C++ Programming: From Problem Analysis to Program Design, Fourth Edition

Chapter 6: User-Defined Functions I

Objectives

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

Introduction

- Functions are like building blocks
- They allow complicated programs to be divided into manageable pieces
- Some advantages of functions:
  - A programmer can focus on just that part of the program and construct it, debug it, and perfect it
  - Different people can work on different functions simultaneously
  - Can be re-used (even in different programs)
  - Enhance program readability

Introduction (continued)

- Functions
  - Called modules
  - Like miniature programs
  - Can be put together to form a larger program

Predefined Functions

- In algebra, a function is defined as a rule or correspondence between values, called the function's arguments, and the unique value of the function associated with the arguments
  - If `f(x) = 2x + 5`, then `f(1) = 7`, `f(2) = 9`, and `f(3) = 11`
    - `1`, `2`, and `3` are arguments
    - `7`, `9`, and `11` are the corresponding values

Predefined Functions (continued)

- Some of the predefined mathematical functions are:

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

- Predefined functions are organized into separate libraries
- I/O functions are in `iostream` header
- Math functions are in `cmath` header

Predefined Functions (continued)

- `pow(x,y)` calculates x^y
  - `pow(2, 3) = 8.0`
  - Returns a value of type `double`
  - `x` and `y` are the parameters (or arguments)
    - The function has two parameters
- `sqrt(x)` calculates the nonnegative square root of `x`, for `x >= 0.0`
  - `sqrt(2.25)` is `1.5`
  - Type `double`

Predefined Functions (continued)

- The `floor` function `floor(x)` calculates largest whole number not greater than `x`
  - `floor(48.79)` is `48.0`
  - Type `double`
  - Has only one parameter

Predefined Functions (continued)

TABLE 6-1 Predefined Functions

Function	Header File	Purpose	Parameter(s) Type	Result
`abs(x)`	`<cstdlib>`	Returns the absolute value of its argument: `abs(-7) = 7`	`int`	`int`
`ceil(x)`	`<cmath>`	Returns the smallest whole number that is not less than x: `ceil(56.34) = 57.0`	`double`	`double`
`cos(x)`	`<cmath>`	Returns the cosine of angle x: `cos(0.0) = 1.0`	`double` (radians)	`double`
`exp(x)`	`<cmath>`	Returns e^x, where e = 2.718: `exp(1.0) = 2.71828`	`double`	`double`
`fabs(x)`	`<cmath>`	Returns the absolute value of its argument: `fabs(-5.67) = 5.67`	`double`	`double`

Predefined Functions (continued)

TABLE 6-1 Predefined Functions (continued)

Function	Header File	Purpose	Parameter(s) Type	Result
`floor(x)`	`<cmath>`	Returns the largest whole number that is not greater than x: `floor(45.67) = 45.00`	`double`	`double`
`pow(x, y)`	`<cmath>`	Returns x^y; If `x` is negative, `y` must be a whole number: `pow(0.16, 0.5) = 0.4`	`double`	`double`
`tolower(x)`	`<cctype>`	Returns the lowercase value of `x` if `x` is uppercase; otherwise, returns `x`	`int`	`int`
`toupper(x)`	`<cctype>`	Returns the uppercase value of `x` if `x` is lowercase; otherwise, returns `x`	`int`	`int`

EXAMPLE 6-1

```
//How to use predefined functions.

#include <iostream>
#include <cmath>
#include <cctype>
#include <cstdlib>

using namespace std;

int main()
{
    int    x;
    double u, v;

    cout << "Line 1: Uppercase a is "
         << static_cast<char>(toupper('a'))
         << endl;                                    //Line 1

    u = 4.2;                                         //Line 2
    v = 3.0;                                         //Line 3
    cout << "Line 4: " << u << " to the power of "
         << v << " = " << pow(u, v) << endl;         //Line 4

    cout << "Line 5: 5.0 to the power of 4 = "
         << pow(5.0, 4) << endl;                     //Line 5

    u = u + pow(3.0, 3);                             //Line 6
    cout << "Line 7: u = " << u << endl;             //Line 7

    x = -15;                                         //Line 8
    cout << "Line 9: Absolute value of " << x
         << " = " << abs(x) << endl;                 //Line 9

    return 0;
}
```

Predefined Functions (continued)

- Example 6-1 sample run:

```
Line 1: Uppercase a is A
Line 4: 4.2 to the power of 3 = 74.088
Line 5: 5.0 to the power of 4 = 625
Line 7: u = 31.2
Line 9: Absolute value of -15 = 15
```

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

Value-Returning Functions

- 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

Value-Returning Functions (continued)

- Because the value returned by a value-returning function is unique, we must:
  - Save the value for further calculation
  - Use the value in some calculation
  - Print the value
- A value-returning function is used in an assignment or in an output statement
- One more thing is associated with functions:
  - The code required to accomplish the task

Value-Returning Functions (continued)

```
int abs(int number)
int abs(int number)
{
    if (number < 0)
        number = -number;

    return number;
}

double pow(double base, double exponent)

double u = 2.5;
double v = 3.0;
double x, y, w;

x = pow(u, v);              //Line 1
y = pow(2.0, 3.2);          //Line 2
w = pow(u, 7);              //Line 3
```

Value-Returning Functions (continued)

- Heading: first four properties above
  - Example: `int abs(int number)`
- Formal Parameter: variable declared in the heading
  - Example: `number`
- Actual Parameter: variable or expression listed in a call to a function
  - Example: `x = pow(u, v)`

Syntax: Value-Returning Function

- Syntax:

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

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

Syntax: Formal Parameter List

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

Function Call

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

Syntax: Actual Parameter List

- The syntax of the actual parameter list is:

