6

Functions and an Introduction to Recursion

Form ever follows function.

— Louis Henri Sullivan

E pluribus unum. (One composed of many.)

— Virgil

O! call back yesterday, bid time return.

— William Shakespeare

Call me Ishmael.

— Herman Melville

When you call me that, smile!

— Owen Wister

Answer me in one word.

— William Shakespeare

There is a point at which methods devour themselves.

— Frantz Fanon

Life can only be understood backwards; but it must be lived forwards.

— Soren Kierkegaard

OBJECTIVES

In this chapter you will learn:

- To construct programs modularly from functions.
- To use common math functions available in the C++ Standard Library.
- To create functions with multiple parameters.
- The mechanisms for passing information between functions and returning results.
- How the function call/return mechanism is supported by the function call stack and activation records.
- To use random number generation to implement game-playing applications.
- How the visibility of identifiers is limited to specific regions of programs.
- To write and use recursive functions, i.e., functions that call themselves.

Outline

Outline

6.1 Introduction

- Divide and conquer technique
  - Construct a large program from small, simple pieces (e.g., components)
- Functions
  - Facilitate the design, implementation, operation and maintenance of large programs
- C++ Standard Library math functions

6.2 Program Components in C++

- **C++ Standard Library**
  - **Rich collection of functions for performing common operations, such as:**
    - **Mathematical calculations**
    - **String manipulations**
    - **Character manipulations**
    - **Input/Output**
    - **Error checking**
  - **Provided as part of the C++ programming environment**

Software Engineering Observation 6.1

Read the documentation for your compiler to familiarize yourself with the functions and classes in the C++ Standard Library.

6.2 Program Components in C++ (Cont.)

- **Functions**
  - **Called methods or procedures in other languages**
  - **Allow programmers to modularize a program by separating its tasks into self-contained units**
    - **Statements in function bodies are written only once**
      - **Reused from perhaps several locations in a program**
      - **Hidden from other functions**
      - **Avoid repeating code**
    - **Enable the divide-and-conquer approach**
    - **Reusable in other programs**
    - **User-defined or programmer-defined functions**

Software Engineering Observation 6.2

To promote software reusability, every function should be limited to performing a single, well-defined task, and the name of the function should express that task effectively. Such functions make programs easier to write, test, debug and maintain.

Error-Prevention Tip 6.1

A small function that performs one task is easier to test and debug than a larger function that performs many tasks.

Software Engineering Observation 6.3

If you cannot choose a concise name that expresses a function's task, your function might be attempting to perform too many diverse tasks. It is usually best to break such a function into several smaller functions.

6.2 Program Components in C++ (cont.)

- **Function (Cont.)**
  - **A function is invoked by a function call**
    - **Called function either returns a result or simply returns to the caller**
    - **Function calls form hierarchical relationships**

