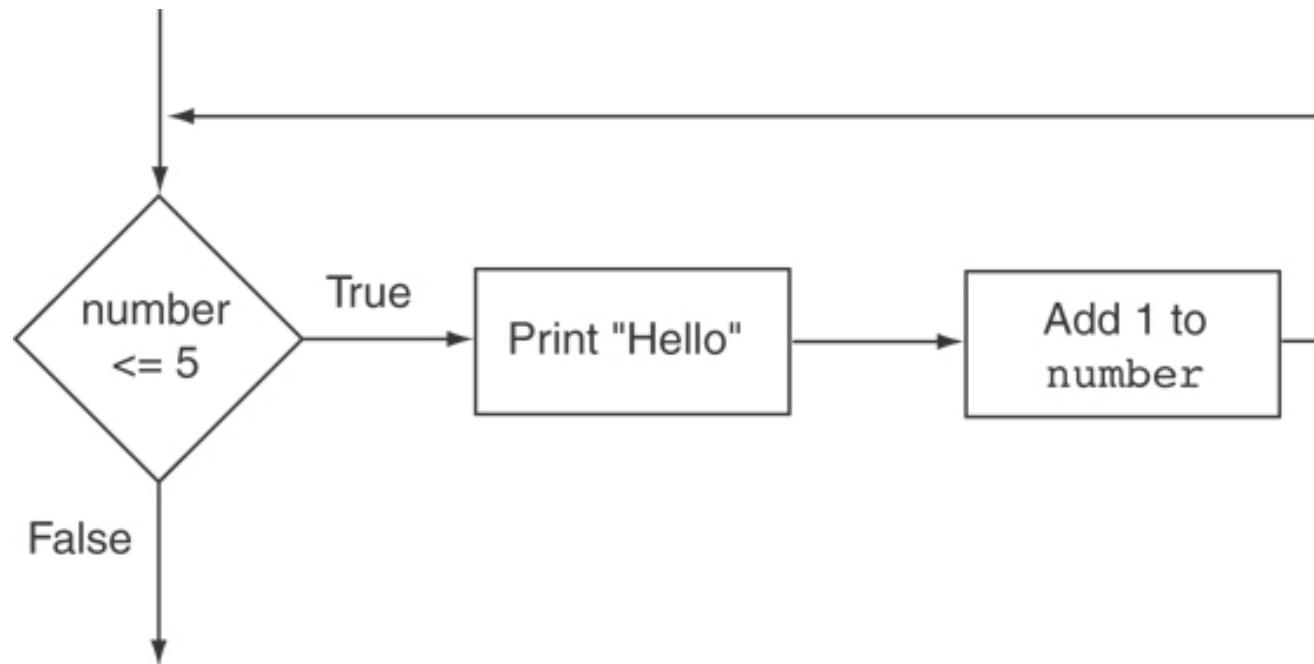


Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Flowchart of the while Loop in Program 5-3



The `while` Loop is a Pretest Loop

expression is evaluated *before* the loop executes. The following loop will never execute:

```
int number = 6;  
while (number <= 5)  
{  
    cout << "Hello\n";  
    number++;  
}
```

Addison-Wesley
is an imprint of



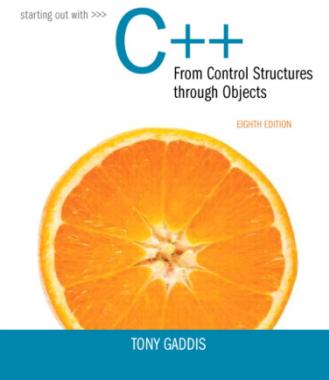
Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Watch Out for Infinite Loops

- The loop must contain code to make *expression* become false
- Otherwise, the loop will have no way of stopping
- Such a loop is called an *infinite loop*, because it will repeat an infinite number of times

Example of an Infinite Loop

```
int number = 1;  
while (number <= 5)  
{  
    cout << "Hello\n";  
}
```



5.3

Using the while Loop for Input Validation

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Using the `while` Loop for Input Validation

- Input validation is the process of inspecting data that is given to the program as input and determining whether it is valid.
- The while loop can be used to create input routines that reject invalid data, and repeat until valid data is entered.

Using the `while` Loop for Input Validation

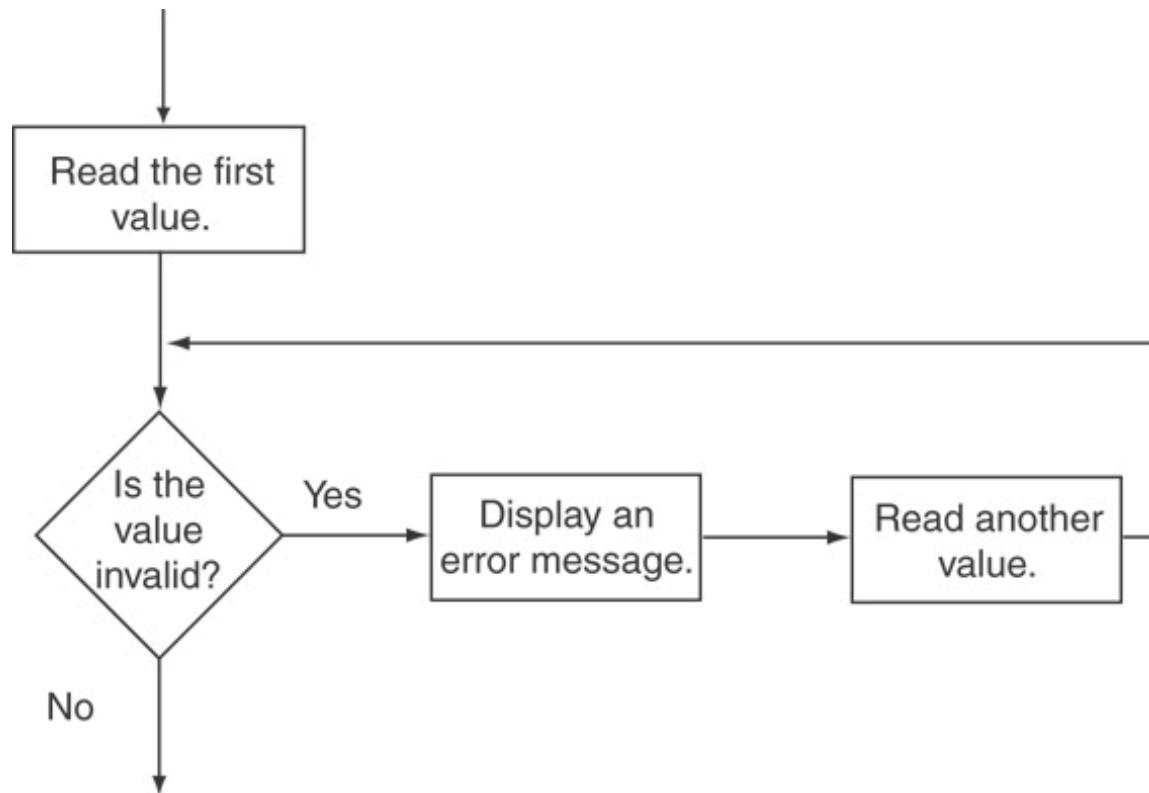
- Here's the general approach, in pseudocode:

Read an item of input.
While the input is invalid
 Display an error message.
 Read the input again.
End While

Input Validation Example

```
cout << "Enter a number less than 10: ";
cin >> number;
while (number >= 10)
{
    cout << "Invalid Entry!"
        << "Enter a number less than 10: ";
    cin >> number;
}
```

Flowchart for Input Validation



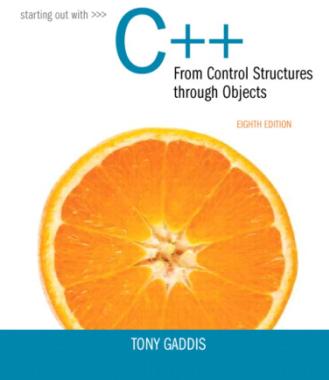
Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Input Validation in Program 5-5

```
20 // Get the number of players per team.  
21 cout << "How many players do you wish per team? ";  
22 cin >> teamPlayers;  
23  
24 // Validate the input.  
25 while (teamPlayers < MIN_PLAYERS || teamPlayers > MAX_PLAYERS)  
26 {  
27     // Explain the error.  
28     cout << "You should have at least " << MIN_PLAYERS  
29         << " but no more than " << MAX_PLAYERS << " per team.\n";  
30  
31     // Get the input again.  
32     cout << "How many players do you wish per team? ";  
33     cin >> teamPlayers;  
34 }  
35  
36 // Get the number of players available.  
37 cout << "How many players are available? ";  
38 cin >> players;  
39  
40 // Validate the input.  
41 while (players <= 0)  
42 {  
43     // Get the input again.  
44     cout << "Please enter 0 or greater: ";  
45     cin >> players;  
46 }
```



5.4

Counters

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Counters

- Counter: a variable that is incremented or decremented each time a loop repeats
- Can be used to control execution of the loop (also known as the loop control variable)
- Must be initialized before entering loop

A Counter Variable Controls the Loop in Program 5-6

Program 5-6

```
1 // This program displays a list of numbers and
2 // their squares.
3 #include <iostream>
4 using namespace std;
5
6 int main()
7 {
8     const int MIN_NUMBER = 1,      // Starting number to square
9                  MAX_NUMBER = 10;    // Maximum number to square
10
11    int num = MIN_NUMBER;        // Counter
12
13    cout << "Number Number Squared\n";
14    cout << "-----\n";
```

A Counter Variable Controls the Loop in Program 5-6

```
15     while (num <= MAX_NUMBER)
16     {
17         cout << num << "\t\t" << (num * num) << endl;
18         num++; //Increment the counter.
19     }
20     return 0;
21 }
```

Program Output

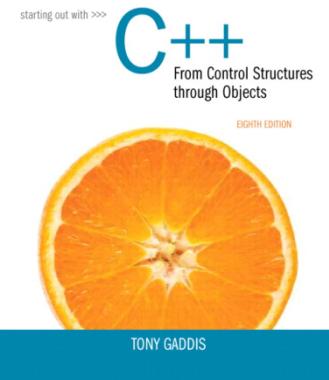
Number Number Squared

Number	Squared
1	1
2	4
3	9
4	16
5	25
6	36
7	49
8	64
9	81
10	100

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.



5.5

The do-while Loop

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

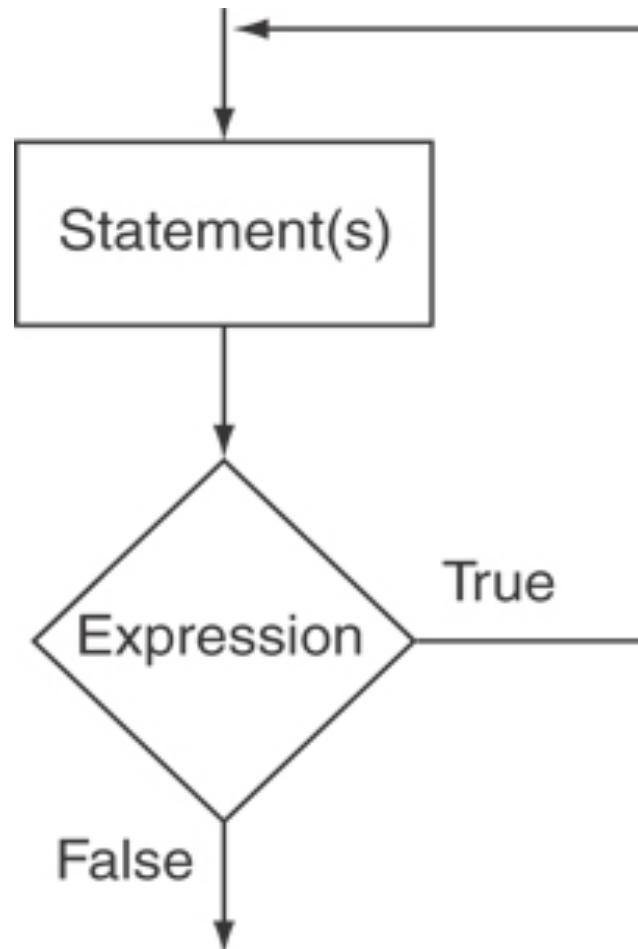
The do-while Loop

- do-while: a posttest loop – execute the loop, then test the expression
- General Format:

```
do  
    statement; // or block in { }  
    while (expression);
```

- Note that a semicolon is required after (expression)

The Logic of a do-while Loop



Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

An Example do-while Loop

```
int x = 1;  
do  
{  
    cout << x << endl;  
} while(x < 0);
```

Although the test expression is false, this loop will execute one time because do-while is a posttest loop.

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

A do-while Loop in Program 5-7

Program 5-7

```
1 // This program averages 3 test scores. It repeats as
2 // many times as the user wishes.
3 #include <iostream>
4 using namespace std;
5
6 int main()
7 {
8     int score1, score2, score3; // Three scores
9     double average;           // Average score
10    char again;              // To hold Y or N input
11
12    do
13    {
14        // Get three scores.
15        cout << "Enter 3 scores and I will average them: ";
16        cin >> score1 >> score2 >> score3;
17
18        // Calculate and display the average.
19        average = (score1 + score2 + score3) / 3.0;
20        cout << "The average is " << average << ".\n";
21
22        // Does the user want to average another set?
23        cout << "Do you want to average another set? (Y/N) ";
24        cin >> again;
25    } while (again == 'Y' || again == 'y');
26    return 0;
27 }
```

Addison-Wesley
is an imprint of

Continued...



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

A do-while Loop in Program 5-7

Program Output with Example Input Shown in Bold

Enter 3 scores and I will average them: **80 90 70 [Enter]**

The average is 80.

Do you want to average another set? (Y/N) **y [Enter]**

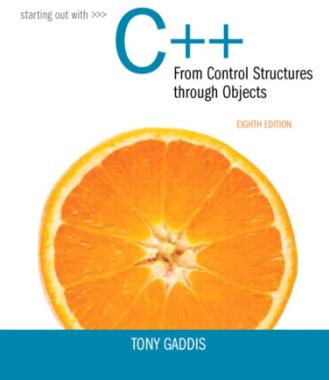
Enter 3 scores and I will average them: **60 75 88 [Enter]**

The average is 74.3333.

