Module 1

C++ Fundamentals

Table of Contents

If there is one language that defines the essence of programming today, it is C++. It is the preeminent language for the development of high-performance software. Its syntax has become the standard for professional programming languages, and its design philosophy reverberates throughout computing.

C++ is also the language from which both Java and C# are derived. Simply stated, to be a professional programmer implies competency in C++. It is the gateway to all of modern programming.

The purpose of this module is to introduce C++, including its history, its design philosophy, and several of its most important features. By far, the hardest thing about learning a programming language is the fact that no element exists in isolation. Instead, the components of the language work together. This interrelatedness makes it difficult to discuss one aspect of C++ without involving others. To help overcome this problem, this module provides a brief overview of several C++ features, including the general form of a C++ program, some basic control statements, and operators. It does not go into too many details, but rather concentrates on the general concepts common to any C++ program.

CRITICAL SKILL 1.1: A Brief History of C++

The history of C++ begins with C. The reason for this is easy to understand: C++ is built upon the foundation of C. Thus, C++ is a superset of C. C++ expanded and enhanced the C language to support object-oriented programming (which is described later in this module). C++ also added several other improvements to the C language, including an extended set of library routines. However, much of the spirit and flavor of C++ is directly inherited from C. Therefore, to fully understand and appreciate C++, you need to understand the "how and why" behind C.

C: The Beginning of the Modern Age of Programming

The invention of C defines the beginning of the modern age of programming. Its impact should not be underestimated because it fundamentally changed the way programming was approached and thought about. Its design philosophy and syntax have influenced every major language since. C was one of the major, revolutionary forces in computing.

C was invented and first implemented by Dennis Ritchie on a DEC PDP-11 using the UNIX operating system. C is the result of a development process that started with an older language called BCPL. BCPL was developed by Martin Richards. BCPL influenced a language called B, which was invented by Ken Thompson and which led to the development of C in the 1970s.

Prior to the invention of C, computer languages were generally designed either as academic exercises or by bureaucratic committees. C was different. It was designed, implemented, and developed by real, working programmers, reflecting the way they approached the job of programming. Its features were honed, tested, thought about, and rethought by the people who actually used the language. As a result, C attracted many proponents and quickly became the language of choice of programmers around the world.

C grew out of the structured programming revolution of the 1960s. Prior to structured programming, large programs were difficult to write because the program logic tended to degenerate into what is known as "spaghetti code," a tangled mass of jumps, calls, and returns that is difficult to follow. Structured languages addressed this problem by adding well-defined control statements, subroutines

with local variables, and other improvements. Using structured languages, it became possible to write moderately large programs.

Although there were other structured languages at the time, such as Pascal, C was the first to successfully combine power, elegance, and expressiveness. Its terse, yet easy-to-use syntax coupled with its philosophy that the programmer (not the language) was in charge quickly won many converts. It can be a bit hard to understand from today's perspective, but C was a breath of fresh air that programmers had long awaited. As a result, C became the most widely used structured programming language of the 1980s.

The Need for C++

Given the preceding discussion, you might be wondering why C++ was invented. Since C was a successful computer programming language, why was there a need for something else? The answer is complexity. Throughout the history of programming, the increasing complexity of programs has driven the need for better ways to manage that complexity. C++ is a response to that need. To better understand the correlation between increasing program complexity and computer language development, consider the following.

Approaches to programming have changed dramatically since the invention of the computer. For example, when computers were first invented, programming was done by using the computer's front panel to toggle in the binary machine instructions. As long as programs were just a few hundred instructions long, this approach worked. As programs grew, assembly language was invented so that programmers could deal with larger, increasingly complex programs by using symbolic representations of the machine instructions. As programs continued to grow, high-level languages were developed to give programmers more tools with which to handle the complexity.

The first widely used computer language was, of course, FORTRAN. While FORTRAN was a very impressive first step, it is hardly a language that encourages clear, easy-to-understand programs. The 1960s gave birth to structured programming, which is the method of programming encouraged by languages such as C. With structured languages it was, for the first time, possible to write moderately complex programs fairly easily. However, even with structured programming methods, once a project reaches a certain size, its complexity exceeds what a programmer can manage. By the late 1970s, many projects were near or at this point.

In response to this problem, a new way to program began to emerge: object-oriented programming (OOP). Using OOP, a programmer could handle larger, more complex programs. The trouble was that C did not support object-oriented programming. The desire for an object-oriented version of C ultimately led to the creation of C++.

