Overview

You learned in the previous chapter that C++ source code has to be entered into a text editor, translated by a compiler, and made into an executable program by a linker.

Your task in this chapter will be to create an actual C++ program on your system. You will first examine the structure of a C++ program. Then you will enter a simple program into the text editor, compile, link, and run the executable file that is created.

CHAPTER 3, SECTION 1

C++ Program Structure

C++ programs have the basic structure illustrated in Figure 3-1. They are:

1. ***Comments*** - Comments are remarks that are ignored by the compiler.
2. ***Compiler Directives*** - Compiler directives are commands for the compiler, which are needed to effectively compile and run your program.
3. ***Main Function*** - The main function is where every C++ program begins.
4. ***Braces*** - Braces are special characters used to mark the beginning and ending of blocks of code.
5. ***Statement*** - A statement is a line of C++ code. Statements end with a semicolon.

Let's examine each part of a C++ program in more detail.

Note

As you will learn later in this course, a C++ program can have other functions in addition to the main function.

COMMENTS

When writing a program, you may think that you will always remember what you did and why. Most programmers, however, eventually forget. But more importantly, others may need to make changes in a program you wrote. They probably will be unaware of what you did when you wrote the program. That is why comments are important.

Use comments to:

- explain the purpose of a program
- keep notes regarding changes to the source code
- store the names of programmers for future reference
- explain the parts of your program.

Figure 3-2 is an example of a program that is well-documented with comments. The comments at the top of the program assign the program a name,

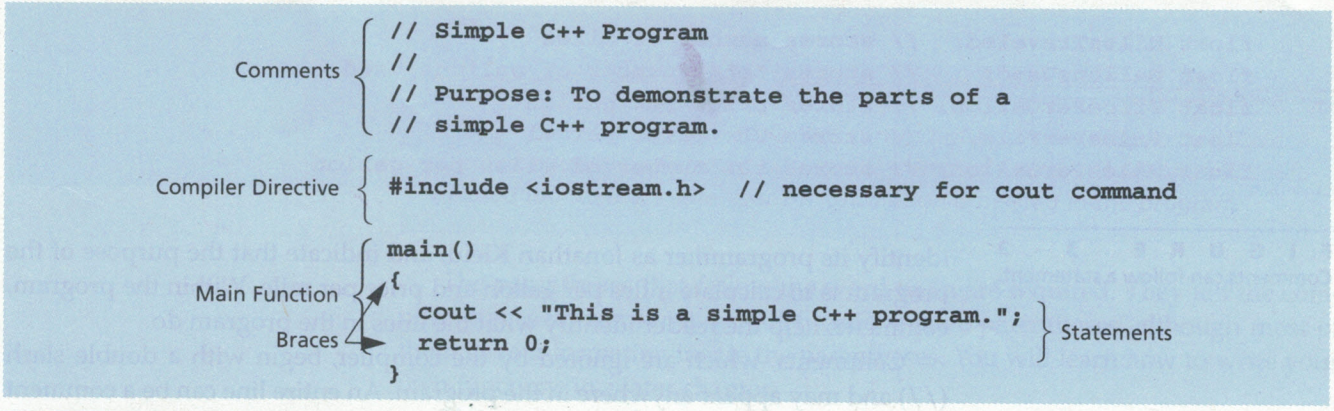

FIGURE 3-1
A C++ program has several parts.

```
// Travel Efficiency
// Programmer: Jonathan Kleid
//
// Purpose: Calculates miles per gallon and price per mile when
// given miles traveled, number of gallons used, and gas price.

#include <iostream.h> // necessary for cin and cout commands

main()
{
 // Variable declarations
 float MilesTraveled;      //  stores number of miles
 float GallonsUsed;        //  stores number of total gallons used
 float PricePerGallon;     //  stores price per gallon
 float PricePerMile;       //  stores the price per mile
 float MilesPerGallon;     //  stores the number of miles per gallon

 // Ask user for input values.
 cout << "How many miles did you travel? ";
 cin  >> MilesTraveled;
 cout << "How many gallons of gas did you use? ";
 cin  >> GallonsUsed;
 cout << "How much did one gallon of gas cost? $";
 cin  >> PricePerGallon;

 // Divide the number of miles by the number of gallons to get MPG.
 MilesPerGallon = MilesTraveled / GallonsUsed;

 // Divide price per gallon by miles per gallon
 // to get price per mile.
 PricePerMile = PricePerGallon / MilesPerGallon;

 // Output miles per gallon and price per mile.
 cout << "You got " << MilesPerGallon << " miles per gallon,\n";
 cout << "and each mile cost $" << PricePerMile << '\n';
 return 0;
}
```

