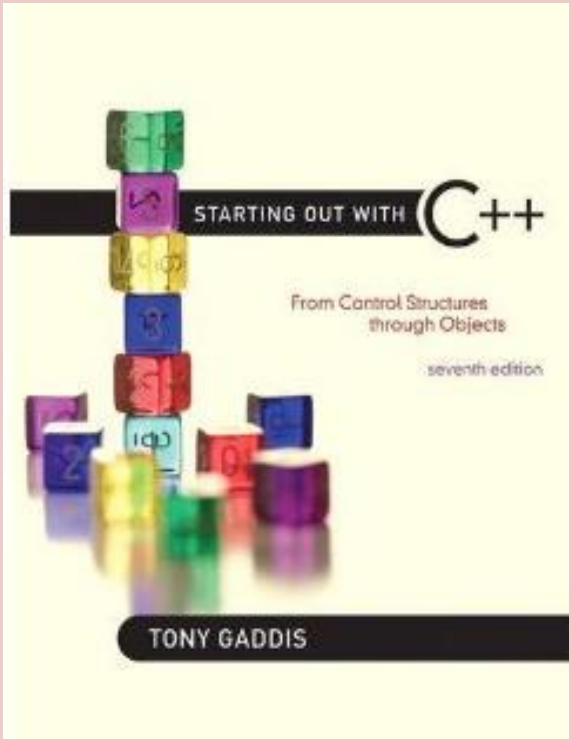
Software Design & Programming I



Starting Out with C++ (From Control Structures through Objects) 7th Edition
Written by: Tony Gaddis
Pearson - Addison Wesley
ISBN: 13-978-0-132-57625-3

Chapter 6

Functions

Modular Programming

A program may be broken up into a set of manageable functions, or modules. This is called modular programming.

A function is a collection of statements that performs a specific task. So far you have used functions in two ways: 1) you have created a function called main in every program you've written, and 2) you have called library functions such as pow and sqrt. In this chapter you will learn how to create your own functions that can be used like library functions. Functions are commonly used to break a problem down into small manageable pieces, or modules. Instead of writing one long function that contains all the statements necessary to solve a problem, several smaller functions can be written, with each one solving a specific part of the problem.

Modular Programming

Figure 6-1

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

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

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

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

main function

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

function 2

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

function 3

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

function 4

Modular Programming

Another reason to write functions is that they simplify programs. If a specific task is performed in several places in a program, a function can be written once to perform that task, and then be executed anytime it is needed. This benefit of using functions is known as code reuse because you are writing the code to perform a task once and then reusing it each time you need to perform the task.

Defining and Calling Functions

A function call is a statement that causes a function to execute. A function definition contains the statements that make up the function.

When creating a function, you must write its definition. All function definitions have the following parts four parts:

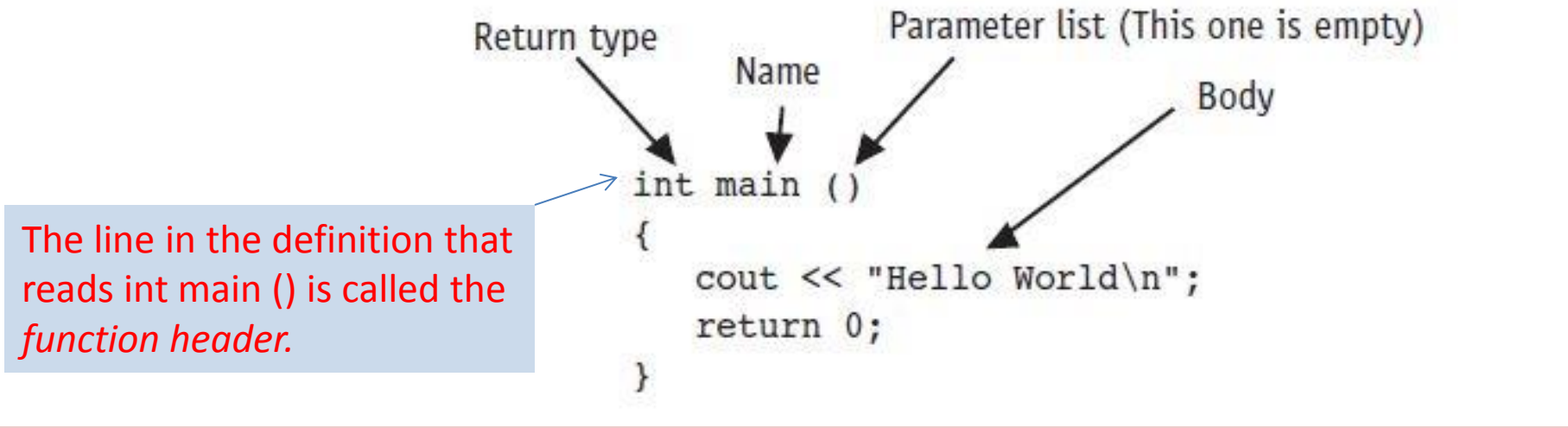
1. **Name** - Every function must have a name. In general, the same rules that apply to variable names also apply to function names.
2. **Parameter list** - The program module that calls a function can send data to it. The parameter list is the list of variables that hold the values being passed to the function. If no values are being passed to the function, its parameter list is empty.

Defining and Calling Functions

3. **Body** - The body of a function is the set of statements that carry out the task the function is performing. These statements are enclosed in a set of braces.
4. **Return type** - A function can send a value back to the program module that called it.

The return type is the data type of the value being sent back.

Figure 6-2



Void Functions

You already know that a function can return a value. The main function in all of the programs you have seen in this book is declared to return an int value to the operating system.

The return 0; statement causes the value 0 to be returned when the main function finishes executing.

It isn't necessary for all functions to return a value, however. Some functions simply perform one or more statements and then return. In C++ these are called void functions. The displayMessage function shown here is an example:

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