Lecture Notes

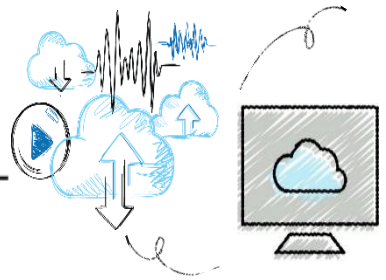
Chapter 6

User-Defined Functions

ECE 111: Introduction to C and C++ Programming

Instructor: Dr. Shayan (Sean) Taheri

Gannon University (GU)



Personal Information

- Name: Shayan (Sean) Taheri.
- Date of Birth: July/28/1991.
- Current Position: Assistant Professor at Gannon University
- Previous Position: Postdoctoral Fellow at University of Florida.
- Ph.D. Degree: Electrical Engineering from the University of Central Florida.
- M.S. Degree: Computer Engineering from the Utah State University.
- University Profile: https://www.gannon.edu/FacultyProfiles.aspx?profile=taheri001

CENGAGE Learning

© 2018 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part, except for use as permitted in a license distributed with a certain product or service or otherwise on a password-protected website for classroom use.

Objectives (1 of 2)

- In this chapter, you will:
  - Learn about standard (predefined) functions and discover how to use them in a program
  - Learn about user-defined functions
  - Examine value-returning functions , including actual and formal parameters
  - Explore how to construct and use a value-returning, user-defined function in a program
  - Learn about function prototypes
  - Learn how to construct and use void functions

© 2018 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part, except for use as permitted in a license distributed with a certain product or service or otherwise on a password-protected website for classroom

Objectives (2 of 2)

- Discover the difference between value and reference parameters
- Explore reference parameters and value-returning functions
- Learn about the scope of an identifier
- Examine the difference between local and global identifiers
- Discover static variables
- Learn how to debug programs using drivers and stubs
- Learn function overloading
- Explore functions with default parameters

CENGAGE Learning

© 2018 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part, except for use as permitted in a license distributed with a certain product or service or otherwise on a password-protected website for classroom

Introduction

- Functions allow complicated programs to be divided into manageable pieces
- Functions are often called modules
- They are like miniature programs that can be combined to form larger programs

© 2018 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part, except for use as permitted in a license distributed with a certain product or service or otherwise on a password-protected website for classroom

Predefined Functions (1 of 2)

- In C++, a function is similar to that of a function in algebra
  - It has a name
  - It does some computation
- Some of the predefined mathematical functions are:

```
pow(x, y)
sqrt(x)
floor(x)
```

© 2018 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part, except for use as permitted in a license distributed with a certain product or service or otherwise on a password-protected website for classroom

Predefined Functions (2 of 2)

- Predefined functions are organized into separate libraries
  - I/O functions are in **`iostream`** header
  - Math functions are in **`cmath`** header
- To use predefined functions, you must include the header file using an **`include`** statement
- See Table 6-1 in the text for some common predefined functions

© 2018 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part, except for use as permitted in a license distributed with a certain product or service or otherwise on a password-protected website for classroom

User-Defined Functions

- Value-returning functions have a return type
  - Return a value of a specific data type using the `return` statement
- Void functions do not have a return type
  - *Do not* use a `return` statement to return a value

© 2018 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part, except for use as permitted in a license distributed with a certain product or service or otherwise on a password-protected website for classroom

Value-Returning Functions (1 of 3)

- To use these functions, you must:
  - Include the appropriate header file in your program using the include statement
  - Know the following items:
    - Name of the function
    - Number of parameters, if any
    - Data type of each parameter
    - Data type of the value returned, called the type of the function

© 2018 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part, except for use as permitted in a license distributed with a certain product or service or otherwise on a password-protected website for classroom

Value-Returning Functions (2 of 3)

- Can use the value returned by a value-returning function by:
  - Saving it for further calculation
  - Using it in some calculation
  - Printing it
- A value-returning function is used:
  - In an assignment
  - As a parameter in a function call
  - In an output statement

© 2018 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part, except for use as permitted in a license distributed with a certain product or service or otherwise on a password-protected website for classroom