FIGURE 3-2
Comments help make this program understandable.

```
float MilesTraveled;  // stores number of miles
float GallonsUsed;    // stores total number of gallons used
float PricePerGallon; // stores price per gallon
float PricePerMile;   // stores the price per mile
float MilesPerGallon; // stores the number of miles per gallon
```

FIGURE 3-3
Comments can follow a statement.

identify its programmer as Jonathan Kleid, and indicate that the purpose of the program is to calculate miles per gallon and price per mile. Within the program, comments help the reader identify what the lines in the program do.

Comments, which are ignored by the compiler, begin with a double slash (//) and may appear anywhere in the program. An entire line can be a comment or may appear to the right of program statements, as shown in Figure 3-3. Everything to the right of the // is ignored. Therefore, do not include any statements to the right of a comment. Be sure to use the forward-leaning slash (/) rather than the backslash (\) or the compiler tries to translate your comments and an error message results.

COMPILER DIRECTIVES

Directives are instructions to the compiler rather than part of the C++ language. The most common compiler directive is the **#include** directive, which instructs the compiler to treat the text file that is enclosed in brackets as if it were keyed into the source code. See Figure 3-4.

So why do you need other code included in your source code? The code you are including makes additional commands available to you. For example, the **#include <iostream.h>** directive that you have seen in programs in this chapter makes a set of input and output commands available. These commands make it easy to get input from the user and print to the screen.

Note

#include is pronounced simply as include. *Ignore the # sign when you pronounce the word.*

```
#include <iostream.h>
```

Name of File to Be Included

FIGURE 3-4
The **#include** compiler directive inserts other code into your program as if it were actually keyed into your program.

What Is iostream.h?

Files like **iostream.h** *are called* ***header files.*** *They may be identified by their file extension ".h". A header file serves as a link between your program code and standard C++ code that is needed to make your program run.*

You will learn more about compiler directives in later chapters.

MAIN FUNCTION

Every C++ program has a main function (see Figure 3-1). The main function is run first. Although simple programs can be written entirely within the main function, C++ programs are typically divided into multiple functions, which are accessed through the main function.

What Is a Function?

*A **function** is a block of code that carries out a specific task. For example, a function could be written to calculate the area of a circle. That function could be used (or "called") wherever the calculation is needed in the program.*

The parentheses that follow the word **main** are required. They tell the compiler that main is a function. All functions have parentheses, although most of them have information inside the parentheses. You will learn how to write your own functions in a later chapter.

The program ends with a **return 0;** statement. This statement ends the main function and returns a value of zero to the operating system. In Chapter 9, you will learn why it is important to include the **return** statement and why a zero is returned to the operating system when your program ends.

BRACES

Braces are used to mark the beginning and end of blocks of code. Every opening brace must have a closing brace. Notice in Figure 3-5 that the main function is enclosed in a set of braces. Providing comments after each closing brace helps to associate it with the appropriate opening brace. Also, aligning the indention of opening and closing braces is a good idea.

STATEMENTS

Functions contain statements that consist of instructions or commands that make the program work. Each statement in C++ ends with a semicolon.

