
CHAPTER 2

A FIRST LOOK AT C++ SOURCE PROGRAMS

We have already seen one C++ source program in the previous chapter in Figure 1-3, but we have not yet tried to understand it. In this chapter, we shall study a couple of computer programs, including the one in Figure 1-3. You will learn some basic things, such as

- how a C++ program displays text on the screen,
- how a C++ program reads in numbers from the keyboard,
- how simple calculations can be made in C++ programs, and
- how C++ programs can be compiled and executed.

After this chapter, you will know something about C++ programming, but it is likely that you will have many questions in your mind. You may think

- how can a variable store information?
- what is actually happening during compilation?
- how does a computer execute programs?

It is perfectly natural to have these questions in your mind. In chapters 3 and 4 we shall study some basics of computing, and these questions will be answered then.

These are sample pages from Kari Laitinen's book
"A Natural Introduction to Computer Programming with C++".
For more information, please visit
<http://www.naturalprogramming.com/cppbook.html>

2.1 A program that can print text to the screen

The simplest things computer programs can do is to read in some data (e.g. numbers or text) from the keyboard, do something to that data, and then print something to the screen. Perhaps the simplest computer program is one that just prints a single sentence of text to the screen. Such a computer program is **first.cpp** which is presented in a program description on the page opposite. Later on in this chapter, you will be instructed how to get the file **first.cpp** on your own computer. When the program is compiled and executed on a computer, the following line will appear on the screen of the computer:

```
I am a simple computer program.
```

By studying program **first.cpp** you can discover that, in some ways, computer programs resemble texts written in natural languages like English. For example, the program **first.cpp** contains such natural words as **include** and **main**. On the other hand, program texts are in many ways much more complex than natural-language texts. For example, there are special characters like # < > { } () ; and a special name **cout** in use in program **first.cpp**. All these characters have a special meaning in the program. The special characters are interpreted by the compiler of the C++ programming language when the program is transformed into an executable form.

The act of printing text to the screen of a computer is called output. We say that a program can output text to the screen. In program **first.cpp** text output is performed by the output statement

```
cout << "I am a simple computer program." ;
```

The output mechanism of the C++ programming language is such that the name **cout** represents the screen and << is an operator that is used to move, or shift, text to the screen. The text to be printed must be written inside double quote characters. << is an output operator, and **cout** is an output stream representing the screen of the computer. We can imagine that with operator << we add text to the output stream.

cout, which is usually pronounced "see-out", is defined in the file named **iostream.h**. This file is included in program **first.cpp** with the command

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

We will have this include command at the beginning of every example program in this book. The file **iostream.h** comes with the C++ compiler, and it contains important definitions related to input and output. It is important that you remember to write the above command at the beginning of your programs, but you do not have to examine what is inside the file **iostream.h**.

All C++ programs must be written according to a certain structure, that is, the syntax of the programming language. All C++ programs up until Chapter 8 of this book follow the structure

```
#include <iostream.h>

int main()
{
    ...
}
```

The meaning of the three dots is "some program lines written with C++". In the place of the three dots there is something different in each program, but in our first programs there is always **int main()** followed by an opening brace { which indicates where the actual program statements begin, and then there is a closing brace } which terminates the program.

In this book, example computer programs are described so that the actual program text is explained within "balloons" such as these. These balloons are not part of the computer program. Instead, they are a means to teach you. The arrow originating from this balloon points to the first line of program **first.cpp**. This program has only six lines of which one is an empty line. Note that the actual program is always written with the font called **courier**.

This is the line in this source program which specifies what will be printed to the screen when the program is executed. The text to be printed to the screen must be written "inside double quote characters", and there must be a semicolon ; at the end.

```
> #include <iostream.h>

int main()
{
    cout << "I am a simple computer program." ;
}
>
```

Pairs of braces { } are used to group the program statements in C++. This closing brace terminates the entire program. In this case there is only one statement inside the braces

This text is not part of the computer program. Instead, it contributes to how computer programs are described in this book. Note that these caption texts always show what the physical file name of the program is. This program can be found by name **first.cpp**.

first.cpp - 1. A C++ program that prints a single line of text to the screen.

```
D:\book1cpp>first
I am a simple computer program.
D:\book1cpp>
```

first.cpp - X. The execution of the compiled program **first.exe**

In this book the execution of an example program is shown in a box which has its own caption text. The text inside the box is what appears on the screen of the computer when the program is executed. A letter X in the caption text means that this describes the execution of the program. The original file name of the program is always shown in the caption of a program description.