Value-Returning Functions (3 of 3)

- Heading (or function header): the first line of the function
  - Example: `int abs(int number)`
- The body is the function's code that accomplishes the task
- Formal parameter: a variable declared in the heading
  - Example: `number`
- Actual parameter: a variable or expression listed in a call to a function
  - Example: `x = pow(u, v)`

© 2018 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part, except for use as permitted in a license distributed with a certain product or service or otherwise on a password-protected website for classroom

Syntax: Value-Returning Function

- Syntax

```
functionType functionName(formal parameter list)
{
    statements
}
```

- **`functionType`** is also called the data type or return type

© 2018 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part, except for use as permitted in a license distributed with a certain product or service or otherwise on a password-protected website for classroom

Syntax: Formal Parameter List

```
dataType identifier, dataType identifier, ...
```

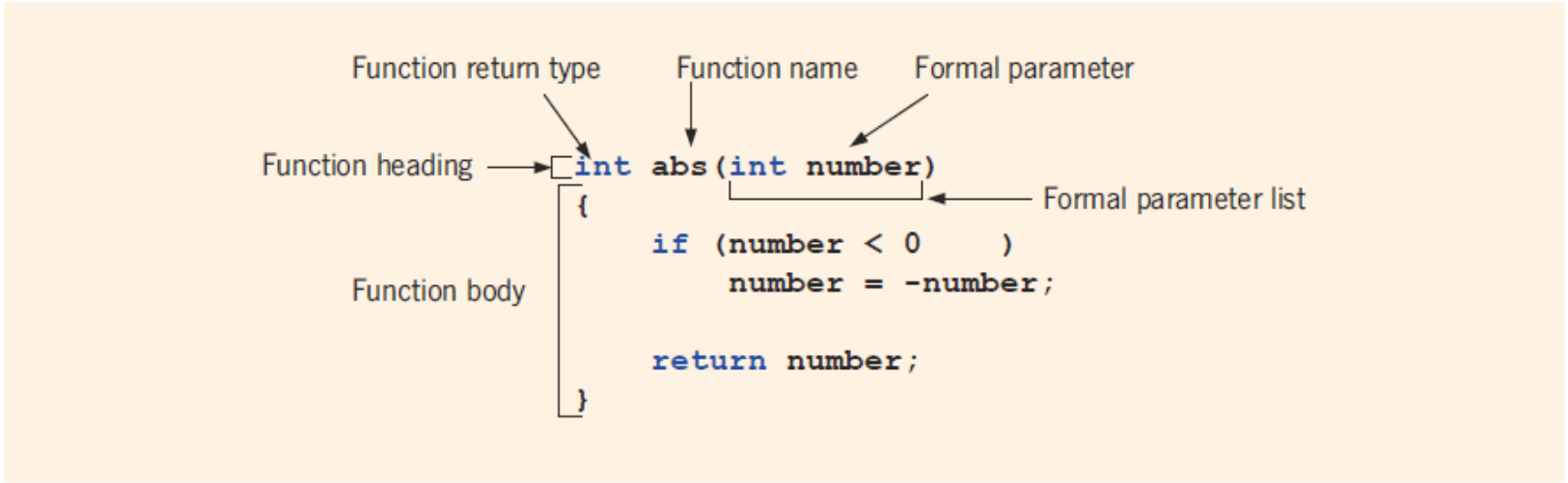


FIGURE 6-1 Various parts of the function **abs**

© 2018 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part, except for use as permitted in a license distributed with a certain product or service or otherwise on a password-protected website for classroom

Function Call

- Syntax to call a value-returning function

```
functionName(actual parameter list)
```

© 2018 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part, except for use as permitted in a license distributed with a certain product or service or otherwise on a password-protected website for classroom

Syntax: Actual Parameter List

- The syntax of the actual parameter list is:

```
expression or variable, expression or variable, ...
```

- The formal parameter list can be empty

```
functionType functionName()
```

- A call to a value-returning function with an empty formal parameter list is:

```
functionName()
```

CENGAGE Learning

© 2018 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part, except for use as permitted in a license distributed with a certain product or service or otherwise on a password-protected website for classroom

