

1

CHAPTER

An Introduction to C++ and the C++ Programming Environment

After studying this chapter, you will be able to:

- ④ Discuss the C++ programming language and its history
- ④ Explain introductory concepts and terminology used in object-oriented programming
- ④ Recognize the structure of a C++ program
- ④ Complete the C++ development cycle, which includes creating a source code file, compiling the source code, and executing a C++ program

You should do the exercises and labs in this chapter only after you have finished Chapter 1 of your book, *Programming Logic and Design, Eighth Edition*, by Joyce Farrell. This chapter introduces the C++ programming language and its history. It explains some introductory object-oriented concepts, and describes the process of compiling and executing a C++ program. You begin writing C++ programs in Chapter 2 of this book.

2

The C++ Programming Language

The C programming language was written in the early 1970s by Dennis Ritchie at AT&T Bell Labs. C is an important programming language because it is both a high-level and a low-level programming language. It is a **high-level language**, which means that it is more English-like and easier for programmers to use than a low-level language. At the same time, it possesses **low-level language** capabilities that allow programmers to directly manipulate the underlying computer hardware.



Due to their power, C and C++ have been used in the programming of special effects for action movies and video games.

The C++ programming language was developed by Bjarne Stroustrup at AT&T Bell Labs in 1979 and inherited the widespread popularity of C. Because many programmers liked using the powerful C programming language, it was an easy step to move on to the new C++ language.

What makes C++ especially useful for today's programmers is that it is an object-oriented programming language. The term **object-oriented** encompasses a number of concepts explained later in this chapter and throughout this book. For now, all you need to know is that an object-oriented programming language is modular in nature, allowing the programmer to build a program from reusable parts of programs called classes, objects, and methods.

An Introduction to Object-Oriented Terminology

You must understand a few object-oriented concepts to be successful at reading and working with C++ programs in this book. Note, however, that you will not learn enough to make you a C++ programmer. You will have to take additional courses in C++ to become a C++ programmer. This book teaches you only the basics.

To fully understand the term *object-oriented*, you need to know a little about procedural programming. Procedural programming is a style of programming that is older than object-oriented programming. **Procedural programs** consist of statements that the computer runs or **executes**. Many of the statements make calls (a request to run or execute) to groups of other statements that are known as procedures, modules, methods, or subroutines. These programs are known as "procedural" because they perform a sequence of procedures. Procedural programming focuses on writing code that takes some data (for example, some sales figures), performs a specific task using the data (for example, adding up the sales figures),

and then produces output (for example, a sales report). When people who use procedural programs (the **users**) decide they want their programs to do something slightly different, a programmer must revise the program code, taking great care not to introduce errors into the logic of the program.

Today, we need computer programs that are more flexible and easy to revise. Object-oriented programming languages, including C++, were introduced to meet this need. In object-oriented programming, the programmer can focus on the data that he or she wants to manipulate, rather than the individual lines of code required to manipulate that data (although those individual lines still must be written eventually). An **object-oriented program** is made up of a collection of interacting objects. An **object** represents something in the real world, such as a car, an employee, a video game character, or an item in an inventory. An object includes (or **encapsulates**) both the data related to the object and the tasks you can perform on that data. The term **behavior** is sometimes used to refer to the tasks you can perform on an object's data. For example, the data for an inventory object might include a list of inventory items, the number of each item in stock, the number of days each item has been in stock, and so on. The behaviors of the inventory object might include calculations that add up the total number of items in stock and calculations that determine the average amount of time each item remains in inventory.



The preceding assumes you are using classes that someone else previously developed. That programmer must write code that manipulates the object's data.

In object-oriented programming, the data items within an object are known collectively as the object's **attributes**. You can think of an attribute as one of the characteristics of an object, such as its shape, its color, or its name. The tasks the object performs on that data are known as the object's **methods**. (You can also think of a method as an object's behavior.) Because methods are built into objects, when you create a C++ program, you do not always have to write line after line of code telling the program exactly how to manipulate the object's data. Instead, you can write a shorter line of code, known as a **call**, that passes a message to the method indicating that you need it to do something.

For example, you can display dialog boxes, scroll bars, and buttons for a user of your program to type in or click on simply by sending a message to an existing object. At other times, you will be responsible for creating your own classes and writing the code for the methods that are part of that class. Whether you use existing, prewritten classes or create your own classes, one of your main jobs as a C++ programmer is to communicate with the various objects in a program (and the methods of those objects) by passing messages. Individual objects in a program can also pass messages to other objects.

When C++ programmers write an object-oriented program, they begin by creating a class. A **class** can be thought of as a template for a group of similar objects. In a class, the programmer specifies the data (attributes) and behaviors (methods) for all objects that belong to that class. An object is sometimes referred to as an **instance** of a class, and the process of creating an object is referred to as **instantiation**.