2.2 A program that can read from the keyboard and calculate

In the same way as writing text to the screen is called output, the act of reading text from the keyboard of a computer is called input. We say that a program can input text or numerical data from the keyboard. The screen is an output device, whereas the keyboard is an input device for a computer program. When speaking about input and output in C++ programming, it is customary to speak about input streams and output streams. The name `cout` represents the standard output stream which is the screen, and the name `cin` (pronounced see-in) represents the standard input stream which is the keyboard.

`sum.cpp` is an example program which can read two integers from the keyboard and calculate the sum of the integers read. Program `sum.cpp` is presented as a program description on the following page. The execution of the program is described below the source program. Note that because the caption of the source program description has a + marked, it means there are some parts of the program explained in more detail in other descriptions.

Integers are one data type in the C++ programming language. Integer variables can be declared with the reserved keyword `int`. An integer variable can store an integer value. Integers are whole numbers (... -2, -1, 0, 1, 2, 3, ...) which do not have any decimal points or decimal fractions. Program `sum.cpp` first declares the necessary integer variables, then it asks the user to give values for the integer variables, then it calculates their sum, and finally the sum of the integers is printed to the screen.

An important element of a computer program is a statement. In C++, statements are terminated so that a semicolon `;` is written at the end of every statement. When the compiler of the programming language finds a semicolon in the program text, it knows that that is the end of a statement. Our first C++ programs consist of a single entity which is called the function `main()`. Inside every function `main()` there are one or more statements. In program `first.cpp` there is only one statement inside the function `main()`, but in program `sum.cpp` there are ten statements and each of them has a semicolon `;` at the end.

When a computer executes a program, the statements are executed one at a time, in a sequential order. As a general rule, the statements which are written earlier in a program are executed before those statements which are written later in the program. For example, in program `sum.cpp`, the statement

```
cout << sum_of_two_integers ;
```

is the last statement to be executed because it is the last statement in the program.

There are basically two kinds of statements in C++ programs:

- The declaration statements which introduce names and define data. They do not cause any action to happen in a program. An example of this kind of a statement is

```
int first_integer ;
```

which declares a variable named `first_integer`. In program `sum.cpp`, this variable is used to store the first integer that is read from the keyboard.

- Action statements which make a program perform some actions. These statements can be thought of as commands to the computer, and they usually come after data declaration statements in a program. In program `sum.cpp`, an example of this kind of statement is

```
sum_of_two_integers = first_integer + second_integer ;
```

This statement can be described in a command form such as: "Calculate the sum of the contents of variables `first_integer` and `second_integer` and store the sum into variable `sum_of_two_integers`." The variables must be declared in data declaration statements before these kinds of action statements can be allowed in a program.

The most important function of a C++ program is called `main()`. The type of the function in this case is `int`. In most of our C++ programs, we will have a function named `main()` whose type is `int`.

Here, three variables of type `int` are declared. These variables can store integers (whole numbers). `int` is simply an abbreviation of the word integer. Note that all program statements in C++ are terminated with a semicolon ;

```
#include <iostream.h>
> int main()
{
    int first_integer ;
    int second_integer ;
    int sum_of_two_integers ;

    cout << "\n Please, type in an integer:      " ;
    cin >> first_integer ;

    cout << "\n Please, type in another integer: " ;
    cin >> second_integer ;

    sum_of_two_integers = first_integer + second_integer ;

    cout << "\n The sum of the given integers is " ;
    cout << sum_of_two_integers ;
}

```

These two statements read numerical values from the keyboard. `>>` is the input operator which reads the input stream `cin`. The numbers which the user of the program types in from the keyboard become the values of the variables `first_integer` and `second_integer`.

This is an arithmetic statement which calculates the sum of the two integers. After this statement has been executed, the sum is stored in the variable named `sum_of_two_integers`.