SEMICOLONS

You must have a semicolon after every statement. The semicolon terminates the statement. In other words, it tells the compiler that the statement is complete. Notice, however, that directives such as **#include** and function declarations like **main()** are exempt from being punctuated by semicolons.

```
// COMMENT.CPP
// This program prints the common uses for comments
// to the screen.
// Program written by Greg Buxkemper

#include <iostream.h> // necessary for output statements

main()
{
  cout << "Use comments to:\n";
  cout << " - explain the purpose of a program.\n";
  cout << " - keep notes regarding changes to the program.\n";
  cout << " - store the names of programmers.\n";
  cout << " - explain the parts of a program.\n";
  return 0;
} // end of main function
```

FIGURE 3-5
Comments can help identify braces.

C++ AND BLANK SPACE

C++ allows for great flexibility in the spacing and layout of the code. Use this feature to make it easier to read the code by indenting and grouping statements as shown in the sample program in Figure 3-2.

UPPERCASE OR LOWERCASE

Remember the ASCII codes? In the computer, an *A* is represented by a different number than an *a*. The capital letters are referred to as ***uppercase,*** and small letters are called ***lowercase.***

C++ is known as ***case sensitive*** because it interprets uppercase and lowercase letters differently. For example, to a C++ compiler, the word *cow* is different than the word *Cow*. Be careful to use the same combination of lettering (either uppercase or lowercase) when you enter source code. Whatever capitalization was used when the command was originally named is what must be used. Most of what you will see in C++ will be in lowercase letters. If you key a command in uppercase that is supposed to be lowercase, you will get an error.

SECTION 3.1 QUESTIONS

1. List four uses for comments.
2. What compiler directive inserts source code from another file into your program?
3. What is a function?
4. What purpose do braces serve?
5. What does the term "case sensitive" mean?

CHAPTER 3, SECTION 2

From Source Code to a Finished Product

The exact process required to enter source code, compile, link, and run will vary depending on the compiler you are using. Your instructor will help you perform these tasks with your compiler.

ENTERING SOURCE CODE

The first step is to enter your C++ source code into a text file. Most C++ compilers have an integrated programming environment that contains a text editor you can use. An integrated programming environment allows you to enter your source code, compile, link, and run while your text editor is on the screen.

EXERCISE 3-1 ENTERING SOURCE CODE

1. Start your text editor with a new, blank file.
2. Enter the C++ source code *exactly* as it is shown below.

```
// MYPROG
// My first C++ Program

#include <iostream.h>

main()
{
  cout << "My first C++ program.\n";
  return 0;
}
```

Special Note: The "**\n**" causes the compiler to move the cursor to the beginning of the next line after printing the output to the screen.

3. Save the file as *MYPROG.CPP* and leave the program on your screen for the next exercise.

COMPILING, LINKING, AND RUNNING THE PROGRAM

Most compilers allow you to compile, link, and run with a single command from the integrated environment.

EXERCISE 3-2 COMPILING, LINKING, AND RUNNING THE PROGRAM

1. Compile, link, and run the program you entered in Exercise 3-1. If your compiler allows all of these operations to be performed with a single command, use that command. If your program fails to compile or link, check to see if you entered the code exactly as shown in Exercise 3-1 and try again.
2. If your program runs successfully, you should see the text *My first C++ program* on the screen. Otherwise, ask your instructor for help. *Note:* Some compilers may require that you give the integrated programming environment a command to show the program's output. Your instructor will know what command is necessary.
3. Leave the source file open for the next exercise.

MAKING CHANGES AND COMPILING AGAIN

You can add, change, or delete lines from a program's source code and compile it again. The next time the program is run, the changes will be in effect.

EXERCISE 3-3 MAKING CHANGES AND COMPILING AGAIN

1. Add the statement below to the main function, substituting your name in place of Angela Askins.

```
cout << "By Angela Askins\n";
```

Your program should now appear like the one below, except your name should be on the new line.

```
// MYPROG
// My first C++ program.

#include <iostream.h>

main()
{
  cout << "My first C++ program.\n";
  cout << "By Angela Askins\n";
  return 0;
}
```

2. Compile, link, and run the program again to see the change.
3. Save the source code file and leave it open for the next exercise.