`return` Statement

- A function returns its value via the **`return`** statement
  - It passes this value outside the function

CENGAGE Learning

© 2018 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part, except for use as permitted in a license distributed with a certain product or service or otherwise on a password-protected website for classroom

Syntax: `return` Statement (1 of 2)

- The **`return`** statement has this syntax:

```
return expr;
```

- In C++, **`return`** is a reserved word
- When a **`return`** statement executes
  - The function immediately terminates
  - Control goes back to the caller
- When a **`return`** statement executes in the function **`main`**, the program terminates

© 2018 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part, except for use as permitted in a license distributed with a certain product or service or otherwise on a password-protected website for classroom

Syntax: `return` Statement (2 of 2)

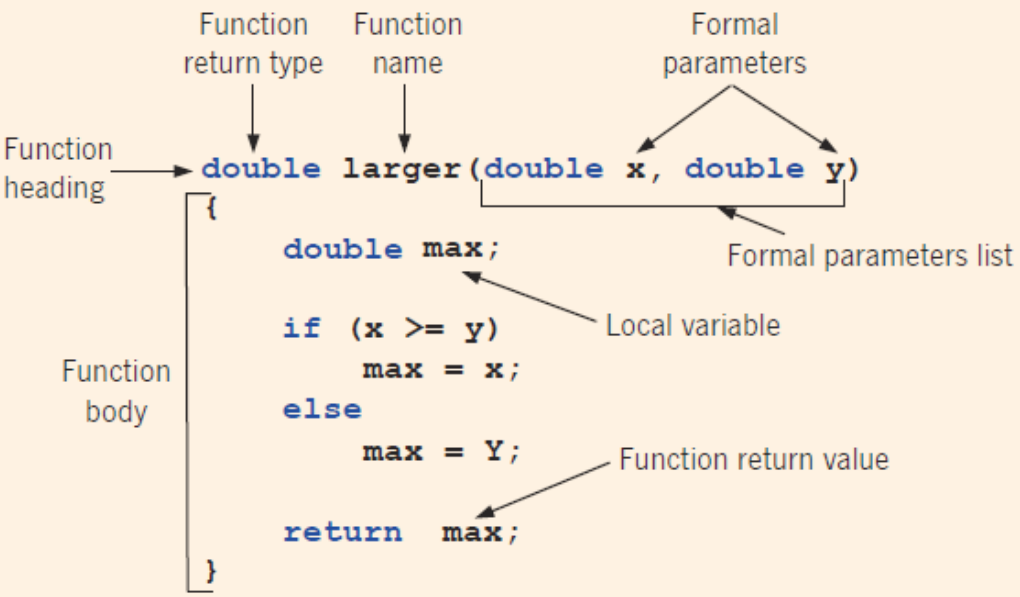


FIGURE 6-2 Various parts of the function **`larger`**

© 2018 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part, except for use as permitted in a license distributed with a certain product or service or otherwise on a password-protected website for classroom

Function Prototype

- A function prototype is the function heading without the body of the function
- The general syntax of the function prototype of a value-returning function is:

```
functionType functionName(parameter list);
```

- It is not necessary to specify the variable name in the parameter list
- The data type of each parameter must be specified

CENGAGE Learning

© 2018 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part, except for use as permitted in a license distributed with a certain product or service or otherwise on a password-protected website for classroom

Value-Returning Functions: Some Peculiarities (1 of 2)

```
int secret(int x)
{
    if (x > 5)      //Line 1
    return 2 * x; //Line 2
}
```

A correct definition of the function **secret** is:

```
int secret(int x)
{
    if (x > 5)              //Line 1
        return 2 * x;       //Line 2
    return x;               //Line 3
}
```

CENGAGE Learning

© 2018 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part, except for use as permitted in a license distributed with a certain product or service or otherwise on a password-protected website for classroom

Value-Returning Functions: Some Peculiarities (2 of 2)

- Examples pointing out that the **return** statement only returns one value

```
return x, y; //only the value of y will be returned


int funcRet1()
{
    int x = 45;
    return 23, x; //only the value of x is returned
}
int funcRet2(int z)
{
    int a = 2;
    int b = 3;
    return 2 * a + b, z + b;
                //only the value of z + b is returned
}
```