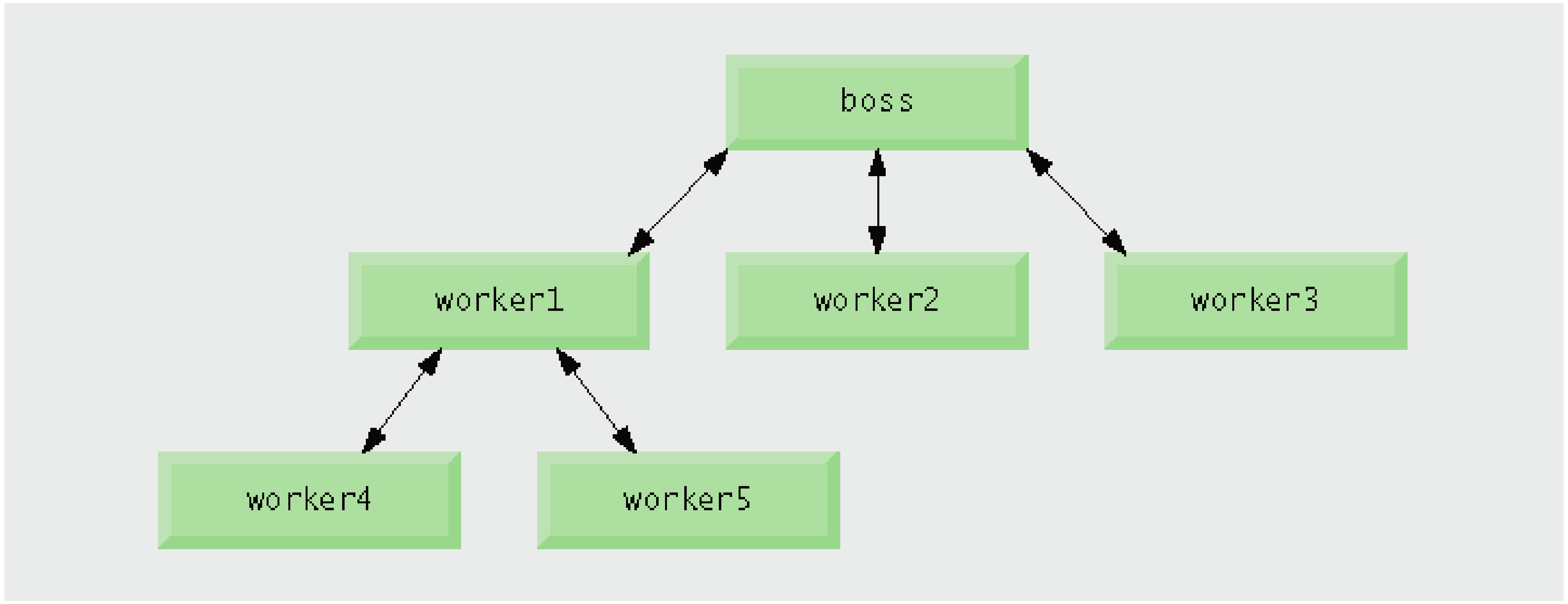

Fig. 6.1 | **Hierarchical boss function/worker function relationship.**

6.3 Math Library Functions

- **Global functions**
  - **Do not belong to a particular class**
  - **Have function prototypes placed in header files**
    - **Can be reused in any program that includes the header file and that can link to the function's object code**
  - **Example: `sqrt` in `<cmath>` header file**
    - `sqrt( 900.0 )`
    - **All functions in `<cmath>` are global functions**

Function	Description	Example

`ceil( x )`	rounds *x* to the smallest integer not less than *x*	`ceil( 9.2 )` is `10.0`
		`ceil( -9.8 )` is `-9.0`

`cos( x )`	trigonometric cosine of *x* (*x* in radians)	`cos( 0.0 )` is `1.0`

`exp( x )`	exponential function e^x	`exp( 1.0 )` is `2.71828`
		`exp( 2.0 )` is `7.38906`

`fabs( x )`	absolute value of *x*	`fabs( 5.1 )` is `5.1`
		`fabs( 0.0 )` is `0.0`
		`fabs( -8.76 )` is `8.76`

`floor( x )`	rounds *x* to the largest integer not greater than *x*	`floor( 9.2 )` is `9.0`
		`floor( -9.8 )` is `-10.0`

`fmod( x, y )`	remainder of *x/y* as a floating-point number	`fmod( 2.6, 1.2 )` is `0.2`

`log( x )`	natural logarithm of *x* (base *e*)	`log( 2.718282 )` is `1.0`
		`log( 7.389056 )` is `2.0`

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

`pow( x, y )`	*x* raised to power *y* (x^y)	`pow( 2, 7 )` is `128`
		`pow( 9, .5 )` is `3`

`sin( x )`	trigonometric sine of *x* (*x* in radians)	`sin( 0.0 )` is `0`
`sqrt( x )`	square root of *x* (where *x* is a nonnegative value)	`sqrt( 9.0 )` is `3.0`
`tan( x )`	trigonometric tangent of *x* (*x* in radians)	`tan( 0.0 )` is `0`

Fig. 6.2 | Math library functions.

6.4 Function Definitions with Multiple Parameters

- **Multiple parameters**
  - **Functions often require more than one piece of information to perform their tasks**
  - **Specified in both the function prototype and the function header as a comma-separated list of parameters**

Outline

GradeBook.h

(1 of 1)

```
// Fig. 6.3: GradeBook.h
// Definition of class GradeBook that finds the maximum of three grades.
// Member functions are defined in GradeBook.cpp
#include <string> // program uses C++ standard string class
using std::string;

// GradeBook class definition
class GradeBook
{
public:
   GradeBook( string ); // constructor initializes course name
   void setCourseName( string ); // function to set the course name
   string getCourseName(); // function to retrieve the course name
   void displayMessage(); // display a welcome message
   void inputGrades(); // input three grades from user
   void displayGradeReport(); // display a report based on the grades
   int maximum( int, int, int ); // determine max of 3 values
private:
   string courseName; // course name for this GradeBook
   int maximumGrade; // maximum of three grades
}; // end class GradeBook
```

Prototype for a member function that takes three arguments

Data member to store maximum grade

Outline

```
// Fig. 6.4: GradeBook.cpp
// Member-function definitions for class GradeBook that
// determines the maximum of three grades.
#include <iostream>
using std::cout;
using std::cin;
using std::endl;

#include "GradeBook.h" // include definition of class GradeBook

// constructor initializes courseName with string supplied as argument;
// initializes studentMaximum to 0
GradeBook::GradeBook( string name )
{
   setCourseName( name ); // validate and store courseName
   maximumGrade = 0; // this value will be replaced by the maximum grade
} // end GradeBook constructor

// function to set the course name; limits name to 25 or fewer characters
void GradeBook::setCourseName( string name )
{
   if ( name.length() <= 25 ) // if name has 25 or fewer characters
      courseName = name; // store the course name in the object
   else // if name is longer than 25 characters
   { // set courseName to first 25 characters of parameter name
      courseName = name.substr( 0, 25 ); // select first 25 characters
      cout << "Name \"" << name << "\" exceeds maximum length (25).\n"
         << "Limiting courseName to first 25 characters.\n" << endl;
   } // end if...else
} // end function setCourseName
```