To understand the terms *class*, *instance*, and *instantiation*, it is helpful to think of them in terms of a real-world example—baking a chocolate cake. The recipe is similar to a class and an actual cake is an object. If you wanted to, you could create many chocolate cakes that are all based on the same recipe. For example, your mother’s birthday cake, your sister’s anniversary cake, and the cake for your neighborhood bake sale all might be based on a single recipe that contains the same data (ingredients) and methods (instructions). In object-oriented programming, you can create as many objects as you need in your program from the same class.

The Structure of a C++ Program

When a programmer learns a new programming language, the first program he or she traditionally writes is a Hello World program—a program that displays the message “Hello World” on the screen. Creating this simple program illustrates that the language is capable of instructing the computer to communicate with the outside world. The C++ version of the Hello World program is shown in Figure 1-1.

```
#include <iostream>
using namespace std;
int main()
{
    cout << "Hello World" << endl;
    return 0;
}
```

Figure 1-1 Hello World program

At this point, you are not expected to understand all the code in Figure 1-1. Just notice that the code begins with the preprocessor directive, `#include <iostream>`. The C++ **preprocessor** is a program that processes your C++ program before the compiler processes it. The `#include` directive tells the compiler to include another file when the program is compiled. This makes it easy for you to use code previously written by you or others in your programs without having to re-create it. You will learn more about the Visual C++ compiler later in this chapter. Following the `#include`, you see `<iostream>`, which is the name of a header file you want to include in this program. The **iostream header file** gives your program access to what it needs to perform input and output in a C++ program. The name of the header file is placed within angle brackets (`< >`). The angle brackets tell the compiler to look for this file in a directory that is specified by the compiler you are using. You will learn more about preprocessor directives throughout this book.



Namespaces are a relatively new addition to C++ and are used primarily in large programs. In this book, the only namespace you will use is the `std` namespace.

The next line (`using namespace std;`) instructs the compiler to use the `std` namespace. You can think of a **namespace** as a container that holds various program elements. The **std** name space contains everything C++ programs need to use the Standard C++ library. The **Standard C++ library** adds functionality to the C++ programming language. For example, this program needs to use the `std` namespace to have access to `cout` and `endl`, which you see on the fifth line in Figure 1-1. Notice that this line ends with a semicolon (`;`). In fact, all C++ statements end with a semicolon. The reason the previous line, `#include <iostream>`, does not end with a semicolon is that it is a preprocessor directive, not a C++ statement.



You can tell `main()` is a function because of the parentheses; all C++ function names are followed by parentheses.

On the third line you see the start of a function named `main()`. A **function** is a group of C++ statements that perform a specified task. This is a special function in a C++ program; the `main()` function is the first function that executes when any program runs. The programs in the first eight chapters of this book will include only a `main()` function. In later chapters you will be able to include additional functions.

The first part of any function is its **header**. In Figure 1-1, the header for the `main()` function begins with the `int` keyword, followed by the function name, which is `main()`. A **keyword** is a special word that is reserved by C++ to have a special meaning. To understand the keyword `int` you need to know that functions often send values back to a calling function (for example, the result of a calculation), which can then be used elsewhere in the program. Another way to say this is that functions sometimes return a value. In Figure 1-1, the keyword `int` indicates that the `main()` function returns an integer. You will learn more about functions returning values in Chapter 9 of this book.

The opening curly brace (`{`) on the fourth line of Figure 1-1 marks the beginning of the body of the `main()` function and the closing curly brace (`}`) on the last line of Figure 1-1 marks the end of the `main()` function. All the code within this pair of curly braces executes when the `main()` function executes. In Figure 1-1, there are two lines of code between the curly braces. The first is:

```
cout << "Hello World" << endl;
```



Do not confuse the last character in `endl`; it is a lowercase letter `l`, not the numeral `1`.

This is the line that causes the words `Hello World` to appear on the user's screen. This line consists of multiple parts. The first part, `cout`, is the name of an object that represents the user's screen. Next, you see `<<`, which is called the **insertion** or **output** operator. You use `cout` and `<<` to output what follows, which in this example is the string constant `"Hello World"`. (The quotation marks will not appear on the screen, but they are necessary to make the program work.) After `"Hello World"` you see another `<<` which causes `endl` to be displayed (after `Hello World`) on the user's screen. For now, think of `endl` as a

newline that causes the cursor to be positioned on the line after `Hello World`. You will learn more about `endl` in Chapter 9 of this book. Note that the semicolon that ends the `cout << "Hello World" << endl;` statement is required because it tells the compiler that this is the end of the statement.