This plus sign means that this program will be further explained on following pages.

sum.cpp - 1.+ A program to calculate the sum of two integers.

```
D:\book1cpp>sum
Please, type in an integer:      17
Please, type in another integer: 95
The sum of the given integers is 112

```

sum.cpp - X. The program is executed here by typing in the integers 17 and 95.

`int` is a reserved keyword in C++. That means that it cannot be used as a name in a C++ program. When the C++ compiler reads the word `int` in a program, it knows that an integer variable is being declared.

`first_integer`, `second_integer`, and `sum_of_two_integers` are names of variables in this program. The names of variables can be chosen by the person who writes a program. In this book we write the names so that they consist of natural words which are joined with underscore characters `_` (No space characters are allowed in a name.)

```
#include <iostream.h>

int main()
{
    int first_integer ;
    int second_integer ;
    int sum_of_two_integers ;

    cout << "\n Please, type in an integer: " ;
    cin >> first_integer ;

    cout << "\n Please, type in another integer: " ;
    cin >> second_integer ;
```

The term `cin` represents the keyboard, the standard input stream, in a C++ program. With the input operator `>>` it is possible to "move data" from the keyboard to a variable. After this statement has been executed, the number that was typed in from the keyboard will be the value of the variable `second_integer`.

These statements print text lines to the screen. The marking `'\n'` at the beginning of the text causes the text to be printed on a new line. Therefore `'\n'` is called the newline character.

sum.cpp - 1 - 1: The first part of the complete program sum.cpp.

This is called an assignment statement. The values of the variables `first_integer` and `second_integer` are summed first, and then the calculated sum is assigned as a value to the variable `sum_of_two_integers`.

Mathematical notations are used here. The plus sign `+` is used as an addition operator. The equal sign `=` is an assignment operator.

```
sum_of_two_integers = first_integer + second_integer ;

cout << "\n The sum of the given integers is " ;
cout << sum_of_two_integers ;
}
```

sum.cpp - 1 - 2. The last part of the complete program sum.cpp.

Usually, there is no single way of writing a computer program. Because programming languages provide a great variety of means to perform various computing activities, computing problems, like the problem of calculating the sum of two integers, can be solved with different kinds of programs. Program **sum_improved.cpp**, on the following page, is an improved version of program **sum.cpp**, and it shows another possible way to write a simple computer program. Although the two programs are doing the same calculation, program **sum_improved.cpp** introduces some new features of the C++ programming language:

- At the beginning of the program, there are some English sentences which explain briefly what the program does. These sentences are called the comments of the program. Comments are actually not part of the program text because they are discarded by the compiler. Comments are intended to be read by humans who want to study the program. The compiler recognizes comment lines through the double slash //. Whenever the compiler finds a double slash on some program line, it discards the double slash and everything that follows until the end of that line.
- Two variables have been given different names in **sum_improved.cpp** than they had in **sum.cpp**. The following modifications have been made to variable names:

```
first_integer    ->  first_integer_from_keyboard
second_integer   ->  second_integer_from_keyboard
```

The names of variables can be chosen by the author of a program. Every variable must have a name that is different from any other variable name in the same program. When a variable name is used, it must always be written in exactly the same way. The names of variables should be chosen so that they describe the purpose of the variable. C++ allows short and abbreviated variable names like `val` or `i`, or variable names containing numbers like `val2`, but we will not use them in this book. Instead, the programs in this book have meaningful variable names which consist of natural words joined together with underscore characters.

- The output and input operations can be concatenated and that possibility has been exploited in program **sum_improved.cpp**. For example,

```
cout <<  "\n Please, type in two integers separated "
      <<  "with spaces:  " ;
```

means the same as

```
cout <<  "\n Please, type in two integers separated " ;
cout <<  "with spaces:  " ;
```

Exercises with program **sum.cpp**

Before you can do these exercises, you must read the following sections where you will learn how to edit, compile, and execute C++ programs.

Exercise 2-1. Modify program **sum.cpp** so that when it is given the numbers 17 and 95 it prints the line

```
17 + 95 = 112
```

The aim is that the program will print the plus sign + and the equal sign = instead of words.

Exercise 2-2. Modify program **sum.cpp** so that it calculates the sum of three integers.