CENGAGE Learning

© 2018 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part, except for use as permitted in a license distributed with a certain product or service or otherwise on a password-protected website for classroom

More Examples of Value-Returning Functions (1 of 2)

```
char courseGrade(int score)
{
    switch (score / 10)
    {
    case 0:
    case 1:
    case 2:
    case 3:
    case 4:
    case 5:
        return 'F';
    case 6:
        return 'D';
    case 7:
        return 'C';
    case 8:
        return 'B';
    case 9:
    case 10:
        return 'A';
    }
}
```

CENGAGE Learning

© 2018 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part, except for use as permitted in a license distributed with a certain product or service or otherwise on a password-protected website for classroom

More Examples of Value-Returning Functions (2 of 2)

- In addition to Example 6-3 **`courseGrade`**, other examples are given in the text
  - Example 6-4 (rolling a pair of dice)
  - Example 6-5 (Fibonacci number)
  - Example 6-6 (palindrome)
  - Example 6-7 (cable company)

CENGAGE Learning

© 2018 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part, except for use as permitted in a license distributed with a certain product or service or otherwise on a password-protected website for classroom

Flow of Compilation and Execution (1 of 2)

- Execution always begins at the first statement in the function **`main`**
- Other functions are executed only when called
- Function prototypes appear before any function definition
  - The compiler translates these first
- The compiler can then correctly translate a function call

© 2018 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part, except for use as permitted in a license distributed with a certain product or service or otherwise on a password-protected website for classroom

Flow of Compilation and Execution (2 of 2)

- A function call transfers control to the first statement in the body of the called function
- When the end of a called function is executed, control is passed back to the point immediately following the function call
  - A function's returned value replaces the function call statement

CENGAGE Learning

© 2018 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part, except for use as permitted in a license distributed with a certain product or service or otherwise on a password-protected website for classroom

Void Functions (1 of 4)

- User-defined void functions can be placed either before or after the function **`main`**
- If user-defined void functions are placed after the function **`main`**
  - The function prototype must be placed before the function **`main`**
- A void function does not have a return type
  - A **`return`** statement without any value is typically used to exit the function early

© 2018 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part, except for use as permitted in a license distributed with a certain product or service or otherwise on a password-protected website for classroom

Void Functions (2 of 4)

- Formal parameters are optional
- A call to a void function is a stand-alone statement
- The function definition of void functions with parameters has the following syntax:

```
void functionName(formal parameter list)
{
    statements
}
```

© 2018 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part, except for use as permitted in a license distributed with a certain product or service or otherwise on a password-protected website for classroom

Void Functions (3 of 4)

- Formal parameter list syntax

```
dataType& variable, dataType& variable, ...
```

- Function call syntax

```
functionName(actual parameter list);
```

- Actual parameter list syntax

```
expression or variable, expression or variable, ...
```

© 2018 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part, except for use as permitted in a license distributed with a certain product or service or otherwise on a password-protected website for classroom

Void Functions (4 of 4)

- Two types of formal parameters
  - Value parameter: a formal parameter that receives a copy of the content of corresponding actual parameter
  - Reference parameter: a formal parameter that receives the location (memory address) of the corresponding actual parameter

© 2018 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part, except for use as permitted in a license distributed with a certain product or service or otherwise on a password-protected website for classroom

Value Parameters

- If a formal parameter is a value parameter, the value of the corresponding actual parameter is copied into it
  - A formal parameter has its own copy of the data
- During program execution, a formal parameter manipulates the data stored in its own memory space

© 2018 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part, except for use as permitted in a license distributed with a certain product or service or otherwise on a password-protected website for classroom

Reference Variables as Parameters (1 of 2)

- If a formal parameter is a reference parameter:
  - It receives the memory address of the corresponding actual parameter
- During program execution to manipulate data:
  - Changes to a formal parameter will change the corresponding actual parameter

CENGAGE Learning

© 2018 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part, except for use as permitted in a license distributed with a certain product or service or otherwise on a password-protected website for classroom