CREATING A STANDALONE PROGRAM

Compiling, linking, and running a program, depending upon your specific compiler, may have already created a standalone program on disk. Your instructor will know for sure.

EXERCISE 3-4 CREATING A STANDALONE PROGRAM

If a standalone program was generated as a result of completing Exercise 3-3, quit the integrated programming environment and run the standalone program from the operating system. Otherwise, complete steps 1-3.

1. Select the option that allows you to compile and link to disk so that a standalone executable file is created.
2. Quit the integrated programming environment.
3. Run the executable program from the operating system.

LOADING AND COMPILING AN EXISTING SOURCE FILE

Often you will load an existing source code file and compile it. Most integrated programming environments have an Open command that can be used to open source files.

EXERCISE 3-5 LOADING AND COMPILING AN EXISTING SOURCE FILE

1. Start your integrated programming environment.
2. Open the source file *TRAVEL.CPP*. Your instructor will either provide you with a work disk or give you instructions for accessing the file from the hard disk or network.

3. Compile, link, and run the program.
4. When the program prompts you for data, enter values that seem realistic to you and see what output the program gives.
5. Run the program several times with different values.
6. Close the source file and quit.

Responsibilities of the Programmer

As you write more advanced computer programs, you should keep certain responsibilities in mind.

1. ***Privacy.*** *Programmers often have access to databases and other information about individuals. Programmers have a responsibility to protect the privacy of this information.*
2. ***Property Rights.*** *Ideas are not protected by copyright law. Actual program code, however, is. Using software that you do not have legal license to use, or using other programmers' source code without proper permissions is illegal and irresponsible.*
3. ***Impact of Software.*** *Software can do physical damage (such as viruses) or have social ramifications. Computers should not be used by programmers to cause harm to users, and the impact of a program on society should be considered before writing a program.*
4. ***Reliability.*** *Individuals, schools, businesses, and the government rely on computers more every year. Programmers have a responsibility to produce software that is as reliable as possible and to report and/or repair problems that may affect the reliability of the system.*

CONGRATULATIONS

Congratulations. You now know the basics of creating and running C++ programs. From here you will simply add to your knowledge to enable you to write more useful programs. If you feel you need more experience with compiling and running C++ programs, repeat the exercises in this chapter or ask your instructor for additional help. Future exercises require that you know how to compile, link, and run.

On the Net

For links to the web pages of major C++ compiler makers, see http://www.ProgramCPP.com. See topic 3.2.1.

SECTION 3.2 QUESTIONS

1. What company developed the compiler you are using?
2. What is the name of the compiler you are using and its version number?
3. What command or commands are used to run a program with your compiler?
4. What command opens a source code file from a disk?
5. What command saves a source code file?

KEY TERMS

braces	header file
case sensitive	lowercase
comments	main function
compiler directive	statements
function	uppercase

SUMMARY

- A C++ program has several parts.
- Comments are remarks that are ignored by the compiler. They allow you to include notes and other information in the program's source code.
- Directives are commands for the compiler, rather than part of the C++ language.
- All C++ programs have a main function. The main function is where the program begins running.
- Braces mark the beginning and end of blocks of code.
- Statements are the lines of code the computer executes. Each statement ends with a semicolon.
- C++ allows you to indent and insert space in any way that you want. You should take advantage of this flexibility to format source code in a way that makes programs more readable.
- C++ is case sensitive, which means that using the wrong capitalization will result in errors.

PROJECTS

PROJECT 3-1

Enter the program shown below but substitute your name and the appropriate information for your compiler. Compile, link, and run. Save the source code as *COMPINFO.CPP*.

```
// COMPINFO
// By Jeremy Wilson

#include <iostream.h>

main()
{
  cout << "This program was compiled using\n";
  cout << "Colossal C++ version 2.5.\n";
  return 0;
}
```

PROJECT 3-2

Open the source file *BRACES.CPP*, compile it, link, and run. After you have run the program, close the source file and quit.

PROJECT 3-3

Write a program that prints the message of your choice to the screen. Make the message at least four lines long. Save the source code file as *MY_MSG.CPP*.