Do you want to average another set? (Y/N) **n [Enter]**

do-while Loop Notes

- Loop always executes at least once
- Execution continues as long as *expression* is true, stops repetition when *expression* becomes false
- Useful in menu-driven programs to bring user back to menu to make another choice (see Program 5-8 on pages 245-246)



5.6

The for Loop

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

The for Loop

- Useful for counter-controlled loop

- General Format:

```
for(initialization; test; update)  
    statement; // or block in { }
```

- No semicolon after the update expression or after the)

for Loop - Mechanics

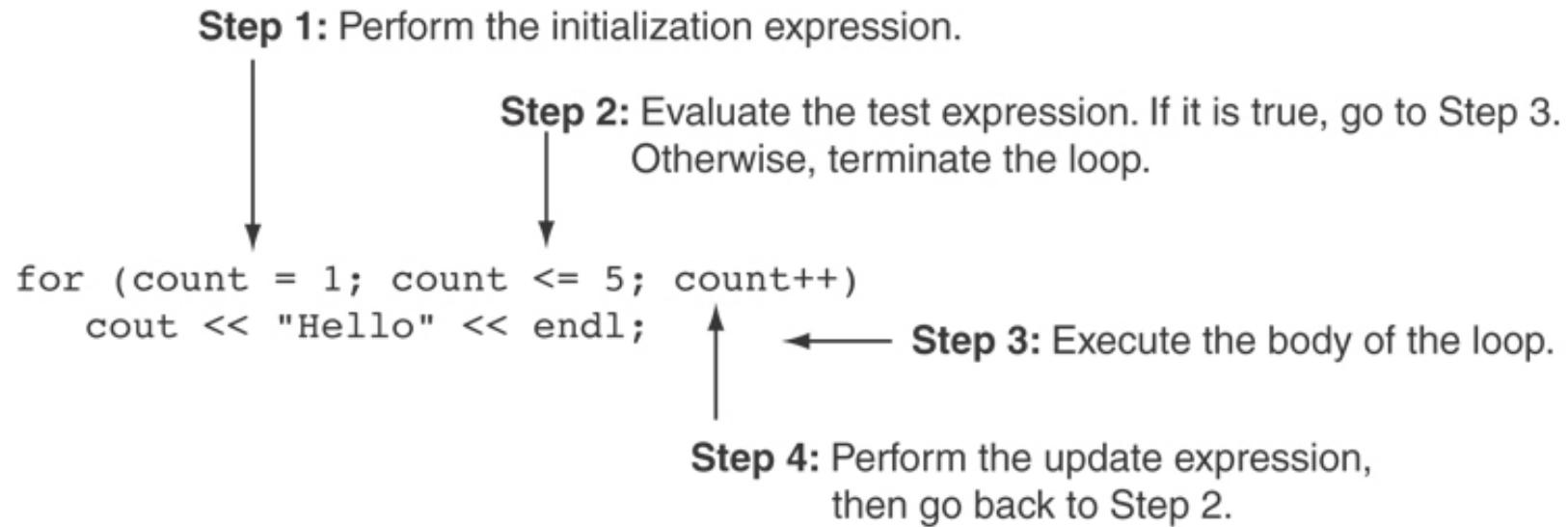
```
for(initialization; test; update)
    statement; // or block in { }
```

- 1) **Perform *initialization***
- 2) **Evaluate *test expression***
 - If true, **execute *statement***
 - If false, **terminate loop execution**
- 3) **Execute *update*, then re-evaluate *test expression***

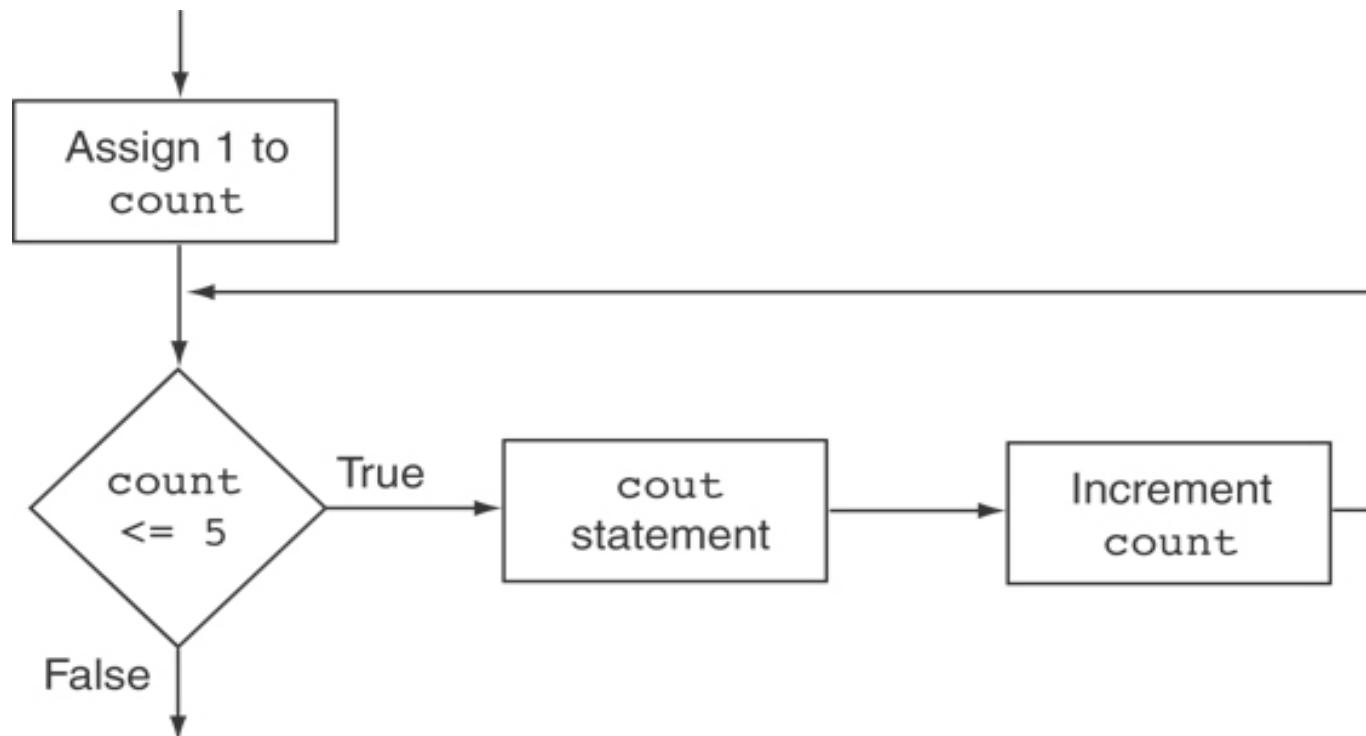
for Loop - Example

```
int count;  
  
for (count = 1; count <= 5; count++)  
    cout << "Hello" << endl;
```

A Closer Look at the Previous Example



Flowchart for the Previous Example



A for Loop in Program 5-9

Program 5-9

```
1 // This program displays the numbers 1 through 10 and
2 // their squares.
3 #include <iostream>
4 using namespace std;
5
6 int main()
7 {
8     const int MIN_NUMBER = 1,      // Starting value
9         MAX_NUMBER = 10;        // Ending value
10    int num;
11
12    cout << "Number Number Squared\n";
13    cout << "-----\n";
14
15    for (num = MIN_NUMBER; num <= MAX_NUMBER; num++)
16        cout << num << "\t\t" << (num * num) << endl;
17
18    return 0;
19 }
```

Addison-Wesley
is an imprint of

Continued...



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

A for Loop in Program 5-9

Program Output

Number	Number Squared
1	1
2	4
3	9
4	16
5	25
6	36
7	49
8	64
9	81
10	100

A Closer Look at Lines 15 through 16 in Program 5-9

Step 1: Perform the initialization expression.

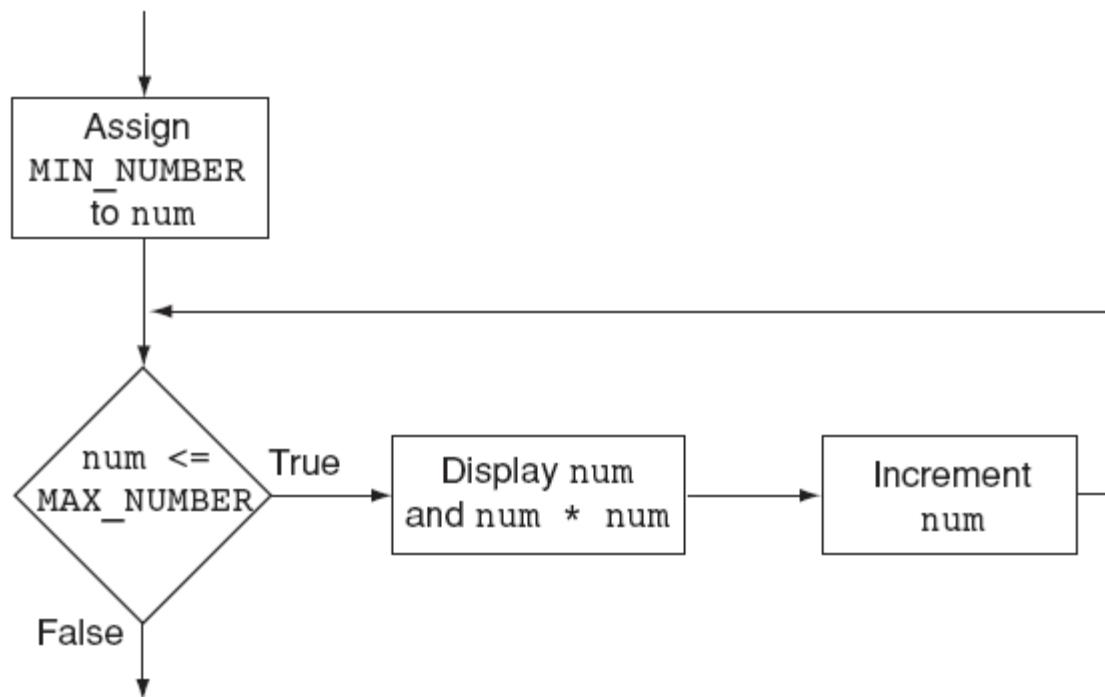
Step 2: Evaluate the test expression.
If it is true, go to Step 3.
Otherwise, terminate the loop.

Step 4: Perform the update expression, then go back to Step 2.

```
for (num = MIN_NUMBER; num <= MAX_NUMBER; num++)  
    cout << num << "\t\t" << (num * num) << endl;
```

Step 3: Execute the body of the loop.

Flowchart for Lines 15 through 16 in Program 5-9



When to Use the `for` Loop

- In any situation that clearly requires
 - an initialization
 - a false condition to stop the loop
 - an update to occur at the end of each iteration

The for Loop is a Pretest Loop

- The for loop tests its test expression before each iteration, so it is a pretest loop.
- The following loop will never iterate:

```
for (count = 11; count <= 10; count++)  
    cout << "Hello" << endl;
```

for Loop - Modifications

- You can have multiple statements in the *initialization* expression. Separate the statements with a comma:

```
Initialization Expression
int x, y;
for (x=1, y=1; x <= 5; x++)
{
    cout << x << " plus " << y
        << " equals " << (x+y)
        << endl;
}
```

for Loop - Modifications

- You can also have multiple statements in the *test expression*. Separate the statements with a comma:

```
int x, y;  
for (x=1, y=1; x <= 5; x++, y++)  
{  
    cout << x << " plus " << y  
        << " equals " << (x+y)  
        << endl;  
}
```

Test Expression



for Loop - Modifications

- You can omit the *initialization* expression if it has already been done:

```
int sum = 0, num = 1;  
for (; num <= 10; num++)  
    sum += num;
```

for Loop - Modifications

- You can declare variables in the *initialization expression*:

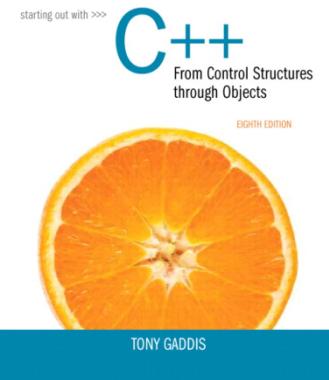
```
int sum = 0;  
for (int num = 0; num <= 10; num+  
+)  
    sum += num;
```

The scope of the variable num is the for loop.

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.



5.7

Keeping a Running Total

Addison-Wesley
is an imprint of



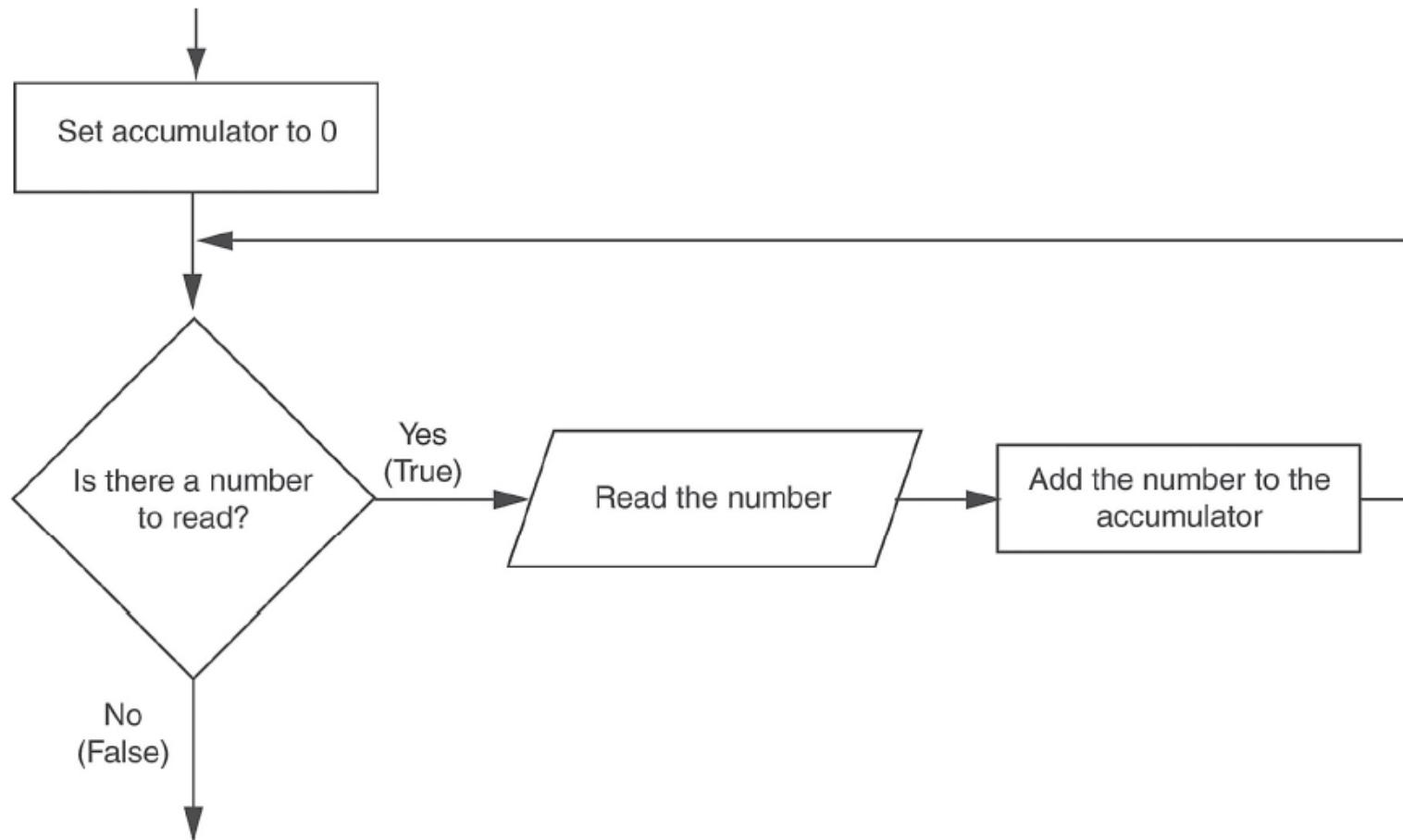
Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Keeping a Running Total

- running total: accumulated sum of numbers from each repetition of loop
- accumulator: variable that holds running total

```
int sum=0, num=1; // sum is the
while (num <= 10) // accumulator
{
    sum += num;
    num++;
}
cout << "Sum of numbers 1 - 10 is"
     << sum << endl;
```

Logic for Keeping a Running Total



Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

A Running Total in Program 5-12

Program 5-12

```
1 // This program takes daily sales figures over a period of time
2 // and calculates their total.
3 #include <iostream>
4 #include <iomanip>
5 using namespace std;
6
7 int main()
8 {
9     int days;           // Number of days
10    double total = 0.0; // Accumulator, initialized with 0
11
12    // Get the number of days.
13    cout << "For how many days do you have sales figures? ";
14    cin >> days;
15
16    // Get the sales for each day and accumulate a total.
17    for (int count = 1; count <= days; count++)
18    {
19        double sales;
20        cout << "Enter the sales for day " << count << ": ";
21        cin >> sales;
22        total += sales; // Accumulate the running total.
23    }
24
```

Addison-Wesley
is an imprint of

Continued...