Reference Variables as Parameters (2 of 2)

- Reference parameters are useful in three situations:
  - When changing the actual parameter
  - When returning more than one value
  - When passing the address would save memory space and time

© 2018 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part, except for use as permitted in a license distributed with a certain product or service or otherwise on a password-protected website for classroom

Value and Reference Parameters and Memory Allocation (1 of 2)

- When a function is called:
  - Memory for its formal parameters and its local variables is allocated in the function data area
- For a value parameter, the actual parameter's value is copied into the formal parameter's memory cell
  - Changes to the formal parameter do not affect the actual parameter's value

© 2018 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part, except for use as permitted in a license distributed with a certain product or service or otherwise on a password-protected website for classroom

Value and Reference Parameters and Memory Allocation (2 of 2)

- For a reference parameter, the actual parameter's address passes to the formal parameter
  - Both formal and actual parameters refer to the same memory location
  - During execution, any change made to the formal parameter's value immediately changes the actual parameter's value

© 2018 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part, except for use as permitted in a license distributed with a certain product or service or otherwise on a password-protected website for classroom

Reference Parameters and Value-Returning Functions

- Can also use reference parameters in a value-returning function
  - Not recommended
- By definition, a value-returning function returns a single value via **`return`** statement
- If a function needs to return more than one value, change it to a void function and use reference parameters to return the values

© 2018 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part, except for use as permitted in a license distributed with a certain product or service or otherwise on a password-protected website for classroom

Scope of an Identifier

- Scope of an identifier: where in the program the identifier is accessible
- Local identifier: identifiers declared within a function (or block)
- Global identifier: identifiers declared outside of every function definition
- C++ does not allow nested functions
  - Definition of one function cannot be included in the body of another function

© 2018 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part, except for use as permitted in a license distributed with a certain product or service or otherwise on a password-protected website for classroom

Rules Applied When an Identifier is Accessed (1 of 2)

- Global identifiers are accessible by a function or block if:
  - The identifier is declared before the function definition (block)
  - The function name different from the identifier
  - All parameters to the function have different names than the identifier name
  - All local identifiers have different names than the identifier name

© 2018 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part, except for use as permitted in a license distributed with a certain product or service or otherwise on a password-protected website for classroom

Rules Applied When an Identifier is Accessed (2 of 2)

- (Nested block) – an identifier declared within a block is accessible:
  - From its point of declaration to the end of the block in which it is declared
  - Within nested blocks if no identifier with the same name exists
- The scope of a function name is similar to the scope of an identifier declared outside any block
  - The function name scope is the same as the global variable scope

© 2018 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part, except for use as permitted in a license distributed with a certain product or service or otherwise on a password-protected website for classroom

Other Notes about Global Variables

- Some compilers initialize global variables to default values
- The scope resolution operator in C++ is `::`
- By using the scope resolution operator:
  - A global variable declared before the definition of a function (or block) can be accessed by the function (or block) even if the function (or block) has an identifier with the same name as the global variable
- To access a global variable declared after the definition of a function, the function must not contain any identifier with the same name
  - Reserved word **`extern`** indicates that a global variable has been declared elsewhere

CENGAGE Learning

© 2018 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part, except for use as permitted in a license distributed with a certain product or service or otherwise on a password-protected website for classroom

Global Variables, Named Constants, and Side Effects

- Using global variables causes side effects
- A function that uses global variables is not independent
- If more than one function uses the same global variable:
  - It can be difficult to debug problems with the code
  - Problems caused in one area of the program may appear to be from another area
- Global named constants have no side effects

© 2018 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part, except for use as permitted in a license distributed with a certain product or service or otherwise on a password-protected website for classroom

Static and Automatic Variables (1 of 2)

- Automatic variable: memory is allocated at block entry and deallocated at block exit
  - By default, variables declared within a block are automatic variables
- Static variable: memory remains allocated as long as the program executes
  - Global variables declared outside of any block are static variables

CENGAGE Learning

© 2018 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part, except for use as permitted in a license distributed with a certain product or service or otherwise on a password-protected website for classroom