These lines are so-called comment lines which explain in plain English what this particular program does. Two adjacent slash characters `//` (a double slash) begin a comment line. When a compiler sees a double slash `//` it ignores those characters and the rest of the program line. Note that it is a good programming practice to mention the program file name and the author of the program on some comment lines.

```
// sum_improved.cpp (c) 1998 Kari Laitinen

// This is a simple calculator program that can
// calculate the sum of the two integers that are
// typed in from the keyboard.

#include <iostream.h>

int main()
{
    int first_integer_from_keyboard ;
    int second_integer_from_keyboard ;
    int sum_of_two_integers ;

    cout << "\n Please, type in two integers separated "
         << "with spaces: " ;

    cin  >> first_integer_from_keyboard
         >> second_integer_from_keyboard ;

    sum_of_two_integers = first_integer_from_keyboard +
                          second_integer_from_keyboard ;

    cout << "\n The sum of "
         << first_integer_from_keyboard
         << " and "
         << second_integer_from_keyboard
         << " is "
         << sum_of_two_integers
         << "\n" ;
}
```

Sometimes, program statements are so long that they need to be written on several lines of program text. The semicolon `;` is the symbol which marks the end of each statement

sum_improved.cpp - 1. A slightly improved version of program sum.cpp.

```
D:\book1cpp>sum_improved

Please, type in two integers separated with spaces: 115 43

The sum of 115 and 43 is 158
```

sum_improved.cpp - X. Program execution with integers 115 and 43.

The MS-DOS (console) window in Windows computers

You should read this box if you are not familiar with the use of the MS-DOS window in a computer that runs some version of the Microsoft Windows operating system. It is important that you learn to use the MS-DOS window because it is used to compile and execute the C++ programs in this book. In addition, you need the MS-DOS window to perform the installations described in this chapter. In Windows XP, for example, the MS-DOS window can be opened by selecting **Start > All Programs > Accessories > Command Prompt**. The opening of the MS-DOS window can be somewhat different in different Windows versions.

Through the MS-DOS window it is possible to carry out many operations by using the commands of the old MS-DOS operating system, and thereby control the entire computer. In many cases, you can use either an MS-DOS command or the mouse of your computer. For example, you can copy a file with the COPY command, or you can use the mouse in the copying operation.

The MS-DOS window can also be called the "command line window" because the MS-DOS commands are lines of text written according to a certain syntax. You need to know at least some of the MS-DOS commands in order to work with this book. Here are some examples of the MS-DOS commands you may need:

D:	Select drive D.
CD \	Change to the root directory in the current drive.
CD mycpp	Change to the directory named mycpp .
CD ..	Change to the parent directory of the current directory.
COPY \book1cpp\sum.cpp	Copy file sum.cpp from the directory named book1cpp to the current directory.
COPY \book1cpp*.cpp	Copy all files which have the name extension .cpp from the directory book1cpp to the current directory.
COPY sum.cpp sum_my.cpp	Copy a file named sum.cpp to file named sum_my.cpp in the current directory.
DIR	Display a list of all files and subdirectories in the current directory.
DIR *.cpp	Display a list of files whose name terminate with .cpp .
DIR s*	Display a list of files whose name begins with letter s.
DIR *sum*	Display a list of files that have the word sum in their name.
HELP MORE	Display a list of all MS-DOS commands.
HELP dir	Display information of the DIR command.
HELP path	Display information of the PATH command.
MD mycpp	Create (make) a subdirectory named mycpp in the current directory.
REN test.txt test.old	Rename a file named test.txt to test.old in the current directory.

Although the example commands above are written with uppercase letters, you can also write them with lowercase letters.

When you use the MS-DOS window in your work, it may be better for your eyes if you adjust the properties of the window. The properties can be adjusted, at least in Windows XP, by clicking the symbol in the upper left corner of the window. I always set the background color to white, text color to black, and the window size to 10 x 18. Some large programs in this book require that the screen buffer size is 80 columns x 25 rows (the same as the window size). When you modify the properties of the MS-DOS window, remember to press the OK button and make the modifications permanent.