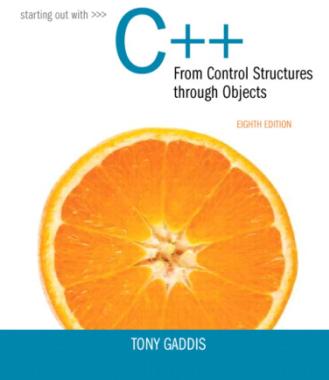
Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

A Running Total in Program 5-12

```
25     // Display the total sales.  
26     cout << fixed << showpoint << setprecision(2);  
27     cout << "The total sales are $" << total << endl;  
28     return 0;  
29 }
```

Program Output with Example Input Shown in Bold

```
For how many days do you have sales figures? 5 [Enter]  
Enter the sales for day 1: 489.32 [Enter]  
Enter the sales for day 2: 421.65 [Enter]  
Enter the sales for day 3: 497.89 [Enter]  
Enter the sales for day 4: 532.37 [Enter]  
Enter the sales for day 5: 506.92 [Enter]  
The total sales are $2448.15
```



5.8

Sentinels

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Sentinels

- sentinel: value in a list of values that indicates end of data
- Special value that cannot be confused with a valid value, e.g., -999 for a test score
- Used to terminate input when user may not know how many values will be entered

A Sentinel in Program 5-13

Program 5-13

```
1 // This program calculates the total number of points a
2 // soccer team has earned over a series of games. The user
3 // enters a series of point values, then -1 when finished.
4 #include <iostream>
5 using namespace std;
6
7 int main()
8 {
9     int game = 1,      // Game counter
10    points,        // To hold a number of points
11    total = 0;      // Accumulator
12
13    cout << "Enter the number of points your team has earned\n";
14    cout << "so far in the season, then enter -1 when finished.\n\n";
15    cout << "Enter the points for game " << game << ": ";
16    cin >> points;
17
18    while (points != -1)
19    {
20        total += points;
21        game++;
22        cout << "Enter the points for game " << game << ": ";
23        cin >> points;
24    }
25    cout << "\nThe total points are " << total << endl;
26    return 0;
27 }
```

Addison-Wesley
is an imprint of

Continued...



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

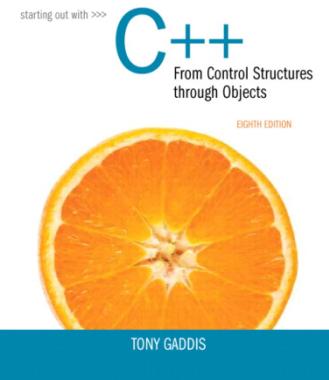
A Sentinel in Program 5-13

Program Output with Example Input Shown in Bold

Enter the number of points your team has earned so far in the season, then enter -1 when finished.

Enter the points for game 1: **7 [Enter]**
Enter the points for game 2: **9 [Enter]**
Enter the points for game 3: **4 [Enter]**
Enter the points for game 4: **6 [Enter]**
Enter the points for game 5: **8 [Enter]**
Enter the points for game 6: **-1 [Enter]**

The total points are 34



5.9

Deciding Which Loop to Use

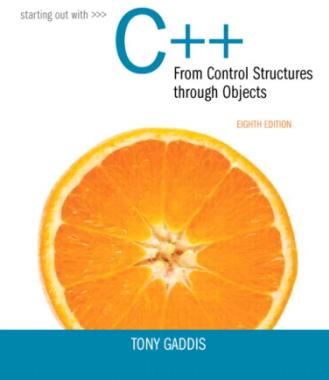
Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Deciding Which Loop to Use

- The `while` loop is a conditional pretest loop
 - Iterates as long as a certain condition exists
 - Validating input
 - Reading lists of data terminated by a sentinel
- The `do-while` loop is a conditional posttest loop
 - Always iterates at least once
 - Repeating a menu
- The `for` loop is a pretest loop
 - Built-in expressions for initializing, testing, and updating
 - Situations where the exact number of iterations is known



5.10

Nested Loops

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Nested Loops

- A nested loop is a loop inside the body of another loop
- Inner (inside), outer (outside) loops:

```
for (row=1; row<=3; row++) //outer  
  for (col=1; col<=3; col++) //inner  
    cout << row * col << endl;
```

Nested for Loop in Program 5-14

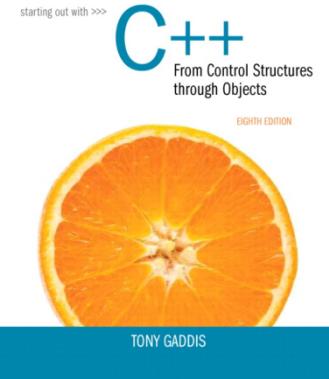
```
26 // Determine each student's average score.  
27 for (int student = 1; student <= numStudents; student++)  
28 {  
29     total = 0;          // Initialize the accumulator.  
30     for (int test = 1; test <= numTests; test++)  
31     {  
32         double score;  
33         cout << "Enter score " << test << " for ";  
34         cout << "student " << student << ": ";  
35         cin >> score;  
36         total += score;  
37     }  
38     average = total / numTests;  
39     cout << "The average score for student " << student;  
40     cout << " is " << average << ".\n\n";  
41 }
```

Inner Loop

Outer Loop

Nested Loops - Notes

- Inner loop goes through all repetitions for each repetition of outer loop
- Inner loop repetitions complete sooner than outer loop
- Total number of repetitions for inner loop is product of number of repetitions of the two loops.



5.11

Using Files for Data Storage

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Using Files for Data Storage

- Can use files instead of keyboard, monitor screen for program input, output
- Allows data to be retained between program runs
- Steps:
 - Open the file
 - Use the file (read from, write to, or both)
 - Close the file

Files: What is Needed

- Use `fstream` header file for file access
- File stream types:
 - `ifstream` for input from a file
 - `ofstream` for output to a file
 - `fstream` for input from or output to a file
- Define file stream objects:
 - `ifstream infile;`
 - `ofstream outfile;`

Opening Files

- Create a link between file name (outside the program) and file stream object (inside the program)

- Use the `open` member function:

```
infile.open("inventory.dat");  
outfile.open("report.txt");
```

- Filename may include drive, path info.
- Output file will be created if necessary; existing file will be erased first
- Input file must exist for `open` to work

Testing for File Open Errors

- Can test a file stream object to detect if an open operation failed:

```
infile.open("test.txt");
if (!infile)
{
    cout << "File open failure!";
}
```

- Can also use the `fail` member function

Using Files

- Can use output file object and << to send data to a file:

```
outfile << "Inventory report";
```

- Can use input file object and >> to copy data from file to variables:

```
infile >> partNum;
```

```
infile >> qtyInStock >>  
qtyOnOrder;
```

Using Loops to Process Files

- The stream extraction operator `>>` returns true when a value was successfully read, false otherwise
- Can be tested in a `while` loop to continue execution as long as values are read from the file:

```
while (inputFile >> number) ...
```

Closing Files

- Orange circle icon: Use the `close` member function:

```
infile.close();  
outfile.close();
```

- Orange circle icon: Don't wait for operating system to close files at program end:

- Orange circle icon: may be limit on number of open files
- Orange circle icon: may be buffered output data waiting to send to file

Letting the User Specify a Filename

- In many cases, you will want the user to specify the name of a file for the program to open.
- In C++ 11, you can pass a `string` object as an argument to a file stream object's `open` member function.

Letting the User Specify a Filename in Program 5-24

Program 5-24

```
1 // This program lets the user enter a filename.
2 #include <iostream>
3 #include <string>
4 #include <fstream>
5 using namespace std;
6
7 int main()
8 {
9     ifstream inputFile;
10    string filename;
11    int number;
12
13    // Get the filename from the user.
14    cout << "Enter the filename: ";
15    cin >> filename;
16
17    // Open the file.
18    inputFile.open(filename);
19
20    // If the file successfully opened, process it.
21    if (inputFile)
```

Letting the User Specify a Filename in Program 5-24

```
22     {
23         // Read the numbers from the file and
24         // display them.
25         while (inputFile >> number)
26         {
27             cout << number << endl;
28         }
29
30         // Close the file.
31         inputFile.close();
32     }
33     else
34     {
35         // Display an error message.
36         cout << "Error opening the file.\n";
37     }
38     return 0;
39 }
```

Program Output with Example Input Shown in Bold

```
Enter the filename: ListOfNumbers.txt [Enter]
100
200
300
400
500
600
700
```

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Using the `c_str` Member Function in Older Versions of C++

- Prior to C++ 11, the `open` member function requires that you pass the name of the file as a null-terminated string, which is also known as a C-string.
- *String literals are stored in memory as null-terminated C-strings, but string objects are not.*

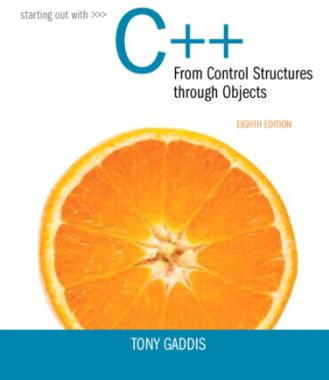
Using the `c_str` Member Function in Older Versions of C++

- `string` objects have a member function named `c_str`
 - It returns the contents of the object formatted as a null-terminated C-string.
 - Here is the general format of how you call the `c_str` function:

```
stringObject.c_str()
```

- Line 18 in Program 5-24 could be rewritten in the following manner:

```
inputFile.open(filename.c_str());
```



5.12

Breaking and Continuing a Loop

Addison-Wesley
is an imprint of



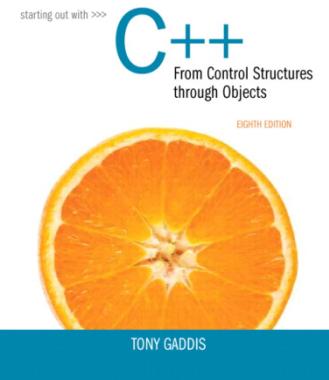
Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Breaking Out of a Loop

- Can use `break` to terminate execution of a loop
- Use sparingly if at all – makes code harder to understand and debug
- When used in an inner loop, terminates that loop only and goes back to outer loop

The continue Statement

- Can use `continue` to go to end of loop and prepare for next repetition
 - while, do-while loops: go to test, repeat loop if test passes
 - for loop: perform update step, then test, then repeat loop if test passes
- Use sparingly – like `break`, can make program logic hard to follow



6.1

Modular Programming

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Modular Programming

- Modular programming: breaking a program up into smaller, manageable functions or modules
- Function: a collection of statements to perform a task
- Motivation for modular programming:
 - Improves maintainability of programs
 - Simplifies the process of writing programs

This program has one long, complex function containing all of the statements necessary to solve a problem.



```
int main()
{
    statement;
    statement;
}
```

In this program the problem has been divided into smaller problems, each of which is handled by a separate function.

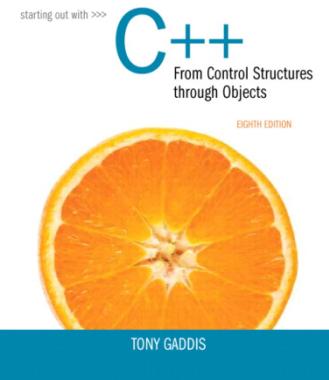


```
int main()
{
    statement;
    statement;
    statement;
}

void function2()
{
    statement;
    statement;
    statement;
}

void function3()
{
    statement;
    statement;
    statement;
}

void function4()
{
    statement;
    statement;
    statement;
}
```



6.2

Defining and Calling Functions

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Defining and Calling Functions

- Function call: statement causes a function to execute
- Function definition: statements that make up a function

Function Definition

- Definition includes:

- return type: data type of the value that function returns to the part of the program that called it
- name: name of the function. Function names follow same rules as variables
- parameter list: variables containing values passed to the function
- body: statements that perform the function's task, enclosed in { }

Function Definition

```
Return type      Parameter list (This one is empty)  
↓             ↓  
Function name   Function body  
int main ()  
{  
    cout << "Hello World\n";  
    return 0;  
}
```

Note: The line that reads `int main()` is the *function header*.

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Function Return Type

- If a function returns a value, the type of the value must be indicated:

```
int main()
```

- If a function does not return a value, its return type is `void`:

```
void printHeading()
{
    cout << "Monthly Sales\n";
}
```

Calling a Function

- To call a function, use the function name followed by () and ;

```
printHeading();
```

- When called, program executes the body of the called function
- After the function terminates, execution resumes in the calling function at point of call.

Functions in Program 6-1

Program 6-1

```
1 // This program has two functions: main and displayMessage
2 #include <iostream>
3 using namespace std;
4
5 //*****
6 // Definition of function displayMessage *
7 // This function displays a greeting. *
8 //*****
9
10 void displayMessage()
11 {
12     cout << "Hello from the function displayMessage.\n";
13 }
14
15 //*****
16 // Function main *
17 //*****
18
19 int main()
20 {
21     cout << "Hello from main.\n";
22     displayMessage();
23     cout << "Back in function main again.\n";
24     return 0;
25 }
```

Program Output

```
Hello from main.
Hello from the function displayMessage.
Back in function main again.
```

Addison-Wesley
is an imprint of

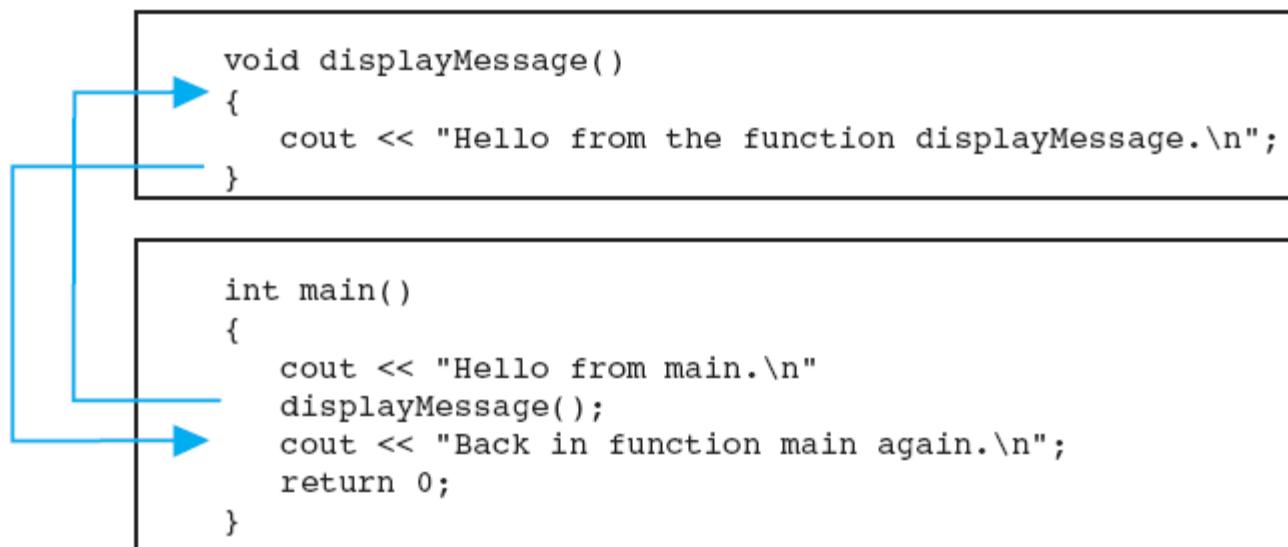


Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Flow of Control in Program 6-1

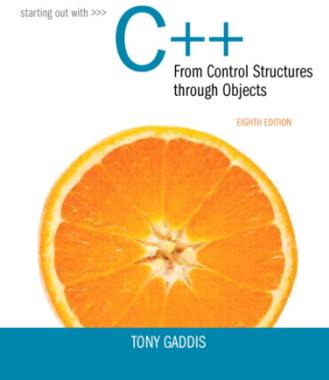
```
void displayMessage()
{
    cout << "Hello from the function displayMessage.\n";
}

int main()
{
    cout << "Hello from main.\n"
    displayMessage();
    cout << "Back in function main again.\n";
    return 0;
}
```



Calling Functions

- `main` can call any number of functions
- Functions can call other functions
- Compiler must know the following about a function before it is called:
 - name
 - return type
 - number of parameters
 - data type of each parameter



6.3

Function Prototypes

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Function Prototypes

- Ways to notify the compiler about a function before a call to the function:
 - Place function definition before calling function's definition
 - Use a function prototype (function declaration) – like the function definition without the body
 - Header: void printHeading()
 - Prototype: void printHeading();

Function Prototypes in Program 6-5

Program 6-5

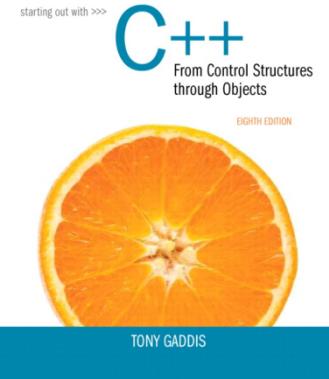
```
1 // This program has three functions: main, First, and Second.  
2 #include <iostream>  
3 using namespace std;  
4  
5 // Function Prototypes  
6 void first();  
7 void second();  
8  
9 int main()  
10 {  
11     cout << "I am starting in function main.\n";  
12     first();    // Call function first  
13     second();   // Call function second  
14     cout << "Back in function main again.\n";  
15     return 0;  
16 }  
17
```

Function Prototypes in Program 6-5

```
18 //*****  
19 // Definition of function first.      *  
20 // This function displays a message.  *  
21 //*****  
22  
23 void first()  
24 {  
25     cout << "I am now inside the function first.\n";  
26 }  
27  
28 //*****  
29 // Definition of function second.      *  
30 // This function displays a message.  *  
31 //*****  
32  
33 void second()  
34 {  
35     cout << "I am now inside the function second.\n";  
36 }
```

Prototype Notes

- Place prototypes near top of program
- Program must include either prototype or full function definition before any call to the function – compiler error otherwise
- When using prototypes, can place function definitions in any order in source file



6.4

Sending Data into a Function

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Sending Data into a Function

- Can pass values into a function at time of call:

```
c = pow(a, b);
```

- Values passed to function are arguments

- Variables in a function that hold the values passed as arguments are parameters

A Function with a Parameter Variable

```
void displayValue(int num)  
{  
    cout << "The value is " << num << endl;  
}
```

The integer variable `num` is a parameter.
It accepts any integer value passed to the function.