Static and Automatic Variables (2 of 2)

- Declare a static variable within a block by using the reserved word **`static`**
- The syntax for declaring a static variable is:

```
static dataType identifier;
```

- Static variables declared within a block are local to the block
  - Have same scope as any other local identifier in that block

© 2018 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part, except for use as permitted in a license distributed with a certain product or service or otherwise on a password-protected website for classroom

Debugging: Using Drivers and Stubs

- A driver program is a separate program used to test a function
- When results calculated by one function are needed in another function, use a function stub
- A function stub is a function that is not fully coded

© 2018 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part, except for use as permitted in a license distributed with a certain product or service or otherwise on a password-protected website for classroom

Function Overloading: An Introduction

- In a C++ program, several functions can have the same name
- Function overloading (or overloading a function name) occurs when creating several functions with the same name
- Two functions are said to have different formal parameter lists if both functions have either:
  - A different number of formal parameters
  - If the number of formal parameters is the same, but the data type of the formal parameters differs in at least one position

© 2018 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part, except for use as permitted in a license distributed with a certain product or service or otherwise on a password-protected website for classroom

Function Overloading

- Overloaded functions must have different formal parameter lists
- The signature: the name and formal parameter list of the function
  - Does not include the return type of the function
- The parameter list supplied in a call to an overloaded function determines which function is executed

CENGAGE Learning

© 2018 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part, except for use as permitted in a license distributed with a certain product or service or otherwise on a password-protected website for classroom

Functions with Default Parameters (1 of 2)

- In a function call, the number of actual and formal parameters must be the same
  - C++ relaxes this condition for functions with default parameters
- Can specify the value of a default parameter in the function prototype
- If you do not specify the value for a default parameter when calling the function, the default value is used

CENGAGE Learning

© 2018 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part, except for use as permitted in a license distributed with a certain product or service or otherwise on a password-protected website for classroom

Functions with Default Parameters (2 of 2)

- All default parameters must be the rightmost parameters of the function
- If a default parameter value is not specified:
  - You must omit all of the arguments to its right
- Default values can be constants, global variables, or function calls
- Cannot assign a constant value as a default value to a reference parameter

© 2018 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part, except for use as permitted in a license distributed with a certain product or service or otherwise on a password-protected website for classroom

Quick Review (1 of 4)

- Functions (modules) divide a program into manageable tasks
- C++ provides standard, predefined functions
- Two types of user-defined functions: value-returning functions and void functions
- Variables defined in a function heading are called formal parameters
- Expressions, variables, or constant values in a function call are called actual parameters

© 2018 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part, except for use as permitted in a license distributed with a certain product or service or otherwise on a password-protected website for classroom

Quick Review (2 of 4)

- Function heading and the body of the function are called the definition of the function
- A value-returning function returns its value via the `return` statement
- A prototype is the function heading without the body of the function
- User-defined functions execute only when they are called
- Void functions do not have a data type

© 2018 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part, except for use as permitted in a license distributed with a certain product or service or otherwise on a password-protected website for classroom

Quick Review (3 of 4)

- There are two types of formal parameters
  - A value parameter receives a copy of its corresponding actual parameter
  - A reference parameter receives the memory address of its corresponding actual parameter
- Variables declared within a function (or block) are called local variables
- Variables declared outside of every function definition (and block) are global variables

© 2018 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part, except for use as permitted in a license distributed with a certain product or service or otherwise on a password-protected website for classroom

Quick Review (4 of 4)

- An automatic variable is a variable for which memory is allocated on function/block entry and deallocated on function/block exit
- A static variable is a variable for which memory remains allocated throughout the execution of the program
- C++ functions can have default parameters

© 2018 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part, except for use as permitted in a license distributed with a certain product or service or otherwise on a password-protected website for classroom

Reading Assignment – Very Important for "GU – ECE 111"

- Malik, D.S., 2014. **C++ programming: Program design including data structures**. Cengage Learning.
  - "**Chapter 6**: **User-Defined Functions**".

CENGAGE Learning

© 2018 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part, except for use as permitted in a license distributed with a certain product or service or otherwise on a password-protected website for classroom