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

- Formal parameter list can be empty:

```
functionType functionName()
```

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

```
functionName()
```

`return` Statement

- Once a value-returning function computes the value, the function returns this value via the `return` statement
  - It passes this value outside the function via the `return` statement

Syntax: `return` Statement

- The `return` statement has the following syntax:

```
return expr;
```

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

```
double larger(double x, double y)
{
    double max;

    if (x >= y)
        max = x;
    else
        max = y;

    return max;
}
```

You can also write this function as follows:

```
double larger(double x, double y)
{
    if (x >= y)
        return x;
    else
        return y;
}
```

```
double larger(double x, double y)
{
    if (x >= y)
        return x;

    return y;
}
```

NOTE

1. In the definition of the function `larger`, `x` and `y` are formal parameters.
2. The `return` statement can appear anywhere in the function. Recall that once a `return` statement executes, all subsequent statements are skipped. Thus, it's a good idea to return the value as soon as it is computed.

Function Prototype

- <u>Function prototype</u>: function heading without the body of the function
- Syntax:

```
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

```
//Program: Largest of three numbers

#include <iostream>

using namespace std;

double larger(double x, double y);
double compareThree(double x, double y, double z);

int main()
{
    double one, two;                                    //Line 1

    cout << "Line 2: The larger of 5 and 10 is "
         << larger(5, 10) << endl;                      //Line 2

    cout << "Line 3: Enter two numbers: ";              //Line 3
    cin >> one >> two;                                  //Line 4
    cout << endl;                                       //Line 5

    cout << "Line 6: The larger of " << one
         << " and " << two << " is "
         << larger(one, two) << endl;                   //Line 6

    cout << "Line 7: The largest of 23, 34, and "
         << "12 is " << compareThree(23, 34, 12)
         << endl;                                       //Line 7

    return 0;
}
```

Function Prototype (continued)

```
double larger(double x, double y)
{
    if (x >= y)
        return x;
    else
        return y;
}

double compareThree (double x, double y, double z)
{
    return larger(x, larger(y, z));
}
```

Sample Run: In this sample run, the user input is shaded.

```
Line 2: The larger of 5 and 10 is 10
Line 3: Enter two numbers: 25 73

Line 6: The larger of 25 and 73 is 73
Line 7: The largest of 23, 34, and 12 is 34
```

Palindrome Number

- A nonnegative integer is a palindrome if it reads forward and backward in the same way
  - Examples: 5, 44, 789656987

```
bool isNumPalindrome(int num)
{
    int pwr = 0;

    if (num < 10)                                          //Step 1
        return true;
    else                                                   //Step 2
    {
                                                           //Step 2.a
        while (num / static_cast<int>(pow(10.0, pwr)) >= 10)
            pwr++;
```

Palindrome Number (continued)

```
        while (num >= 10)                               //Step 2.b
        {
            int tenTopwr = static_cast<int>(pow(10.0, pwr));

            if ((num / tenTopwr) != (num % 10))
                return false;                           //Step 2.b.1
            else                                        //Step 2.b.2
            {
                num = num % tenTopwr;                   //Step 2.b.2.1
                num = num / 10;                         //Step 2.b.2.1
                pwr = pwr - 2;                          //Step 2.b.2.2
            }
        }//end while

        return true;
    }//end else
}
```

Flow of Execution

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

Flow of Execution (continued)

- A function call results in transfer of control to the first statement in the body of the called function
- After the last statement of a function is executed, control is passed back to the point immediately following the function call
- A value-returning function returns a value
  - After executing the function the returned value replaces the function call statement

Programming Example: Largest Number

- The function `larger` is used to determine the largest number from a set of numbers
- Program determines the largest number from a set of 10 numbers
- <u>Input</u>: a set of 10 numbers
- <u>Output</u>: the largest of 10 numbers

Programming Example: Program Analysis

- Suppose that the input data is:

  `15 20 7 8 28 21 43 12 35 3`

- Read the first number of the data set
  - Because this is the only number read to this point, you may assume that it is the largest number so far and call it `max`
- Read the second number and call it `num`
  - Compare `max` and `num`, and store the larger number into `max`

Programming Example: Program Analysis (continued)

- Now `max` contains the larger of the first two numbers
- Read the third number and compare it with `max` and store the larger number into `max`
  - `max` contains the largest of the first three numbers
- Read the next number, compare it with `max`, and store the larger into `max`
- Repeat this process for each remaining number in the data set

Programming Example: Algorithm Design

- Read the first number
  - Because this is the only number that you have read, it is the largest number so far
  - Save it in a variable called `max`
- For each remaining number in the list
  - Read the next number
  - Store it in a variable called `num`
  - Compare `num` and `max`

Programming Example: Algorithm Design (continued)

- For each remaining number in the list (continued)
  - If `max < num`
    - `num` is the new largest number
    - update the value of `max` by copying `num` into `max`
  - If `max >= num`, discard `num`; that is, do nothing
- Because `max` now contains the largest number, print it

Summary

- Functions (modules) are miniature programs
  - Divide a program into manageable tasks
- C++ provides the standard 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

Summary (continued)

- In a function call, the number of actual parameters and their types must match with the formal parameters in the order given
- To call a function, use its name together with the actual parameter list
- Function heading and the body of the function are called the definition of the function
- If a function has no parameters, you need empty parentheses in heading and call
- A value-returning function returns its value via the `return` statement

Summary (continued)

- A prototype is the function heading without the body of the function; prototypes end with the semicolon
- Prototypes are placed before every function definition, including `main`
- User-defined functions execute only when they are called
- In a call statement, specify only the actual parameters, not their data types