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Function with a Parameter in Program 6-6

Program 6-6

```
1 // This program demonstrates a function with a parameter.  
2 #include <iostream>  
3 using namespace std;  
4  
5 // Function Prototype  
6 void displayValue(int);  
7  
8 int main()  
9 {  
10    cout << "I am passing 5 to displayValue.\n";  
11    displayValue(5); // Call displayValue with argument 5  
12    cout << "Now I am back in main.\n";  
13    return 0;  
14 }  
15
```

Function with a Parameter in Program 6-6

Program 6-6

(continued)

```
16 //*****  
17 // Definition of function displayValue. *  
18 // It uses an integer parameter whose value is displayed. *  
19 //*****  
20  
21 void displayValue(int num)  
22 {  
23     cout << "The value is " << num << endl;  
24 }
```

Program Output

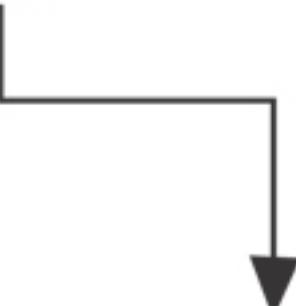
```
I am passing 5 to displayValue.  
The value is 5  
Now I am back in main.
```

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Function with a Parameter in Program 6-6

```
displayValue(5);  
  
void displayValue(int num)  
{  
    cout << "The value is " << num << endl;  
}
```

The function call in line 11 passes the value 5
as an argument to the function.

Other Parameter Terminology

- A parameter can also be called a formal parameter or a formal argument
- An argument can also be called an actual parameter or an actual argument

Parameters, Prototypes, and Function Headers

- For each function argument,
 - the prototype must include the data type of each parameter inside its parentheses
 - the header must include a declaration for each parameter in its ()

```
void evenOrOdd(int); //prototype  
void evenOrOdd(int num) //header  
evenOrOdd(val); //call
```

Function Call Notes

- Value of argument is copied into parameter when the function is called
- A parameter's scope is the function which uses it
- Function can have multiple parameters
- There must be a data type listed in the prototype () and an argument declaration in the function header () for each parameter
- Arguments will be promoted/demoted as necessary to match parameters

Passing Multiple Arguments

When calling a function and passing multiple arguments:

- ➊ the number of arguments in the call must match the prototype and definition
- ➋ the first argument will be used to initialize the first parameter, the second argument to initialize the second parameter, etc.

Passing Multiple Arguments in Program 6-8

Program 6-8

```
1 // This program demonstrates a function with three parameters.  
2 #include <iostream>  
3 using namespace std;  
4  
5 // Function Prototype  
6 void showSum(int, int, int);  
7  
8 int main()  
9 {  
10     int value1, value2, value3;  
11  
12     // Get three integers.  
13     cout << "Enter three integers and I will display "  
14     cout << "their sum: ";  
15     cin >> value1 >> value2 >> value3;  
16  
17     // Call showSum passing three arguments.  
18     showSum(value1, value2, value3);  
19     return 0;  
20 }  
21
```

(Program Continues)

Passing Multiple Arguments in Program 6-8

```
22 //*****  
23 // Definition of function showSum. *  
24 // It uses three integer parameters. Their sum is displayed. *  
25 //*****  
26  
27 void showSum(int num1, int num2, int num3)  
28 {  
29     cout << (num1 + num2 + num3) << endl;  
30 }
```

Program Output with Example Input Shown in Bold

Enter three integers and I will display their sum: **4 8 7 [Enter]**

19

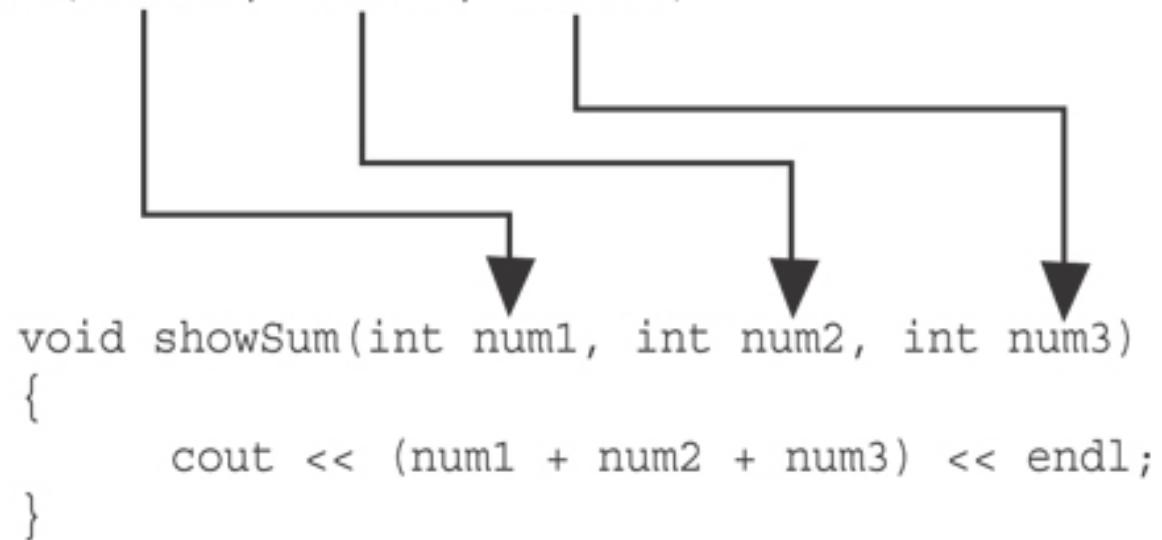
Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Passing Multiple Arguments in Program 6-8

Function Call → showSum(value1, value2, value3)

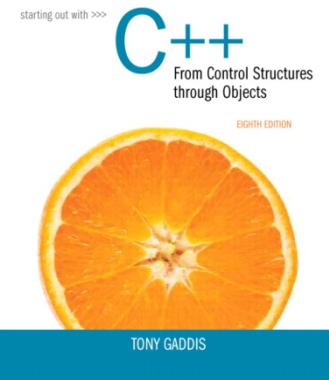


The function call in line 18 passes value1, value2, and value3 as arguments to the function.

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.



6.5

Passing Data by Value

Addison-Wesley
is an imprint of



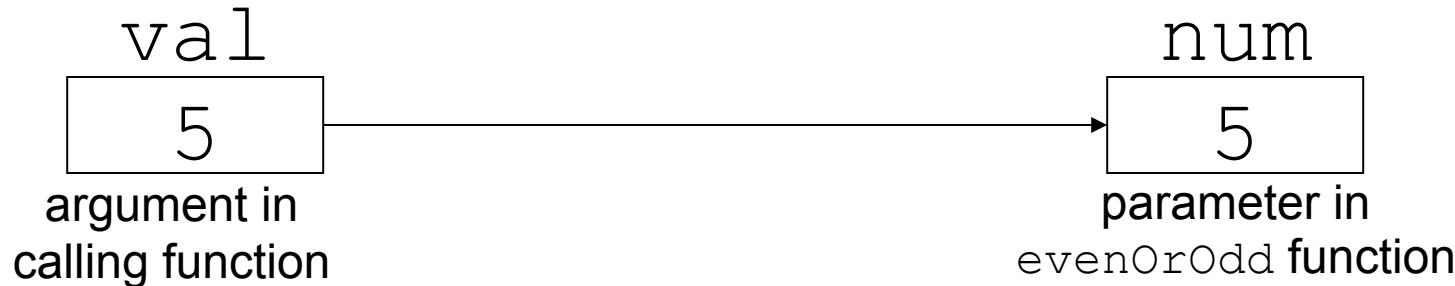
Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Passing Data by Value

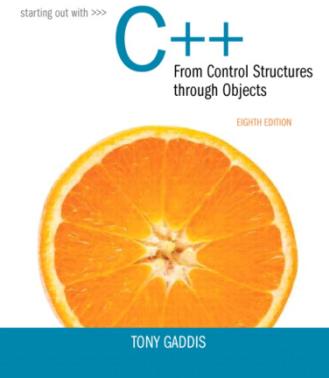
- Pass by value: when an argument is passed to a function, its value is copied into the parameter.
- Changes to the parameter in the function do not affect the value of the argument

Passing Information to Parameters by Value

- Example: `int val=5;
evenOrOdd(val);`



- evenOrOdd can change variable `num`, but it will have no effect on variable `val`



6.6

Using Functions in Menu-Driven Programs

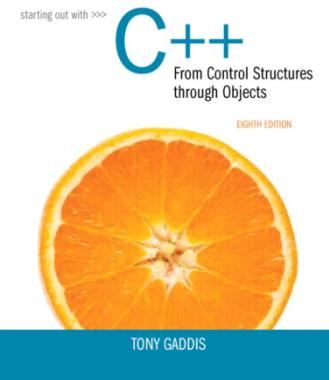
Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Using Functions in Menu-Driven Programs

- Functions can be used
 - to implement user choices from menu
 - to implement general-purpose tasks:
 - Higher-level functions can call general-purpose functions, minimizing the total number of functions and speeding program development time
- See *Program 6-10 in the book*



6.7

The return Statement

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

The return Statement

- Used to end execution of a function
- Can be placed anywhere in a function
 - Statements that follow the `return` statement will not be executed
- Can be used to prevent abnormal termination of program
- In a `void` function without a `return` statement, the function ends at its last `}`

Performing Division in Program 6-11

Program 6-11

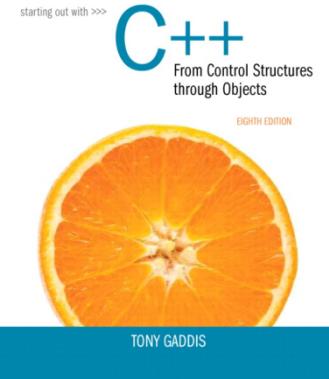
```
1 // This program uses a function to perform division. If division
2 // by zero is detected, the function returns.
3 #include <iostream>
4 using namespace std;
5
6 // Function prototype.
7 void divide(double, double);
8
9 int main()
10 {
11     double num1, num2;
12
13     cout << "Enter two numbers and I will divide the first\n";
14     cout << "number by the second number: ";
15     cin >> num1 >> num2;
16     divide(num1, num2);
17     return 0;
18 }
```

Performing Division in Program 6-11

```
20 //*****  
21 // Definition of function divide. *  
22 // Uses two parameters: arg1 and arg2. The function divides arg1*  
23 // by arg2 and shows the result. If arg2 is zero, however, the *  
24 // function returns. *  
25 //*****  
26  
27 void divide(double arg1, double arg2)  
28 {  
29     if (arg2 == 0.0)  
30     {  
31         cout << "Sorry, I cannot divide by zero.\n";  
32         return;  
33     }  
34     cout << "The quotient is " << (arg1 / arg2) << endl;  
35 }
```

Program Output with Example Input Shown in Bold

Enter two numbers and I will divide the first
number by the second number: **12 0 [Enter]**
Sorry, I cannot divide by zero.



6.8

Returning a Value From a Function

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Returning a Value From a Function

- A function can return a value back to the statement that called the function.
- You've already seen the `pow` function, which returns a value:

```
double x;  
x = pow(2.0, 10.0);
```

Returning a Value From a Function

- In a value-returning function, the `return` statement can be used to return a value from function to the point of call. Example:

```
int sum(int num1, int num2)
{
    double result;
    result = num1 + num2;
    return result;
}
```

A Value-Returning Function

Return Type



```
int sum(int num1, int num2)
{
    double result;
    result = num1 + num2;
    return result;
}
```



Value Being Returned

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

A Value-Returning Function

```
int sum(int num1, int num2)
{
    return num1 + num2;
}
```

Functions can return the values of expressions, such as `num1 + num2`

Function Returning a Value in Program 6-12

Program 6-12

```
1 // This program uses a function that returns a value.
2 #include <iostream>
3 using namespace std;
4
5 // Function prototype
6 int sum(int, int);
7
8 int main()
9 {
10     int value1 = 20,    // The first value
11         value2 = 40,    // The second value
12         total;        // To hold the total
13
14     // Call the sum function, passing the contents of
15     // value1 and value2 as arguments. Assign the return
16     // value to the total variable.
17     total = sum(value1, value2);
18
19     // Display the sum of the values.
20     cout << "The sum of " << value1 << " and "
21         << value2 << " is " << total << endl;
22     return 0;
23 }
```

Addison-Wesley
is an imprint of

(Program Continues)



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Function Returning a Value in Program 6-12

```
24
25 //*****
26 // Definition of function sum. This function returns *
27 // the sum of its two parameters. *
28 //*****
29
30 int sum(int num1, int num2)
31 {
32     return num1 + num2;
33 }
```

Program Output

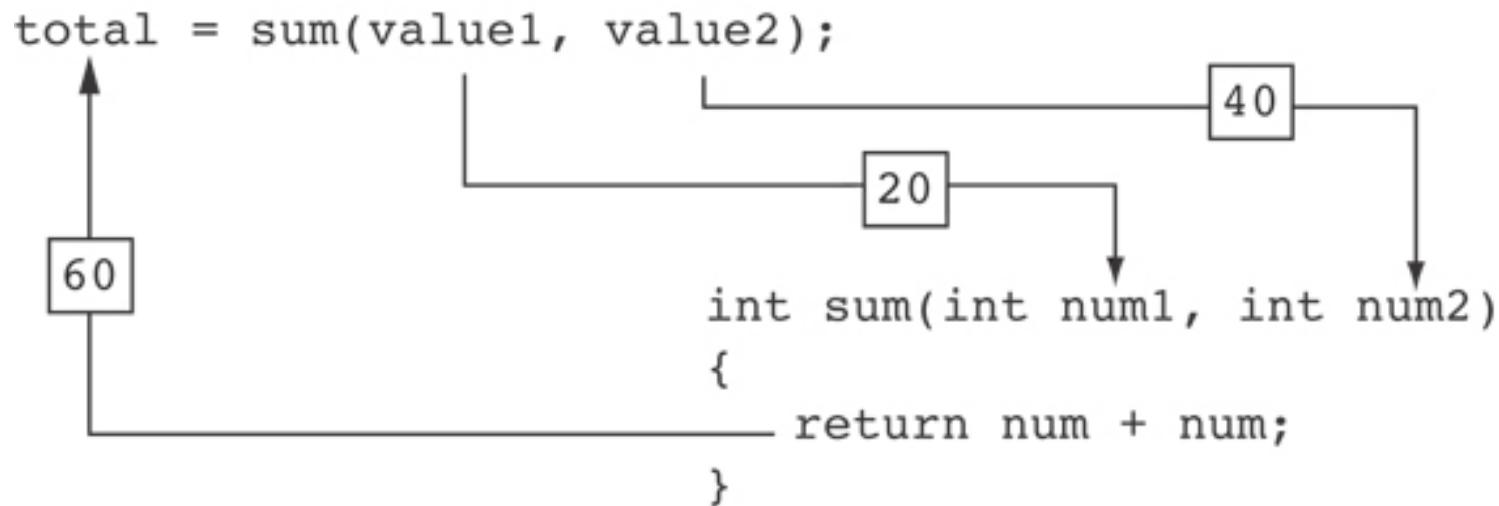
The sum of 20 and 40 is 60

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Function Returning a Value in Program 6-12



The statement in line 17 calls the `sum` function, passing `value1` and `value2` as arguments.
The return value is assigned to the `total` variable.