2.3 Getting a C++ compiler for your computer

As you have now seen a couple of computer programs, you may be eager to experiment with them on a computer. It is best that you have a personal computer with the Windows operating system at your disposal when you continue studying with this book, and on your computer you need to have a C++ compiler to transform C++ source programs into an executable form.

The C++ programming language is a general-purpose language that can be used to produce programs for different computers. There thus exist many compilers for the C++ language. Many of the compilers are commercial products, but some compilers are freely available on the Internet. One free compiler is offered by the Borland Corporation, and I recommend that you install that compiler into your PC.

To get the free Borland C++ compiler, you need an Internet connection on your PC. The Internet connection should preferably be a fast one because the file that you have to download from Borland's web pages has a size of more than 8 MB (megabytes). Downloading a file with this size can take something like half an hour with a modem-based Internet connection. If you have trouble downloading the compiler from the Borland's web pages, you should consult somebody who knows more about downloading larger files from the Internet.

Before you start downloading the free compiler, you need to have a directory (folder) to store temporary files on your computer. A temporary directory is also necessary later when you download other material from the Internet. You can create a temporary directory named **D:\TEMP** by writing the following commands in an MS-DOS window

```
D:
CD \
MD TEMP
```

When you are ready to get the free Borland C++ compiler, go to the Internet address

```
http://www.borland.com/bcppbuilder/freecompiler/
```

with your Internet browser (e.g. Netscape or Internet Explorer). Then you must follow the instructions that are given on the Borland's Internet pages. The free compiler is called C++ Builder Compiler on Borland's pages. Before you can get the free C++ compiler, you must become a member in the Borland Developer Network, and you have to fill out a form where you give your e-mail address and some other information about yourself. The Borland Corporation may advertise its products by sending you e-mails, but in my opinion that is a low price for an excellent C++ compiler. Logging in to the Borland Developer Network requires that you invent an exotic login name which does not clash with other login names.

When you have given the requested information on Borland's web pages, you can download the compiler, which is a single file named **freecommandLinetools.exe**. You should store this file in the **D:\TEMP** directory on your computer. The file that contains the compiler has that name because it is a compiler that can be used on the command line in an MS-DOS window. The file contains also tools other than just the compiler.

After you have received the file from Borland, you can install the compiler by selecting **Start > My Computer > D: > TEMP > freecommandLinetools.exe**. When you click the icon that represents the compiler file, the installation process starts. During the installation process, you may see various information and need to click the Next button to proceed.

At some phase of the installation process, you are asked for the name of the folder where the compiler should be installed. (The word "folder" means the same as the word "directory".) I suggest that you install the compiler in a folder (directory) named

```
D:\BCC55
```

All the advice given below is based on the assumption that this is the installation folder. So

you have to type in the above folder name, and later on you have to give the installation program the permission to create the new folder. After you have given the installation program the necessary information, it starts installing the compiler. Many files are copied to the new folder, and the installation takes something like a minute.

When the installation program stops, the Borland C++ 5.5 compiler is installed, but still a few things have to be done before the compiler is fully operational. If you installed the compiler in the directory **D:\BCC55**, you should copy two configuration files from this book's Internet pages to the subdirectory named **D:\BCC55\BIN**. To do this, you have to go to the Internet address

<http://www.naturalprogramming.com/borlandcfgfiles/>

with your Internet browser. You should use the Netscape browser in this operation. (See the box on page 24 if you do not have the Netscape browser.) When the above address is selected, you see two files, **bcc32.cfg** and **ilink32.cfg**, with your browser. You can store these files locally to your computer by first clicking the file name so that the browser shows the contents of the file, and then doing a **File > Save As** operation. While doing the **Save As** operation, you are asked to give the location where the files should be stored. You must store the files in the directory (folder) **D:\BCC55\BIN**. You must store the two files separately. (Note that the **bcc32.cfg** and **ilink32.cfg** files that you can obtain from the above Internet address work only if your C++ compiler is installed in directory **D:\BCC55**. If you installed the compiler in some other directory, you should read the **readme.txt** file in that directory in order to create appropriate **.cfg** files.)