In the final analysis, although C is one of the most liked and widely used professional programming languages in the world, there comes a time when its ability to handle complexity is exceeded. Once a program reaches a certain size, it becomes so complex that it is difficult to grasp as a totality. The

purpose of C++ is to allow this barrier to be broken and to help the programmer comprehend and manage larger, more complex programs.

C++ Is Born

C++ was invented by Bjarne Stroustrup in 1979, at Bell Laboratories in Murray Hill, New Jersey. He initially called the new language "C with Classes." However, in 1983 the name was changed to C++.

Stroustrup built C++ on the foundation of C, including all of C's features, attributes, and benefits. He also adhered to C's underlying philosophy that the programmer, not the language, is in charge. At this point, it is critical to understand that Stroustrup did not create an entirely new programming language. Instead, he enhanced an already highly successful language.

Most of the features that Stroustrup added to C were designed to support object-oriented programming. In essence, C++ is the object-oriented version of C. By building upon the foundation of C, Stroustrup provided a smooth migration path to OOP. Instead of having to learn an entirely new language, a C programmer needed to learn only a few new features before reaping the benefits of the object-oriented methodology.

When creating C++, Stroustrup knew that it was important to maintain the original spirit of C, including its efficiency, flexibility, and philosophy, while at the same time adding support for object-oriented programming. Happily, his goal was accomplished. C++ still provides the programmer with the freedom and control of C, coupled with the power of objects.

Although C++ was initially designed to aid in the management of very large programs, it is in no way limited to this use. In fact, the object-oriented attributes of C++ can be effectively applied to virtually any programming task. It is not uncommon to see C++ used for projects such as editors, databases, personal file systems, networking utilities, and communication programs. Because C++ shares C's efficiency, much high-performance systems software is constructed using C++. Also, C++ is frequently the language of choice for Windows programming.

The Evolution of C++

Since C++ was first invented, it has undergone three major revisions, with each revision adding to and altering the language. The first revision was in 1985 and the second in 1990. The third occurred during the C++ standardization process. Several years ago, work began on a standard for C++. Toward that end, a joint ANSI (American National Standards Institute) and ISO (International Standards Organization) standardization committee was formed. The first draft of the proposed standard was created on January 25, 1994. In that draft, the ANSI/ISO C++ committee (of which I was a member) kept the features first defined by Stroustrup and added some new ones. But, in general, this initial draft reflected the state of C++ at the time.

Soon after the completion of the first draft of the C++ standard, an event occurred that caused the standard to be greatly expanded: the creation of the Standard Template Library (STL) by Alexander Stepanov. The STL is a set of generic routines that you can use to manipulate data. It is both powerful

and elegant. But it is also quite large. Subsequent to the first draft, the committee voted to include the STL in the specification for C++. The addition of the STL expanded the scope of C++ well beyond its original definition. While important, the inclusion of the STL, among other things, slowed the standardization of C++.

It is fair to say that the standardization of C++ took far longer than anyone had expected. In the process, many new features were added to the language, and many small changes were made. In fact, the version of C++ defined by the ANSI/ISO C++ committee is much larger and more complex than Stroustrup's original design. The final draft was passed out of committee on November 14, 1997, and an ANSI/ISO standard for C++ became a reality in 1998. This is the specification for C++ that is usually referred to as Standard C++.

The material in this book describes Standard C++. This is the version of C++ supported by all mainstream C++ compilers, including Microsoft's Visual C++. Thus, the code and information in this book are fully portable.

CRITICAL SKILL 1.2: How C++ Relates to Java and C#

In addition to C++, there are two other important, modern programming languages: Java and C#. Java was developed by Sun Microsystems, and C# was created by Microsoft. Because there is sometimes confusion about how these two languages relate to C++, a brief discussion of their relationship is in order.

C++ is the parent for both Java and C#. Although both Java and C# added, removed, and modified various features, in total the syntax for these three languages is nearly identical. Furthermore, the object model used by C++ is similar to the ones used by Java and C#. Finally, the overall "look and feel" of these languages is very similar. This means that once you know C++, you can easily learn Java or C#. The opposite is also true. If you know Java or C#, learning C++ is easy. This is one reason that Java and C# share C++'s syntax and object model; it facilitated their rapid adoption by legions of experienced C++ programmers.

The main difference between C++, Java, and C# is the type of computing environment for which each is designed. C++ was created to produce high-performance programs for a specific type of CPU and operating system. For example, if you want to write a program that runs on an Intel Pentium under the Windows operating system, then C++ is the best language to use.

Ask the Expert

Q: How do Java and C# create cross-platform, portable programs, and why can't C++ do the same?