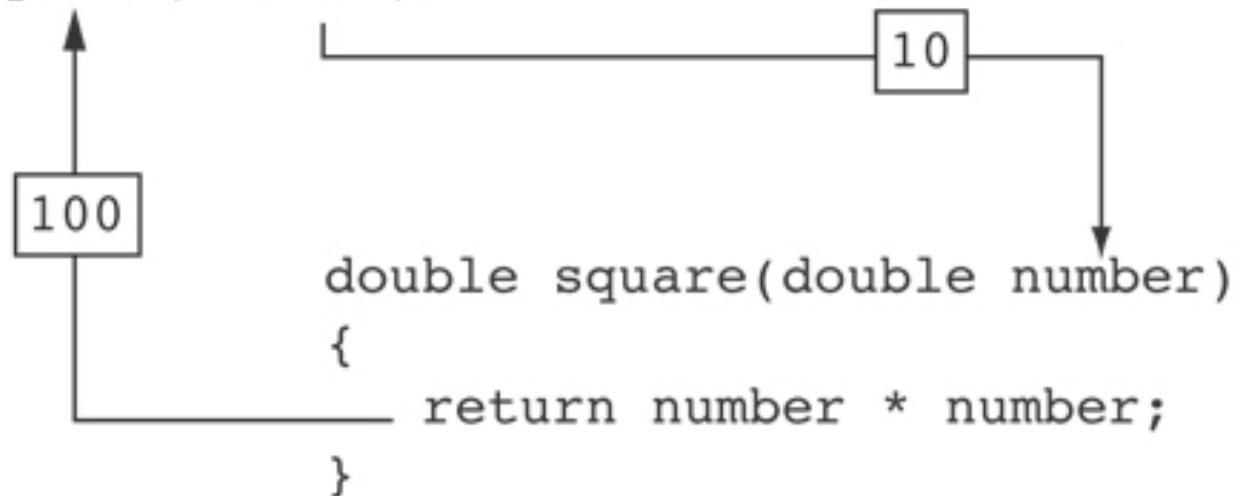
Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

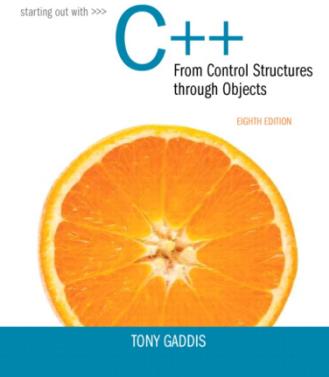
Another Example from Program 6-13

```
area = PI * square(radius);
```



Returning a Value From a Function

- The prototype and the definition must indicate the data type of return value (not `void`)
- Calling function should use return value:
 - assign it to a variable
 - send it to `cout`
 - use it in an expression



6.9

Returning a Boolean Value

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Returning a Boolean Value

- Function can return true or false
- Declare return type in function prototype and heading as bool
- Function body must contain return statement(s) that return true or false
- Calling function can use return value in a relational expression

Returning a Boolean Value in Program 6-15

Program 6-15

```
1 // This program uses a function that returns true or false.
2 #include <iostream>
3 using namespace std;
4
5 // Function prototype
6 bool isEven(int);
7
8 int main()
9 {
10     int val;
11
12     // Get a number from the user.
13     cout << "Enter an integer and I will tell you ";
14     cout << "if it is even or odd: ";
15     cin >> val;
16
17     // Indicate whether it is even or odd.
18     if (isEven(val))
19         cout << val << " is even.\n";
20     else
21         cout << val << " is odd.\n";
22     return 0;
23 }
```

(Program Continues)

Addison-Wesley
is an imprint of



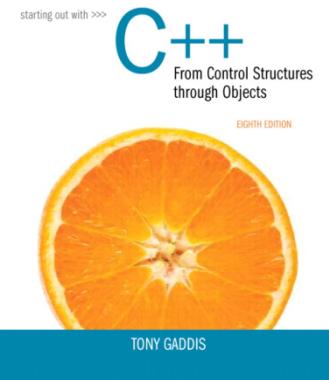
Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Returning a Boolean Value in Program 6-15

```
25 //*****  
26 // Definition of function isEven. This function accepts an      *  
27 // integer argument and tests it to be even or odd. The function  *  
28 // returns true if the argument is even or false if the argument  *  
29 // is odd. The return value is a bool.                         *  
30 //*****  
31  
32 bool isEven(int number)  
33 {  
34     bool status;  
35  
36     if (number % 2 == 0)  
37         status = true; // The number is even if there is no remainder.  
38     else  
39         status = false; // Otherwise, the number is odd.  
40     return status;  
41 }
```

Program Output with Example Input Shown in Bold

Enter an integer and I will tell you if it is even or odd: **5 [Enter]**
5 is odd.



6.10

Local and Global Variables

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Local and Global Variables

- Variables defined inside a function are *local* to that function. They are hidden from the statements in other functions, which normally cannot access them.
- Because the variables defined in a function are hidden, other functions may have separate, distinct variables with the same name.

Local Variables in Program 6-16

Program 6-16

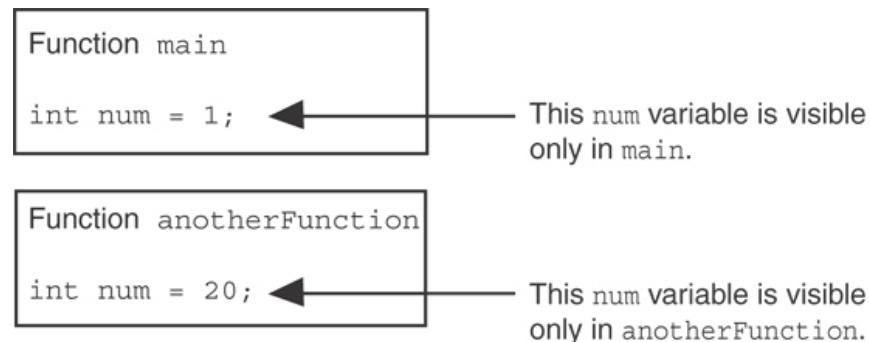
```
1 // This program shows that variables defined in a function
2 // are hidden from other functions.
3 #include <iostream>
4 using namespace std;
5
6 void anotherFunction(); // Function prototype
7
8 int main()
9 {
10     int num = 1;    // Local variable
11
12     cout << "In main, num is " << num << endl;
13     anotherFunction();
14     cout << "Back in main, num is " << num << endl;
15     return 0;
16 }
17
18 //*****
19 // Definition of anotherFunction
20 // It has a local variable, num, whose initial value
21 // is displayed.
22 //*****
23
24 void anotherFunction()
25 {
26     int num = 20;  // Local variable
27
28     cout << "In anotherFunction, num is " << num << endl;
29 }
```

Local Variables in Program 6-16

Program Output

```
In main, num is 1
In anotherFunction, num is 20
Back in main, num is 1
```

When the program is executing in `main`, the `num` variable defined in `main` is visible. When `anotherFunction` is called, however, only variables defined inside it are visible, so the `num` variable in `main` is hidden.



Local Variable Lifetime

- A function's local variables exist only while the function is executing. This is known as the *lifetime* of a local variable.
- When the function begins, its local variables and its parameter variables are created in memory, and when the function ends, the local variables and parameter variables are destroyed.
- This means that any value stored in a local variable is lost between calls to the function in which the variable is declared.

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Global Variables and Global Constants

- A global variable is any variable defined outside all the functions in a program.
- The scope of a global variable is the portion of the program from the variable definition to the end.
- This means that a global variable can be accessed by *all* functions that are defined after the global variable is defined.

Global Variables and Global Constants

- You should avoid using global variables because they make programs difficult to debug.
- Any global that you create should be *global constants*.

Global Constants in Program 6-19

Program 6-19

```
1 // This program calculates gross pay.  
2 #include <iostream>  
3 #include <iomanip>  
4 using namespace std;  
5  
6 // Global constants  
7 const double PAY_RATE = 22.55;      // Hourly pay rate  
8 const double BASE_HOURS = 40.0;     // Max non-overtime hours  
9 const double OT_MULTIPLIER = 1.5;   // Overtime multiplier  
10  
11 // Function prototypes  
12 double getBasePay(double);  
13 double getOvertimePay(double);  
14  
15 int main()  
16 {  
17     double hours,           // Hours worked  
18         basePay,          // Base pay  
19         overtime = 0.0,    // Overtime pay  
20         totalPay;         // Total pay
```

Global constants defined for values that do not change throughout the program's execution.

Global Constants in Program 6-19

The constants are then used for those values throughout the program.

```
29      // Get overtime pay, if any.  
30      if (hours > BASE_HOURS)  
31          overtime = getOvertimePay(hours);  
  
56      // Determine base pay.  
57      if (hoursWorked > BASE_HOURS)  
58          basePay = BASE_HOURS * PAY_RATE;  
59      else  
60          basePay = hoursWorked * PAY_RATE;  
  
75      // Determine overtime pay.  
76      if (hoursWorked > BASE_HOURS)  
77      {  
78          overtimePay = (hoursWorked - BASE_HOURS) *  
79              PAY_RATE * OT_MULTIPLIER;  
80      }  
81  }
```

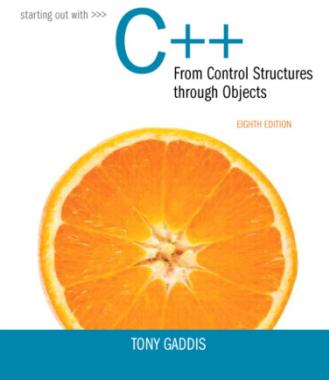
Initializing Local and Global Variables

- Local variables are not automatically initialized. They must be initialized by programmer.
- Global variables (not constants) are automatically initialized to 0 (numeric) or NULL (character) when the variable is defined.

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.



6.11

Static Local Variables

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Static Local Variables

- Local variables only exist while the function is executing. When the function terminates, the contents of local variables are lost.
- static local variables retain their contents between function calls.
- static local variables are defined and initialized only the first time the function is executed. 0 is the default initialization value.

Local Variables Do Not Retain Values Between Function calls in Program 6-21

Program 6-21

```
1 // This program shows that local variables do not retain
2 // their values between function calls.
3 #include <iostream>
4 using namespace std;
5
6 // Function prototype
7 void showLocal();
8
9 int main()
10 {
11     showLocal();
12     showLocal();
13     return 0;
14 }
15
```

Addison-Wesley
is an imprint of

(Program Continues)



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Local Variables Do Not Retain Values Between Function calls in Program 6-21

Program 6-21 *(continued)*

```
16 //*****  
17 // Definition of function showLocal. *  
18 // The initial value of localNum, which is 5, is displayed. *  
19 // The value of localNum is then changed to 99 before the *  
20 // function returns. *  
21 //*****  
22  
23 void showLocal()  
24 {  
25     int localNum = 5; // Local variable  
26  
27     cout << "localNum is " << localNum << endl;  
28     localNum = 99;  
29 }
```

Program Output

```
localNum is 5  
localNum is 5
```

In this program, each time `showLocal` is called, the `localNum` variable is re-created and initialized with the value 5.

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

A Different Approach, Using a Static Variable in Program 6-22

Program 6-22

```
1 // This program uses a static local variable.  
2 #include <iostream>  
3 using namespace std;  
4  
5 void showStatic(); // Function prototype  
6  
7 int main()  
8 {  
9     // Call the showStatic function five times.  
10    for (int count = 0; count < 5; count++)  
11        showStatic();  
12    return 0;  
13 }  
14
```

Addison-Wesley
is an imprint of

(Program Continues)



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

A Different Approach, Using a Static Variable in Program 6-22

Program 6-22 *(continued)*

```
15 //*****  
16 // Definition of function showStatic. *  
17 // statNum is a static local variable. Its value is displayed *  
18 // and then incremented just before the function returns. *  
19 //*****  
20  
21 void showStatic()  
22 {  
23     static int statNum;  
24  
25     cout << "statNum is " << statNum << endl;  
26     statNum++;  
27 }
```

Program Output

```
statNum is 0  
statNum is 1  
statNum is 2  
statNum is 3  
statNum is 4
```

statNum is automatically initialized to 0. Notice that it retains its value between function calls.

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

If you do initialize a local static variable, the initialization only happens once. See Program 6-23.

```
16 //*****  
17 // Definition of function showStatic. *  
18 // statNum is a static local variable. Its value is displayed *  
19 // and then incremented just before the function returns. *  
20 //*****  
21  
22 void showStatic()  
23 {  
24     static int statNum = 5;  
25  
26     cout << "statNum is " << statNum << endl;  
27     statNum++;  
28 }
```

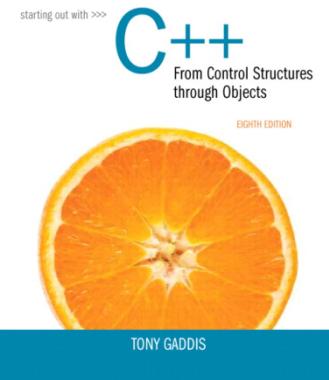
Program Output

```
statNum is 5  
statNum is 6  
statNum is 7  
statNum is 8  
statNum is 9
```

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.



6.12

Default Arguments

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Default Arguments

A Default argument is an argument that is passed automatically to a parameter if the argument is missing on the function call.

- Must be a constant declared in prototype:

```
void evenOrOdd(int = 0);
```

- Can be declared in header if no prototype

- Multi-parameter functions may have default arguments for some or all of them:

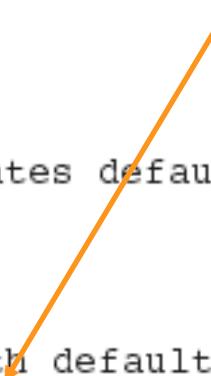
```
int getSum(int, int=0, int=0);
```

Default Arguments in Program 6-24

Default arguments specified in the prototype

Program 6-24

```
1 // This program demonstrates default function arguments.  
2 #include <iostream>  
3 using namespace std;  
4  
5 // Function prototype with default arguments  
6 void displayStars(int = 10, int = 1);  
7  
8 int main()  
9 {  
10    displayStars();           // Use default values for cols and rows.  
11    cout << endl;  
12    displayStars(5);         // Use default value for rows.  
13    cout << endl;  
14    displayStars(7, 3);      // Use 7 for cols and 3 for rows.  
15    return 0;  
16 }
```



(Program Continues)

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Default Arguments in Program 6-24

```
18 //*****
19 // Definition of function displayStars. *
20 // The default argument for cols is 10 and for rows is 1.* *
21 // This function displays a square made of asterisks. *
22 //*****
23
24 void displayStars(int cols, int rows)
25 {
26     // Nested loop. The outer loop controls the rows
27     // and the inner loop controls the columns.
28     for (int down = 0; down < rows; down++)
29     {
30         for (int across = 0; across < cols; across++)
31             cout << "*";
32         cout << endl;
33     }
34 }
```

Program Output

```
*****
*****
*****
*****
*****
```

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

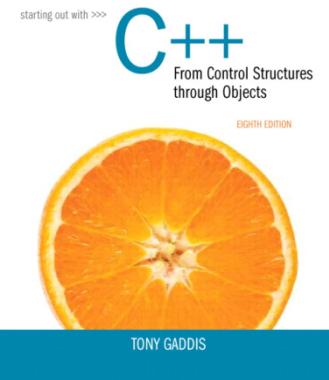
Default Arguments

- If not all parameters to a function have default values, the defaultless ones are declared first in the parameter list:

```
int getSum(int, int=0, int=0); // OK  
int getSum(int, int=0, int); // NO
```

- When an argument is omitted from a function call, all arguments after it must also be omitted:

```
sum = getSum(num1, num2); // OK  
sum = getSum(num1, , num3); // NO
```



6.13

Using Reference Variables as Parameters

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Using Reference Variables as Parameters

- A mechanism that allows a function to work with the original argument from the function call, not a copy of the argument
- Allows the function to modify values stored in the calling environment
- Provides a way for the function to ‘return’ more than one value

Passing by Reference

- A reference variable is an alias for another variable
- Defined with an ampersand (&)

```
void getDimensions(int&, int&);
```
- Changes to a reference variable are made to the variable it refers to
- Use reference variables to implement passing parameters *by reference*

Passing a Variable By Reference in Program 6-25

Program 6-25

The & here in the prototype indicates that the parameter is a reference variable.

```
1 // This program uses a reference variable as a function
2 // parameter.
3 #include <iostream>
4 using namespace std;
5
6 // Function prototype. The parameter is a reference variable.
7 void doubleNum(int &);
8
9 int main()
10 {
11     int value = 4;
12
13     cout << "In main, value is " << value << endl;
14     cout << "Now calling doubleNum..." << endl;
15     doubleNum(value);
16     cout << "Now back in main. value is " << value << endl;
17     return 0;
18 }
```

Here we are passing value by reference.