The last thing to do in making your Borland C++ compiler operational is to ensure that the compiler is found when you invoke the compiler in an MS-DOS window. As the executable file of your compiler is now in directory (folder) **D:\BCC55\BIN**, you have to tell to your computer's operating system that it should search the compiler in that directory. This can be achieved, for example, in one of the following ways:

- If you are using the modern Windows XP (Home Edition) operating system, you should first select **Control Panel > Performance and Maintenance > System > Advanced > Environment Variables**, and then you should check if User Variable named **PATH** is defined. If the **PATH** variable is defined, you should modify its value so that the value of the variable ends like

```
other directories ; D:\bcc55\bin
```

If the **PATH** variable is not defined, you should define it and put **D:\BCC55\BIN** as its value. The **PATH** variable is a list of directory names separated by semicolons. The operating system uses these directories when it searches executable programs. After you have done the above modification and saved the new settings, the operating system knows that it should also search in the directory named **D:\BCC55\BIN**.

- If you are using Windows NT or Windows 2000 operating system, the User Variable **PATH** must be set in the same way as in the case of Windows XP. In Windows NT, for example, you must select **Settings > Control Panel > System > Environment** to set the variables. (Note that if a Windows NT/2000/XP computer is well protected, you may need to have administrative privileges to modify the variables.)
- If you are using an older Windows operating system like Windows 98, you should modify a file named **autoexec.bat** that is in the root of the **C:** drive. You should write the line

```
path = %path%;D:\bcc55\bin
```

at the end of the **autoexec.bat** file. You can modify the file by using the standard Windows text editor called Notepad. You should restart your computer after modifying the **autoexec.bat** file.

After you have done one of the above modifications, you can test whether your C++ compiler can be invoked from an MS-DOS window. Therefore, you should first close all possibly open MS-DOS windows, and then open a new MS-DOS window and write on the command line

```
bcc32 test.cpp
```

If you get a response like

```
Borland C++ 5.5.1 for Win32 Copyright (c)1993,2000 Borland
Error E2194: Could not find file 'test.cpp'
```

your compiler works, although it says that it cannot find the **.cpp** file that was given on the command line. If the above command resulted in a response where "Borland C++" is not mentioned, it is likely that something went wrong with the installation of the compiler, or the PATH is not set correctly.

If your compiler does not work, you can check if the search path is correctly set by typing

```
path
```

in an open MS-DOS window. With this MS-DOS command the current search path is displayed. If the text **D:\BCC55\BIN** is not mentioned among the directories that are on the search path, the path is not correctly set, and you have to study what went wrong. Check first that you have not made any small typing mistakes.

If your C++ compiler does not work, although the path is correctly set, it is possible that something went wrong with the installation of the compiler. In this situation you should check that there really exists a directory named **BCC55** in drive **D:**, and that directory has a subdirectory named **BIN** that contains a file named **bcc32.exe**. If you cannot find the directories, you should find out what went wrong with the installation.

Uppercase and lowercase letters in file names

When you work with files and directories in the Windows/MS-DOS operating systems, you should remember that the file and directory names can be written both in uppercase and lowercase letters. This means, for example, that the directory names **BCC55** and **bcc55** mean the same, and the file names **SUM.CPP** and **sum.cpp** can refer to the same file.

We say that the Windows/MS-DOS operating systems are not case sensitive in regard to the file and directory names. The UNIX and Linux operating systems are different in this respect. They are case sensitive, which means, for example, that a UNIX/Linux directory may contain two different files that have names **SUM.CPP** and **sum.cpp**.

The difference between uppercase and lowercase letters is important also in programming languages. You will learn that the C++ language is case sensitive, which means, for example, that names like **SOME_VARIABLE** and **some_variable** are considered different names.

Working with other C++ compilers

In this chapter you are given instructions for using the Borland C++ compiler. However, most of the C++ programs that are presented in this book can also be compiled with C++ compilers other than the Borland C++ compiler. Therefore, you can use other C++ compilers in your studies if you know how to operate them. The Microsoft Corporation provides commercial C++ compilers. If you are using a Linux or UNIX computer, you probably have a C++ compiler named **g++** installed on your machine.

If you are studying with this book in an educational institution where C++ compilers are already installed in computers that the students may use, you should probably use those compilers or ask your teacher to recommend a compiler.