6

The next line of code is `return 0;`. This statement instructs the compiler to return the value 0 from the `main()` function. Remember, when you saw the header for the `main()` function on the third line of this program, you used the keyword `int` to specify that the `main()` function returns an integer; 0 is the returned integer. Conventionally, when a program returns a 0, it means “everything went well and the program is finished.”

Next, you learn about the C++ development cycle so later in this chapter you can compile the Hello World program and execute it.

The C++ Development Cycle

When you finish designing a program and writing the C++ code that implements your design, you must compile and execute your program. This three-step process of writing code, compiling code, and executing code is called the C++ development cycle. It is illustrated in Figure 1-2.

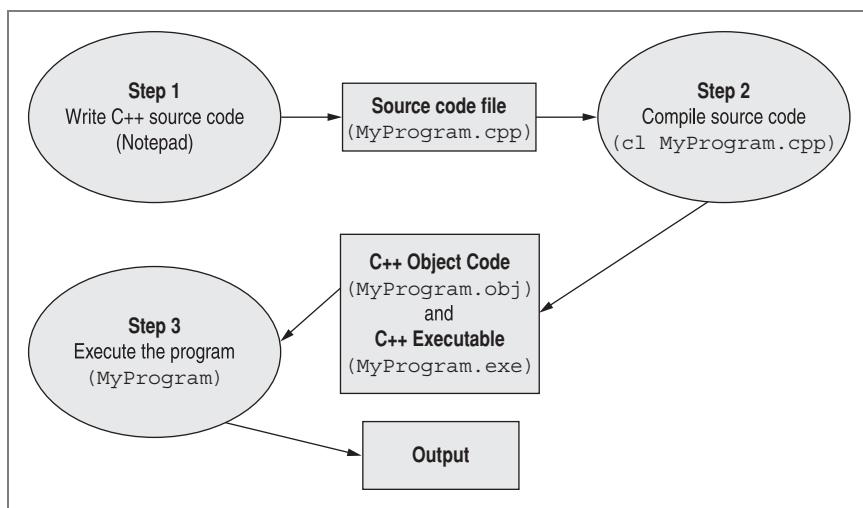


Figure 1-2 C++ development cycle

Writing C++ Source Code

As you learned in the previous section, you write a C++ program by creating a function named `main()` that contains C++ statements. But what do you use to write the program, and where do you save it?

To write a C++ program, you can use any text editor, but the steps in this book assume you are using Windows Notepad. To start Notepad in Windows 7, click the **Start** button, point to **All Programs**, click **Accessories**, and then click **Notepad**. To start Notepad in Windows 8, right-click on a blank area of the **Start** screen, click **All apps** scroll to the end, and then click **Notepad**. In the next version of Windows 8, you click the down arrow on the **Start** screen, scroll to the end, and then click **Notepad**. Once Notepad starts, you simply type in your C++ source code. **Source code** is the term used for the statements that make up a C++ program. For example, the code shown earlier in Figure 1-1 is source code.

When you save the file that contains the source code, it is important to give the file a meaningful name, and then add the extension `.cpp`. For the Hello World program, an appropriate name for the source code file is `HelloWorld.cpp`. Of course, it is also important to remember the location of the folder in which you save your source code file.

Compiling a C++ Program

The Visual C++ compiler is named `c1`. It is a program that is responsible for a two-step process that takes your source code and transforms it into object code and then links the object code to create executable code.



The Visual C++ compiler, `c1`, is actually a compiler and a linker.

Object code is code in computer-readable form that is linked with libraries to create an executable file. **Executable code** is the *1s* and *0s* a computer needs to execute a program. The C++ compiler automatically saves the object code in a file. This file has the same name as the source code file, but it has an `.obj` extension rather than a `.cpp` extension.

The following steps show how to compile a source code file. These steps assume you have already created and saved the `HelloWorld.cpp` source code file.

1. Open a Command Prompt window. To do this in Windows 7, click the **Start** button, point to **All Programs**, click **Accessories**, click **Visual Studio 2012 Express**, and then click **Developer Command Prompt for VS2012**. In Windows 8, right-click on a blank area of the **Start** screen, click **All apps**, scroll until you see the Microsoft Visual Studio 2012 area, and then click **Developer Command Prompt for VS2012**. In the next version of Windows 8, you click the down arrow on the **Start** screen, click **All apps**, scroll until you see the Microsoft Visual Studio 2012 area, and then click **Developer Command Prompt for VS2012**. The cursor blinks to the right of the current file path.
2. To compile your source code file, you first have to change to the file path containing your source code file. To do this, type `cd drive letter:\path` where **drive letter** is the drive containing your file, and **path** is the path to the folder containing your file. For example, to gain access to a file stored in a folder named “Testing”, which is in turn stored in a folder named “My Program”, which is stored on the C drive, you would type `cd C:\My Program\Testing`. After you type the command, press **Enter**. The cursor now blinks next to the file path for the folder containing your source code file.