(Program Continues)

Passing a Variable By Reference in Program 6-25

The & also appears here in the function header.

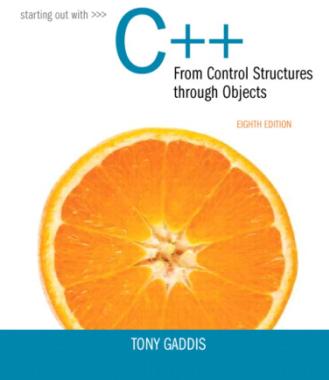
```
20 //*****  
21 // Definition of doubleNum.  
22 // The parameter refVar is a reference variable. The value  
23 // in refVar is doubled.  
24 //*****  
25  
26 void doubleNum (int &refVar)  
27 {  
28     refVar *= 2;  
29 }
```

Program Output

```
In main, value is 4  
Now calling doubleNum...  
Now back in main. value is 8
```

Reference Variable Notes

- Each reference parameter must contain &
- Space between type and & is unimportant
- Must use & in both prototype and header
- Argument passed to reference parameter must be a variable – cannot be an expression or constant
- Use when appropriate – don't use when argument should not be changed by function, or if function needs to return only 1 value



6.14

Overloading Functions

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Overloading Functions

- Overloaded functions have the same name but different parameter lists
- Can be used to create functions that perform the same task but take different parameter types or different number of parameters
- Compiler will determine which version of function to call by argument and parameter lists

Function Overloading Examples

Using these overloaded functions,

```
void getDimensions(int);           // 1
void getDimensions(int, int);      // 2
void getDimensions(int, double);   // 3
void getDimensions(double, double); // 4
```

the compiler will use them as follows:

```
int length, width;
double base, height;
getDimensions(length);           // 1
getDimensions(length, width);    // 2
getDimensions(length, height);   // 3
getDimensions(height, base);     // 4
```

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Function Overloading in Program 6-27

Program 6-27

```
1 // This program uses overloaded functions.  
2 #include <iostream>  
3 #include <iomanip>  
4 using namespace std;  
5  
6 // Function prototypes  
7 int square(int);  
8 double square(double);  
9  
10 int main()  
11 {  
12     int userInt;  
13     double userFloat;  
14  
15     // Get an int and a double.  
16     cout << fixed << showpoint << setprecision(2);  
17     cout << "Enter an integer and a floating-point value: ";  
18     cin >> userInt >> userFloat;  
19  
20     // Display their squares.  
21     cout << "Here are their squares: ";  
22     cout << square(userInt) << " and " << square(userFloat);  
23     return 0;  
24 }
```

The overloaded functions have different parameter lists

Passing a double

Passing an int

(Program Continues)

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Function Overloading in Program 6-27

```
26 //*****
27 // Definition of overloaded function square. *
28 // This function uses an int parameter, number. It returns the *
29 // square of number as an int. *
30 //*****
31
32 int square(int number)
33 {
34     return number * number;
35 }
36
37 //*****
38 // Definition of overloaded function square. *
39 // This function uses a double parameter, number. It returns *
40 // the square of number as a double. *
41 //*****
42
43 double square(double number)
44 {
45     return number * number;
46 }
```

Program Output with Example Input Shown in Bold

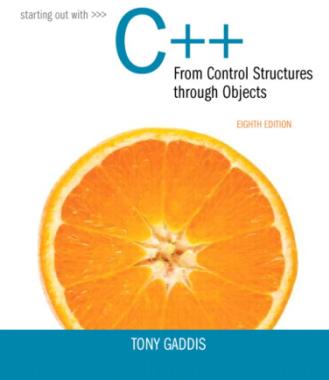
Enter an integer and a floating-point value: **12 4.2 [Enter]**

Here are their squares: 144 and 17.64

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.



6.15

The exit() Function

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

The exit () Function

- Terminates the execution of a program
- Can be called from any function
- Can pass an `int` value to operating system to indicate status of program termination
- Usually used for abnormal termination of program
- Requires `cstdlib` header file

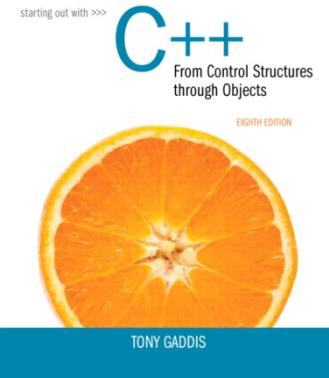
The exit () Function

- Example:

```
exit(0);
```

- The `cstdlib` header defines two constants that are commonly passed, to indicate success or failure:

```
exit(EXIT_SUCCESS);  
exit(EXIT_FAILURE);
```



6.16

Stubs and Drivers

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

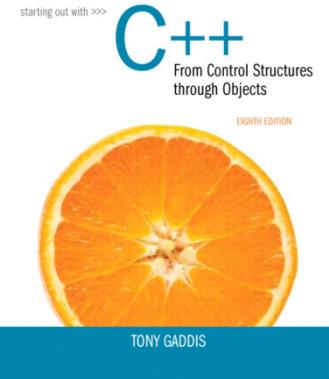
Stubs and Drivers

- Useful for testing and debugging program and function logic and design
- Stub: A dummy function used in place of an actual function
 - Usually displays a message indicating it was called. May also display parameters
- Driver: A function that tests another function by calling it
 - Various arguments are passed and return values are tested

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.



7.1

Arrays Hold Multiple Values

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Arrays Hold Multiple Values

- Array: variable that can store multiple values of the same type
- Values are stored in adjacent memory locations
- Declared using [] operator:

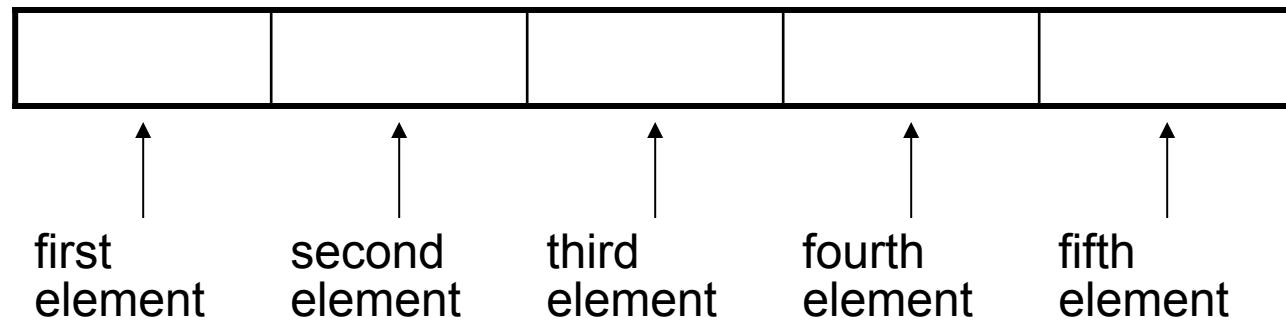
```
int tests[5];
```

Array - Memory Layout

- The definition:

```
int tests[5];
```

allocates the following memory:



Array Terminology

In the definition `int tests[5];`

- `int` is the data type of the array elements
- `tests` is the name of the array
- `5`, in `[5]`, is the size declarator. It shows the number of elements in the array.
- The size of an array is (number of elements) * (size of each element)

Array Terminology

- The size of an array is:
 - the total number of bytes allocated for it
 - $(\text{number of elements}) * (\text{number of bytes for each element})$

- Examples:

`int tests[5]` is an array of 20 bytes,
assuming 4 bytes for an `int`

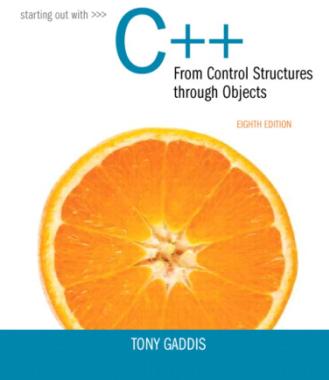
`long double measures[10]` is an array of
80 bytes, assuming 8 bytes for a `long double`

Size Declarators

- Named constants are commonly used as size declarators.

```
const int SIZE = 5;  
int tests[SIZE];
```

- This eases program maintenance when the size of the array needs to be changed.



7.2

Accessing Array Elements

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Accessing Array Elements

- Each element in an array is assigned a unique *subscript*.
- Subscripts start at 0

subscripts:



Accessing Array Elements

- The last element's subscript is $n-1$ where n is the number of elements in the array.

subscripts:



Accessing Array Elements

- Array elements can be used as regular variables:

```
tests[0] = 79;  
cout << tests[0];  
cin >> tests[1];  
tests[4] = tests[0] + tests[1];
```

- Arrays must be accessed via individual elements:

```
cout << tests; // not legal
```

Accessing Array Elements in Program 7-1

Program 7-1

```
1 // This program asks for the number of hours worked
2 // by six employees. It stores the values in an array.
3 #include <iostream>
4 using namespace std;
5
6 int main()
7 {
8     const int NUM_EMPLOYEES = 6;
9     int hours[NUM_EMPLOYEES];
10
11    // Get the hours worked by each employee.
12    cout << "Enter the hours worked by "
13        << NUM_EMPLOYEES << " employees: ";
14    cin >> hours[0];
15    cin >> hours[1];
16    cin >> hours[2];
17    cin >> hours[3];
18    cin >> hours[4];
19    cin >> hours[5];
20
```

(Program Continues)

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

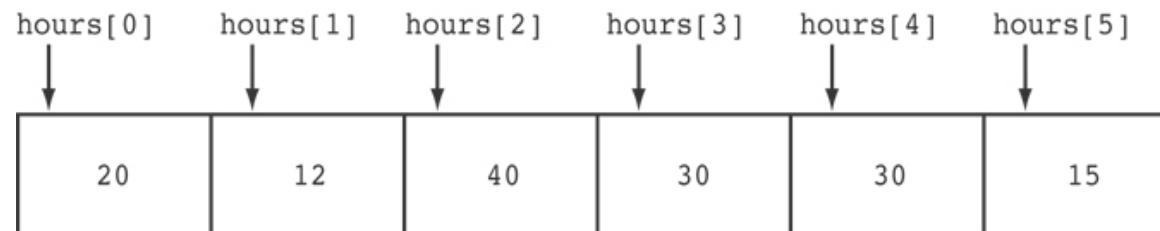
Accessing Array Elements in Program 7-1

```
21 // Display the values in the array.  
22 cout << "The hours you entered are:";  
23 cout << " " << hours[0];  
24 cout << " " << hours[1];  
25 cout << " " << hours[2];  
26 cout << " " << hours[3];  
27 cout << " " << hours[4];  
28 cout << " " << hours[5] << endl;  
29 return 0;  
30 }
```

Program Output with Example Input Shown in Bold

Enter the hours worked by 6 employees: **20 12 40 30 30 15** [Enter]
The hours you entered are: 20 12 40 30 30 15

Here are the contents of the `hours` array, with the values entered by the user in the example output:



Accessing Array Contents

- Can access element with a constant or literal subscript:

```
cout << tests[3] << endl;
```

- Can use integer expression as subscript:

```
int i = 5;
```

```
cout << tests[i] << endl;
```

Using a Loop to Step Through an Array

- Example – The following code defines an array, numbers, and assigns 99 to each element:

```
const int ARRAY_SIZE = 5;  
int numbers[ARRAY_SIZE];  
  
for (int count = 0; count < ARRAY_SIZE; count++)  
    numbers[count] = 99;
```

A Closer Look At the Loop

The variable count starts at 0, which is the first valid subscript value.

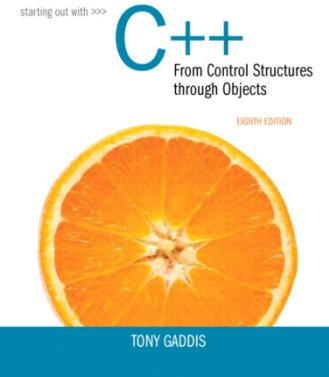
```
for (count = 0; count < ARRAY_SIZE; count++)  
    numbers[count] = 99;
```

The loop ends when the variable count reaches 5, which is the first invalid subscript value.

The variable count is incremented after each iteration.

Default Initialization

- Global array → all elements initialized to 0 by default
- Local array → all elements *uninitialized* by default



7.3

No Bounds Checking in C++

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

No Bounds Checking in C++

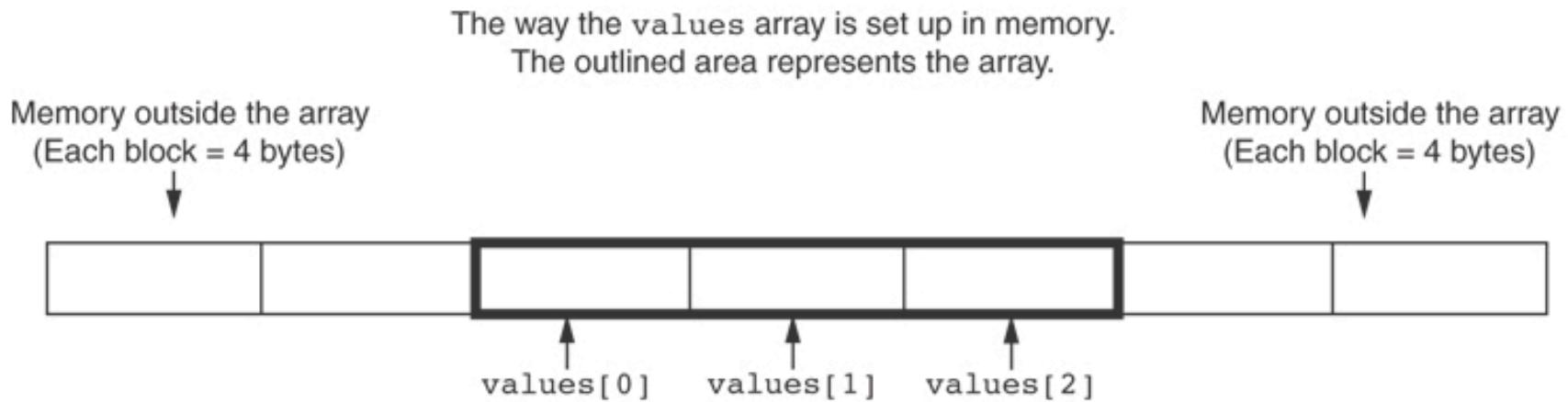
- When you use a value as an array subscript, C++ does not check it to make sure it is a *valid* subscript.
- In other words, you can use subscripts that are beyond the bounds of the array.

Code From Program 7-5

- The following code defines a three-element array, and then writes five values to it!