GradeBook.cpp

(1 of 3)

Outline

GradeBook.cpp

(2 of 3)

```

// function to retrieve the course name
string GradeBook::getCourseName()
{
   return courseName;
} // end function getCourseName

// display a welcome message to the GradeBook user
void GradeBook::displayMessage()
{
   // this statement calls getCourseName to get the
   // name of the course this GradeBook represents
   cout << "Welcome to the grade book for\n" << getCourseName() << "!\n"
      << endl;
} // end function displayMessage

// input three grades from user; determine maximum
void GradeBook::inputGrades()
{
   int grade1; // first grade entered by user
   int grade2; // second grade entered by user
   int grade3; // third grade entered by user

   cout << "Enter three integer grades: ";
   cin >> grade1 >> grade2 >> grade3;

   // store maximum in member studentMaximum
   maximumGrade = maximum( grade1, grade2, grade3 );
} // end function inputGrades
```

Call to function **`maximum`** passes three arguments

Outline

k.cpp

(3 of 3)

```
// returns the maximum of its three integer parameters
int GradeBook::maximum( int x, int y, int z )
{
   int maximumValue = x; // assume x is the largest to start

   // determine whether y is greater than maximumValue
   if ( y > maximumValue )
      maximumValue = y; // make y the new maximumValue

   // determine whether z is greater than maximumValue
   if ( z > maximumValue )
      maximumValue = z; // make z the new maximumValue

   return maximumValue;
} // end function maximum

// display a report based on the grades entered by user
void GradeBook::displayGradeReport()
{
   // output maximum of grades entered
   cout << "Maximum of grades entered: " << maximumGrade << endl;
} // end function displayGradeReport
```

`maximum` member function header

Comma-separated parameter list

Returning a value to the caller

Outline

fig06_05.cpp

(1 of 1)

```
// Fig. 6.5: fig06_05.cpp
// Create GradeBook object, input grades and display grade report.
#include "GradeBook.h" // include definition of class GradeBook

int main()
{
   // create GradeBook object
   GradeBook myGradeBook( "CS101 C++ Programming" );

   myGradeBook.displayMessage(); // display welcome message
   myGradeBook.inputGrades(); // read grades from user
   myGradeBook.displayGradeReport(); // display report based on grades
   return 0; // indicate successful termination
} // end main
```

```
Welcome to the grade book for
CS101 C++ Programming!

Enter three integer grades: 86 67 75
Maximum of grades entered: 86
```

```
Welcome to the grade book for
CS101 C++ Programming!

Enter three integer grades: 67 86 75
Maximum of grades entered: 86
```

```
Welcome to the grade book for
CS101 C++ Programming!

Enter three integer grades: 67 75 86
Maximum of grades entered: 86
```

Software Engineering Observation 6.4

The commas used in line 58 of Fig. 6.4 to separate the arguments to function `maximum` are not comma operators as discussed in Section 5.3. The comma operator guarantees that its operands are evaluated left to right. The order of evaluation of a function's arguments, however, is not specified by the C++ standard. Thus, different compilers can evaluate function arguments in different orders. The C++ standard does guarantee that all arguments in a function call are evaluated before the called function executes.

6.4 Function Definitions with Multiple Parameters (Cont.)

- Compiler uses a function prototype to:
  - Check that calls to the function contain the correct number and types of arguments in the correct order
  - Ensure that the value returned by the function is used correctly in the expression that called the function
- Each argument must be consistent with the type of the corresponding parameter
  - Parameters are also called formal parameters

Portability Tip 6.1

Sometimes when a function’s arguments are more involved expressions, such as those with calls to other functions, the order in which the compiler evaluates the arguments could affect the values of one or more of the arguments. If the evaluation order changes between compilers, the argument values passed to the function could vary, causing subtle logic errors.

Error-Prevention Tip 6.2

If you have doubts about the order of evaluation of a function's arguments and whether the order would affect the values passed to the function, evaluate the arguments in separate assignment statements before the function call, assign the result of each expression to a local variable, then pass those variables as arguments to the function.

Common Programming Error 6.1

Declaring method parameters of the same type as `double x, y` instead of `double x, double y` is a syntax error—an explicit type is required for each parameter in the parameter list.

Common Programming Error 6.2

Compilation errors occur if the function prototype, function header and function calls do not all agree in the number, type and order of arguments and parameters, and in the return type.

Software Engineering Observation 6.5

A function that has many parameters may be performing too many tasks. Consider dividing the function into smaller functions that perform the separate tasks. Limit the function header to one line if possible.

6.4 Function Definitions with Multiple Parameters (Cont.)

- **Three ways to return control to the calling statement:**
  - **If the function does not return a result:**
    - **Program flow reaches the function-ending right brace or**
    - **Program executes the statement `return;`**
  - **If the function does return a result:**
    - **Program executes the statement `return` *expression*`;`**
      - ***expression* is evaluated and its value is returned to the caller**