3. Type the following command, which uses the C++ compiler, `c1`, to compile the program:

c1 HelloWorld.cpp

If there are no syntax errors in your source code, a file named `HelloWorld.obj` and a file named `HelloWorld.exe` are created. You do not see anything special happen. However, the files you just created contain the object code (`HelloWorld.obj`) and executable code (`HelloWorld.exe`) for the Hello World program. If there are syntax errors, you will see error messages on the screen; in that case, you need to go back to Notepad to fix the errors, save the source code file again, and recompile until there are no syntax errors remaining. **Syntax errors** are messages from the compiler that tell you what your errors are and where they are located in your source code file. For example, omitting a semicolon at the end of the statement `cout << "Hello World"` `<< endl` results in a syntax error.



At this point in your programming career, do not expect to understand the contents of files with an `.obj` or `.exe` extension if you open one using a text editor such as Notepad.

4. After the program is compiled, you can use the `dir` command to display a directory listing to see the files named `HelloWorld.obj` and `HelloWorld.exe`. To execute the `dir` command, you type `dir` at the command prompt. For example, if your source code file is located at `C:\My Program\Testing`, the command prompt and `dir` command should be `C:\My Program\Testing> dir`. The `HelloWorld.obj` and `HelloWorld.exe` files should be in the same directory as the source code file `HelloWorld.cpp`.

Executing a C++ Program

To execute the Hello World program, do the following:

1. Open a Command Prompt window. To do this, refer to Step 1 under Compiling a C++ program. Change to the file path containing your executable code file, if necessary, and then enter the following command:

HelloWorld



You must be in the same directory that contains the `.exe` file when you execute the program.

2. When the program executes, the words `Hello World` appear in the Command Prompt window.

Figure 1-3 illustrates the steps involved in compiling `HelloWorld.cpp` using the `cl` compiler, and executing the `dir` command to verify that the files `HelloWorld.obj` and `HelloWorld.exe` were created, as well as the output generated by executing the Hello World program.

The screenshot shows a Windows Command Prompt window titled "Command Prompt". The window contains the following text:

```
c:\C++>cl HelloWorld.cpp
Microsoft (R) 32-bit C/C++ Optimizing Compiler Version 16.00.30319.01 for 80x86
Copyright (C) Microsoft Corporation. All rights reserved.

HelloWorld.cpp
C:\Program Files\Microsoft Visual Studio 10.0\VC\INCLUDE\xlocale(323) : warning
C4530: C++ exception handler used, but unwind semantics are not enabled. Specify
/EHsc
Microsoft (R) Incremental Linker Version 10.00.30319.01
Copyright (C) Microsoft Corporation. All rights reserved.

/out:HelloWorld.exe
HelloWorld.obj

c:\C++>dir
Volume in drive C is OS
Volume Serial Number is BC42-7996

Directory of c:\C++
09/06/2011  11:32 AM    <DIR>
09/06/2011  11:32 AM           104 HelloWorld.cpp
09/06/2011  11:31 AM           99,328 HelloWorld.exe
09/06/2011  11:33 AM           45,711 HelloWorld.obj
               3 File(s)   145,143 bytes
               2 Dir(s)  31,612,461,056 bytes free

c:\C++>HelloWorld
Hello World
c:\C++>
```

Figure 1-3 Compiling and executing the Hello World program

Exercise 1-1: Understanding the C++ Compiler

In this exercise, assume you have written a C++ program and stored your source code in a file named `MyCPlusPlusProgram.cpp`. Answer the following questions:

1. What command would you use to compile the source code?

2. What command would you use to execute the program?

Lab 1-1: Compiling and Executing a C++ Program

In this lab, you compile and execute a prewritten C++ program, and then answer some questions about the program.

1. Open the source code file named `Programming.cpp` using Notepad or the text editor of your choice.
2. Save this source code file in a directory of your choice, and then change to that directory.

- 10
-
3. Compile the source code file. There should be no syntax errors. Record the command you used to compile the source code file.

 4. Execute the program. Record the command you used to execute the program, and record the output of this program.

 5. Modify the program so it displays "I'm learning how to program in C++". Save the file as `C++Programming.cpp`. Compile and execute.
 6. Modify the `C++Programming` program so it prints two lines of output. Add a second output statement that displays "That's Awesome!" Save the modified file as `Awesome.cpp`. Compile and execute the program.