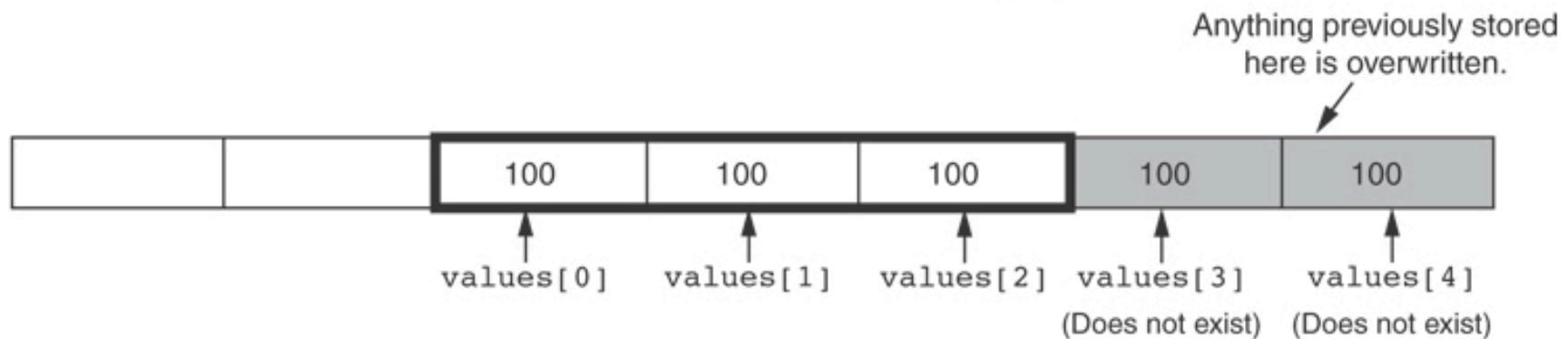
```
9  const int SIZE = 3;    // Constant for the array size
10 int values[SIZE];      // An array of 3 integers
11 int count;              // Loop counter variable
12
13 // Attempt to store five numbers in the three-element array.
14 cout << "I will store 5 numbers in a 3 element array!\n";
15 for (count = 0; count < 5; count++)
16     values[count] = 100;
```

What the Code Does



How the numbers assigned to the array overflow the array's boundaries.
The shaded area is the section of memory illegally written to.

Add
is a



No Bounds Checking in C++

- Be careful not to use invalid subscripts.
- Doing so can corrupt other memory locations, crash program, or lock up computer, and cause elusive bugs.

Addison-Wesley
is an imprint of

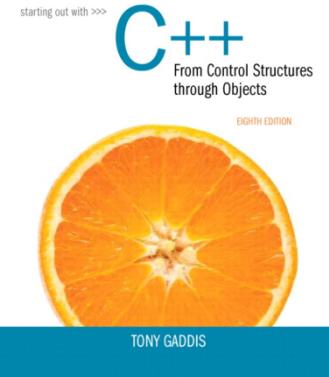


Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Off-By-One Errors

- An off-by-one error happens when you use array subscripts that are off by one.
- This can happen when you start subscripts at 1 rather than 0:

```
// This code has an off-by-one error.  
const int SIZE = 100;  
int numbers[SIZE];  
for (int count = 1; count <= SIZE; count++)  
    numbers [count] = 0;
```



7.4

Array Initialization

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Array Initialization

- Arrays can be initialized with an initialization list:

```
const int SIZE = 5;  
int tests[SIZE] = {79, 82, 91, 77, 84};
```

- The values are stored in the array in the order in which they appear in the list.
- The initialization list cannot exceed the array size.

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Code From Program 7-6

```
7     const int MONTHS = 12;
8     int days[MONTHS] = { 31, 28, 31, 30,
9                         31, 30, 31, 31,
10                        30, 31, 30, 31};
11
12    for (int count = 0; count < MONTHS; count++)
13    {
14        cout << "Month " << (count + 1) << " has ";
15        cout << days[count] << " days.\n";
16    }
```

Program Output

```
Month 1 has 31 days.
Month 2 has 28 days.
Month 3 has 31 days.
Month 4 has 30 days.
Month 5 has 31 days.
Month 6 has 30 days.
Month 7 has 31 days.
Month 8 has 31 days.
Month 9 has 30 days.
Month 10 has 31 days.
Month 11 has 30 days.
Month 12 has 31 days.
```

Addison-Wesley
is an imprint of

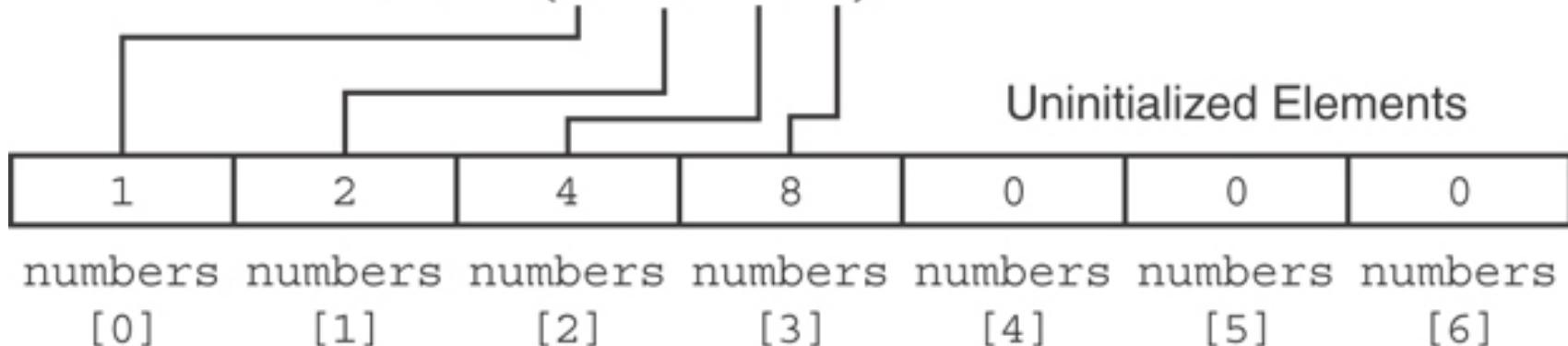


Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Partial Array Initialization

- If array is initialized with fewer initial values than the size declarator, the remaining elements will be set to 0 :

```
int numbers[7] = {1, 2, 4, 8};
```



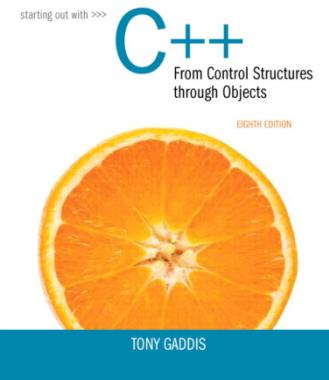
Implicit Array Sizing

- Can determine array size by the size of the initialization list:

```
int quizzes[] = {12, 17, 15, 11};
```

12	17	15	11
----	----	----	----

- Must use either array size declarator or initialization list at array definition



7.5

The Range-Based for Loop

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

The Range-Based for Loop

- C++ 11 provides a specialized version of the `for` loop that, in many circumstances, simplifies array processing.
- *The range-based for loop is a loop that iterates once for each element in an array.*
- *Each time the loop iterates, it copies an element from the array to a built-in variable, known as the range variable.*
- The range-based `for` loop automatically knows the number of elements in an array.
 - You do not have to use a counter variable.
 - You do not have to worry about stepping outside the bounds of the array.

The Range-Based for Loop

- Here is the general format of the range-based for loop:

```
for (dataType rangeVariable : array)
    statement;
```

- dataType* is the data type of the range variable.
- rangeVariable* is the name of the range variable. This variable will receive the value of a different array element during each loop iteration.
- array* is the name of an array on which you wish the loop to operate.
- statement* is a statement that executes during a loop iteration. If you need to execute more than one statement in the loop, enclose the statements in a set of braces.

The range-based for loop in Program 7-10

```
// This program demonstrates the range-based for
loop.

#include <iostream>
using namespace std;

int main()
{
    // Define an array of integers.
    int numbers[] = { 10, 20, 30, 40, 50 };

    // Display the values in the array.
    for (int val : numbers)
        cout << val << endl;

    return 0;
}
```

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Modifying an Array with a Range-Based for Loop

- As the range-based `for` loop executes, its range variable contains only a copy of an array element.
 - You cannot use a range-based `for` loop to modify the contents of an array unless you declare the range variable as a reference.
 - To declare the range variable as a reference variable, simply write an ampersand (`&`) in front of its name in the loop header.
- Program 7-12 demonstrates

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Modifying an Array with a Range-Based for Loop in Program 7-12

```
const int SIZE = 5;
int numbers[5];

// Get values for the array.
for (int &val : numbers)
{
    cout << "Enter an integer value: ";
    cin >> val;
}

// Display the values in the array.
cout << "Here are the values you entered:\n";
for (int val : numbers)
    cout << val << endl;
```

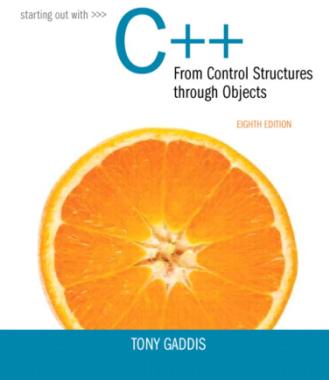
Modifying an Array with a Range-Based for Loop

You can use the `auto` key word with a reference range variable. For example, the code in lines 12 through 16 in Program 7-12 could have been written like this:

```
for (auto &val : numbers)
{
    cout << "Enter an integer value: ";
    cin >> val;
}
```

The Range-Based for Loop versus the Regular for Loop

- The range-based for loop can be used in any situation where you need to step through the elements of an array, and you do not need to use the element subscripts.
- If you need the element subscript for some purpose, use the regular for loop.



7.6

Processing Array Contents

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Processing Array Contents

- Array elements can be treated as ordinary variables of the same type as the array
- When using `++`, `--` operators, don't confuse the element with the subscript:

```
tests[i]++; // add 1 to tests[i]
tests[i++]; // increment i, no
            // effect on tests
```

Array Assignment

To copy one array to another,

- Don't try to assign one array to the other:

```
newTests = tests; // Won't work
```

- Instead, assign element-by-element:

```
for (i = 0; i < ARRAY_SIZE; i++)  
    newTests[i] = tests[i];
```

Printing the Contents of an Array

- You can display the contents of a *character* array by sending its name to cout:

```
char fName[] = "Henry";  
cout << fName << endl;
```

But, this ONLY works with character arrays!

Printing the Contents of an Array

- For other types of arrays, you must print element-by-element:

```
for (i = 0; i < ARRAY_SIZE; i++)  
    cout << tests[i] << endl;
```

Printing the Contents of an Array

- In C++ 11 you can use the range-based for loop to display an array's contents, as shown here:

```
for (int val : numbers)
    cout << val << endl;
```

Summing and Averaging Array Elements

- Use a simple loop to add together array elements:

```
int tnum;  
double average, sum = 0;  
for (tnum = 0; tnum < SIZE; tnum++)  
    sum += tests[tnum];
```

- Once summed, can compute average:

```
average = sum / SIZE;
```

Summing and Averaging Array Elements

- In C++ 11 you can use the range-based for loop, as shown here:

```
double total = 0;    // Initialize accumulator
double average;      // Will hold the average
for (int val : scores)
    total += val;
average = total / NUM_SCORES;
```

Finding the Highest Value in an Array

```
int count;
int highest;
highest = numbers[0];
for (count = 1; count < SIZE; count++)
{
    if (numbers[count] > highest)
        highest = numbers[count];
}
```

When this code is finished, the `highest` variable will contain the highest value in the `numbers` array.

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Finding the Lowest Value in an Array

```
int count;
int lowest;
lowest = numbers[0];
for (count = 1; count < SIZE; count++)
{
    if (numbers[count] < lowest)
        lowest = numbers[count];
}
```

When this code is finished, the `lowest` variable will contain the lowest value in the `numbers` array.

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

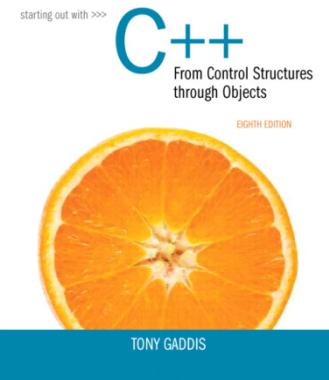
Partially-Filled Arrays

- If it is unknown how much data an array will be holding:
 - Make the array large enough to hold the largest expected number of elements.
 - Use a counter variable to keep track of the number of items stored in the array.

Comparing Arrays

- To compare two arrays, you must compare element-by-element:

```
const int SIZE = 5;
int firstArray[SIZE] = { 5, 10, 15, 20, 25 };
int secondArray[SIZE] = { 5, 10, 15, 20, 25 };
bool arraysEqual = true; // Flag variable
int count = 0;           // Loop counter variable
// Compare the two arrays.
while (arraysEqual && count < SIZE)
{
    if (firstArray[count] != secondArray[count])
        arraysEqual = false;
    count++;
}
if (arraysEqual)
    cout << "The arrays are equal.\n";
else
    cout << "The arrays are not equal.\n";
```



7.7

Using Parallel Arrays

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Using Parallel Arrays

- Parallel arrays: two or more arrays that contain related data
- A subscript is used to relate arrays: elements at same subscript are related
- Arrays may be of different types

Parallel Array Example

```
const int SIZE = 5;      // Array size
int id[SIZE];           // student ID
double average[SIZE];   // course average
char grade[SIZE];       // course grade

...
for(int i = 0; i < SIZE; i++)
{
    cout << "Student ID: " << id[i]
        << " average: " << average[i]
        << " grade: " << grade[i]
        << endl;
}
```

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Parallel Arrays in Program 7-15

Program 7-15

```
1 // This program uses two parallel arrays: one for hours
2 // worked and one for pay rate.
3 #include <iostream>
4 #include <iomanip>
5 using namespace std;
6
7 int main()
8 {
9     const int NUM_EMPLOYEES = 5;    // Number of employees
10    int hours[NUM_EMPLOYEES];      // Holds hours worked
11    double payRate[NUM_EMPLOYEES]; // Holds pay rates
12
13    // Input the hours worked and the hourly pay rate.
14    cout << "Enter the hours worked by " << NUM_EMPLOYEES
15        << " employees and their\n"
16        << "hourly pay rates.\n";
17    for (int index = 0; index < NUM_EMPLOYEES; index++)
18    {
19        cout << "Hours worked by employee #" << (index+1) << ":" ;
20        cin >> hours[index];
21        cout << "Hourly pay rate for employee #" << (index+1) << ":" ;
22        cin >> payRate[index];
23    }
24 }
```

(Program Continues)

Parallel Arrays in Program 7-15

```
25 // Display each employee's gross pay.  
26 cout << "Here is the gross pay for each employee:\n";  
27 cout << fixed << showpoint << setprecision(2);  
28 for (int index = 0; index < NUM_EMPLOYEES; index++)  
29 {  
30     double grossPay = hours[index] * payRate[index];  
31     cout << "Employee #" << (index + 1);  
32     cout << ": $" << grossPay << endl;  
33 }  
34 return 0;  
35 }
```

Program Output with Example Input Shown in Bold

Enter the hours worked by 5 employees and their hourly pay rates.

Hours worked by employee #1: **10** [Enter]
Hourly pay rate for employee #1: **9.75** [Enter]
Hours worked by employee #2: **15** [Enter]
Hourly pay rate for employee #2: **8.62** [Enter]
Hours worked by employee #3: **20** [Enter]
Hourly pay rate for employee #3: **10.50** [Enter]
Hours worked by employee #4: **40** [Enter]
Hourly pay rate for employee #4: **18.75** [Enter]
Hours worked by employee #5: **40** [Enter]
Hourly pay rate for employee #5: **15.65** [Enter]

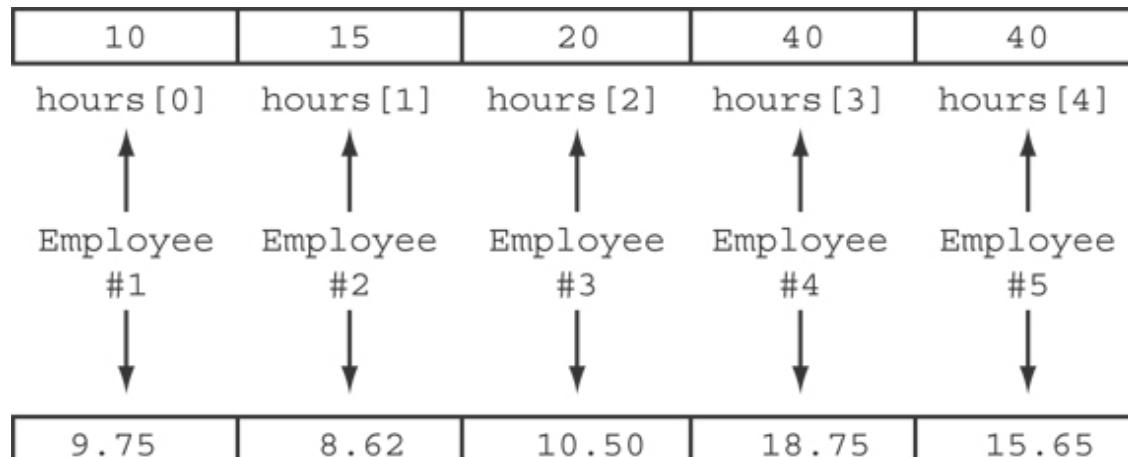
Parallel Arrays in Program 7-15

Program 7-15 *(continued)*

Here is the gross pay for each employee:

Employee #1: \$97.50
Employee #2: \$129.30
Employee #3: \$210.00
Employee #4: \$750.00
Employee #5: \$626.00

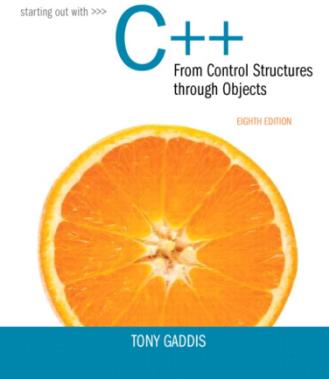
The hours and payRate arrays are related through their subscripts:



Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.



7.8

Arrays as Function Arguments

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Arrays as Function Arguments

- To pass an array to a function, just use the array name:

```
showScores(tests);
```

- To define a function that takes an array parameter, use empty [] for array argument:

```
void showScores(int []);
                    // function prototype
void showScores(int tests[])
                    // function header
```

Arrays as Function Arguments

- When passing an array to a function, it is common to pass array size so that function knows how many elements to process:

```
showScores(tests, ARRAY_SIZE);
```

- Array size must also be reflected in prototype, header:

```
void showScores(int [], int);  
                    // function prototype  
void showScores(int tests[], int size)  
                    // function header
```

Passing an Array to a Function in Program 7-17

Program 7-17

```
1 // This program demonstrates an array being passed to a function.  
2 #include <iostream>  
3 using namespace std;  
4  
5 void showValues(int [], int); // Function prototype  
6  
7 int main()  
8 {  
9     const int ARRAY_SIZE = 8;  
10    int numbers[ARRAY_SIZE] = {5, 10, 15, 20, 25, 30, 35, 40};  
11  
12    showValues(numbers, ARRAY_SIZE);  
13    return 0;  
14 }  
15
```

Passing an Array to a Function in Program 7-17

```
16 //*****  
17 // Definition of function showValue. *  
18 // This function accepts an array of integers and *  
19 // the array's size as its arguments. The contents *  
20 // of the array are displayed. *  
21 //*****  
22  
23 void showValues(int nums[ ], int size)  
24 {  
25     for (int index = 0; index < size; index++)  
26         cout << nums[index] << " ";  
27     cout << endl;  
28 }
```

Program Output

```
5 10 15 20 25 30 35 40
```

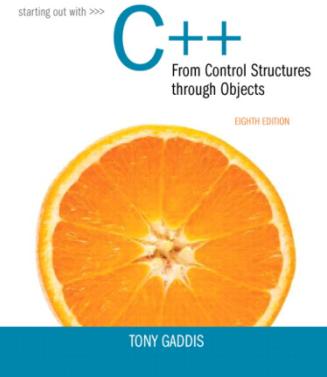
Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Modifying Arrays in Functions

- Array names in functions are like reference variables – changes made to array in a function are reflected in actual array in calling function
- Need to exercise caution that array is not inadvertently changed by a function



7.9

Two-Dimensional Arrays

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Two-Dimensional Arrays

- Can define one array for multiple sets of data
- Like a table in a spreadsheet
- Use two size declarators in definition:

```
const int ROWS = 4, COLS = 3;  
int exams [ROWS] [COLS];
```

- First declarator is number of rows; second is number of columns

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Two-Dimensional Array Representation

```
const int ROWS = 4, COLS = 3; int  
exams [ROWS] [COLS];
```

columns

exams [0] [0]	exams [0] [1]	exams [0] [2]
exams [1] [0]	exams [1] [1]	exams [1] [2]
exams [2] [0]	exams [2] [1]	exams [2] [2]
exams [3] [0]	exams [3] [1]	exams [3] [2]

rows

- Use two subscripts to access element:

```
exams [2] [2] = 86;
```

A Two-dimensional Array in Program 7-21

Program 7-21

```
1 // This program demonstrates a two-dimensional array.  
2 #include <iostream>  
3 #include <iomanip>  
4 using namespace std;  
5  
6 int main()  
7 {  
8     const int NUM_DIVS = 3;           // Number of divisions  
9     const int NUM_QTRS = 4;          // Number of quarters  
10    double sales[NUM_DIVS][NUM_QTRS]; // Array with 3 rows and 4 columns.  
11    double totalSales = 0;           // To hold the total sales.  
12    int div, qtr;                  // Loop counters.  
13  
14    cout << "This program will calculate the total sales of\n";  
15    cout << "all the company's divisions.\n";  
16    cout << "Enter the following sales information:\n\n";  
17
```

(program continues)

A Two-dimensional Array in Program 7-21

Program 7-21 *(continued)*

```
18     // Nested loops to fill the array with quarterly
19     // sales figures for each division.
20     for (div = 0; div < NUM_DIVS; div++)
21     {
22         for (qtr = 0; qtr < NUM_QTRS; qtr++)
23         {
24             cout << "Division " << (div + 1);
25             cout << ", Quarter " << (qtr + 1) << ": $";
26             cin >> sales[div][qtr];
27         }
28         cout << endl; // Print blank line.
29     }
30
31     // Nested loops used to add all the elements.
32     for (div = 0; div < NUM_DIVS; div++)
33     {
34         for (qtr = 0; qtr < NUM_QTRS; qtr++)
35             totalSales += sales[div][qtr];
36     }
37
38     cout << fixed << showpoint << setprecision(2);
39     cout << "The total sales for the company are: $";
40     cout << totalSales << endl;
41     return 0;
42 }
```

A Two-dimensional Array in Program 7-21

Program Output with Example Input Shown in Bold

This program will calculate the total sales of all the company's divisions.

Enter the following sales data:

Division 1, Quarter 1: **\$31569.45 [Enter]**

Division 1, Quarter 2: **\$29654.23 [Enter]**

Division 1, Quarter 3: **\$32982.54 [Enter]**

Division 1, Quarter 4: **\$39651.21 [Enter]**

Division 2, Quarter 1: **\$56321.02 [Enter]**

Division 2, Quarter 2: **\$54128.63 [Enter]**

Division 2, Quarter 3: **\$41235.85 [Enter]**

Division 2, Quarter 4: **\$54652.33 [Enter]**

Division 3, Quarter 1: **\$29654.35 [Enter]**

Division 3, Quarter 2: **\$28963.32 [Enter]**

Division 3, Quarter 3: **\$25353.55 [Enter]**

Division 3, Quarter 4: **\$32615.88 [Enter]**

The total sales for the company are: \$456782.34

2D Array Initialization

- Two-dimensional arrays are initialized row-by-row:

```
const int ROWS = 2, COLS = 2;  
int exams [ROWS] [COLS] = { {84, 78},  
                           {92, 97} };
```

84	78
92	97

- Can omit inner { }, some initial values in a row – array elements without initial values will be set to 0 or NULL

Two-Dimensional Array as Parameter, Argument

- Use array name as argument in function call:

```
getExams(exams, 2);
```

- Use empty [] for row, size declarator for column in prototype, header:

```
const int COLS = 2;
```

```
// Prototype
```

```
void getExams(int [] [COLS], int);
```

```
// Header
```

```
void getExams(int exams [] [COLS], int rows)
```

Example – The showArray Function from Program 7-22

```
30 //*****  
31 // Function Definition for showArray *  
32 // The first argument is a two-dimensional int array with COLS *  
33 // columns. The second argument, rows, specifies the number of *  
34 // rows in the array. The function displays the array's contents. *  
35 //*****  
36  
37 void showArray(int array[][COLS], int rows)  
38 {  
39     for (int x = 0; x < rows; x++)  
40     {  
41         for (int y = 0; y < COLS; y++)  
42         {  
43             cout << setw(4) << array[x][y] << " ";  
44         }  
45         cout << endl;  
46     }  
47 }
```

How showArray is Called

```
15     int table1[TBL1_ROWS][COLS] = {{1, 2, 3, 4},  
16                     {5, 6, 7, 8},  
17                     {9, 10, 11, 12}};  
18     int table2[TBL2_ROWS][COLS] = {{10, 20, 30, 40},  
19                     {50, 60, 70, 80},  
20                     {90, 100, 110, 120},  
21                     {130, 140, 150, 160}};  
22  
23     cout << "The contents of table1 are:\n";  
24     showArray(table1, TBL1_ROWS);  
25     cout << "The contents of table2 are:\n";  
26     showArray(table2, TBL2_ROWS);
```

Summing All the Elements in a Two-Dimensional Array

- Given the following definitions:

```
const int NUM_ROWS = 5; // Number of rows
const int NUM_COLS = 5; // Number of columns
int total = 0;           // Accumulator
int numbers[NUM_ROWS][NUM_COLS] =
    {{2, 7, 9, 6, 4},
     {6, 1, 8, 9, 4},
     {4, 3, 7, 2, 9},
     {9, 9, 0, 3, 1},
     {6, 2, 7, 4, 1}};
```

Summing All the Elements in a Two-Dimensional Array

```
// Sum the array elements.  
for (int row = 0; row < NUM_ROWS; row++)  
{  
    for (int col = 0; col < NUM_COLS; col++)  
        total += numbers[row][col];  
}  
  
// Display the sum.  
cout << "The total is " << total << endl;
```

Summing the Rows of a Two-Dimensional Array

- Given the following definitions:

```
const int NUM_STUDENTS = 3;
const int NUM_SCORES = 5;
double total;    // Accumulator
double average; // To hold average scores
double scores[NUM_STUDENTS][NUM_SCORES] =
    {{88, 97, 79, 86, 94},
     {86, 91, 78, 79, 84},
     {82, 73, 77, 82, 89}};
```

Summing the Rows of a Two-Dimensional Array

```
// Get each student's average score.  
for (int row = 0; row < NUM_STUDENTS; row++)  
{  
    // Set the accumulator.  
    total = 0;  
    // Sum a row.  
    for (int col = 0; col < NUM_SCORES; col++)  
        total += scores[row][col];  
    // Get the average  
    average = total / NUM_SCORES;  
    // Display the average.  
    cout << "Score average for student "  
        << (row + 1) << " is " << average << endl;  
}
```

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Summing the Columns of a Two-Dimensional Array

- Given the following definitions:

```
const int NUM_STUDENTS = 3;
const int NUM_SCORES = 5;
double total;    // Accumulator
double average; // To hold average scores
double scores[NUM_STUDENTS][NUM_SCORES] =
    {{88, 97, 79, 86, 94},
     {86, 91, 78, 79, 84},
     {82, 73, 77, 82, 89}};
```

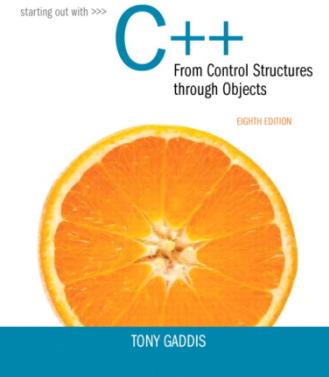
Summing the Columns of a Two-Dimensional Array

```
// Get the class average for each score.  
for (int col = 0; col < NUM_SCORES; col++)  
{  
    // Reset the accumulator.  
    total = 0;  
    // Sum a column  
    for (int row = 0; row < NUM_STUDENTS; row++)  
        total += scores[row][col];  
    // Get the average  
    average = total / NUM_STUDENTS;  
    // Display the class average.  
    cout << "Class average for test " << (col + 1)  
        << " is " << average << endl;
```

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.



7.10

Arrays with Three or More Dimensions

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Arrays with Three or More Dimensions

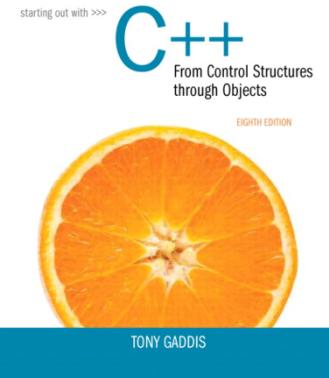
- Can define arrays with any number of dimensions:

```
short rectSolid[2][3][5];
```

```
double timeGrid[3][4][3][4];
```

- When used as parameter, specify all but 1st dimension in prototype, heading:

```
void getRectSolid(short [][] [3] [5]);
```



7.12

Introduction to the STL vector

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Introduction to the STL vector

- A data type defined in the Standard Template Library (covered more in Chapter 16)
- Can hold values of any type:

```
vector<int> scores;
```
- Automatically adds space as more is needed – no need to determine size at definition
- Can use [] to access elements

Addison-Wesley
is an imprint of

PEARSON

Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Declaring Vectors

- You must `#include<vector>`
- Declare a vector to hold `int` element:
`vector<int> scores;`
- Declare a vector with initial size 30:
`vector<int> scores(30);`
- Declare a vector and initialize all elements to 0:
`vector<int> scores(30, 0);`
- Declare a vector initialized to size and contents of another vector:
`vector<int> finals(scores);`

Adding Elements to a Vector

- If you are using C++ 11, you can initialize a vector with a list of values:

```
vector<int> numbers { 10, 20, 30, 40 };
```

- Use `push_back` member function to add element to a full array or to an array that had no defined size:

```
scores.push_back(75);
```

- Use `size` member function to determine size of a vector:

```
howbig = scores.size();
```

Removing Vector Elements

- Use `pop_back` member function to remove last element from vector:

```
scores.pop_back();
```

- To remove all contents of vector, use `clear` member function:

```
scores.clear();
```

- To determine if vector is empty, use `empty` member function:

```
while (!scores.empty()) ...
```

Using the Range-Based for Loop with a vector in C++ 11

Program 7-25

```
1 // This program demonstrates the range-based for loop with a vector.  
2 include <iostream>  
3 #include <vector>  
4 using namespace std;  
5  
6 int main()  
7 {  
8     // Define and initialize a vector.  
9     vector<int> numbers { 10, 20, 30, 40, 50 };  
10  
11    // Display the vector elements.  
12    for (int val : numbers)  
13        cout << val << endl;  
14  
15    return 0;  
16 }
```

Program Output

```
10  
20  
30  
40  
50
```

Addison-Wesley
is an imprint of



Copyright © 2015, 2012, 2009 Pearson Education, Inc., Publishing as Addison-Wesley All rights reserved.

Other Useful Member Functions

Member Function	Description	Example
at(elt)	Returns the value of the element at position elt in the vector	<code>cout << vec1.at(i);</code>
capacity()	Returns the maximum number of elements a vector can store without allocating more memory	<code>maxelts = vec1.capacity();</code>
reverse()	Reverse the order of the elements in a vector	<code>vec1.reverse();</code>
resize(elts, val)	Add elements to a vector, optionally initializes them	<code>vec1.resize(5, 0);</code>
swap(vec2)	Exchange the contents of two vectors	<code>vec1.swap(vec2);</code>