A: Java and C# can create cross-platform, portable programs and C++ can't because of the type of object code produced by the compiler. In the case of C++, the output from the compiler is machine code

that is directly executed by the CPU. Thus, it is tied to a specific CPU and operating system. If you want to run a C++ program on a different system, you need to recompile it into machine code specifically targeted for that environment. To create a C++ program that would run in a variety of environments, several different executable versions of the program are needed.

Java and C# achieve portability by compiling a program into a pseudocode, intermediate language. In the case of Java, this intermediate language is called bytecode. For C#, it is called Microsoft Intermediate Language (MSIL). In both cases, this pseudocode is executed by a runtime system. For Java, this runtime system is called the Java Virtual Machine (JVM). For C#, it is the Common Language Runtime (CLR). Therefore, a Java program can run in any environment for which a JVM is available, and a C# program can run in any environment in which the CLR is implemented.

Since the Java and C# runtime systems stand between a program and the CPU, Java and C# programs incur an overhead that is not present in the execution of a C++ program. This is why C++ programs usually run faster than the equivalent programs written in Java or C#.

Java and C# were developed in response to the unique programming needs of the online environment of the Internet. (C# was also designed to simplify the creation of software components.) The Internet is connected to many different types of CPUs and operating systems. Thus, the ability to produce cross-platform, portable programs became an overriding concern.

The first language to address this need was Java. Using Java, it is possible to write a program that runs in a wide variety of environments. Thus, a Java program can move about freely on the Internet. However, the price you pay for portability is efficiency, and Java programs execute more slowly than do C++ programs. The same is true for C#. In the final analysis, if you want to create high-performance software, use C++. If you need to create highly portable software, use Java or C#.

One final point: Remember that C++, Java, and C# are designed to solve different sets of problems. It is not an issue of which language is best in and of itself. Rather, it is a question of which language is right for the job at hand.

1. From what language is C++ derived?
2. What was the main factor that drove the creation of C++?
3. C++ is the parent of Java and C#. True or False?

Answer Key:

1. C++ is derived from C.
2. Increasing program complexity was the main factor that drove the creation of C++.
3. True.

CRITICAL SKILL 1.3: Object-Oriented Programming

Central to C++ is object-oriented programming (OOP). As just explained, OOP was the impetus for the creation of C++. Because of this, it is useful to understand OOP's basic principles before you write even a simple C++ program.

Object-oriented programming took the best ideas of structured programming and combined them with several new concepts. The result was a different and better way of organizing a program. In the most general sense, a program can be organized in one of two ways: around its code (what is happening) or around its data (who is being affected). Using only structured programming techniques, programs are typically organized around code. This approach can be thought of as "code acting on data."

Object-oriented programs work the other way around. They are organized around data, with the key principle being "data controlling access to code." In an object-oriented language, you define the data and the routines that are permitted to act on that data. Thus, a data type defines precisely what sort of operations can be applied to that data.

To support the principles of object-oriented programming, all OOP languages, including C++, have three traits in common: encapsulation, polymorphism, and inheritance. Let's examine each.

Encapsulation

Encapsulation is a programming mechanism that binds together code and the data it manipulates, and that keeps both safe from outside interference and misuse. In an object-oriented language, code and data can be bound together in such a way that a self-contained black box is created. Within the box are all necessary data and code. When code and data are linked together in this fashion, an object is created. In other words, an object is the device that supports encapsulation.

Ask the Expert

Q: I have heard the term method applied to a subroutine. Is a method the same as a function?

A: In general, the answer is yes. The term method was popularized by Java. What a C++ programmer calls a function, a Java programmer calls a method. C# programmers also use the term method. Because it is becoming so widely used, sometimes the term method is also used when referring to a C++

function.

Within an object, code or data or both may be private to that object or public. Private code or data is known to and accessible by only another part of the object. That is, private code or data cannot be accessed by a piece of the program that exists outside the object. When code or data is public, other parts of your program can access it even though it is defined within an object. Typically, the public parts of an object are used to provide a controlled interface to the private elements of the object.

C++'s basic unit of encapsulation is the class. A class defines the form of an object. It specifies both the data and the code that will operate on that data. C++ uses a class specification to construct objects. Objects are instances of a class. Thus, a class is essentially a set of plans that specifies how to build an object.

The code and data that constitute a class are called members of the class. Specifically, member variables, also called instance variables, are the data defined by the class. Member functions are the code that operates on that data. Function is C++'s term for a subroutine.

Polymorphism

Polymorphism (from Greek, meaning "many forms") is the quality that allows one interface to access a general class of actions. A simple example of polymorphism is found in the steering wheel of an automobile. The steering wheel (the interface) is the same no matter what type of actual steering mechanism is used. That is, the steering wheel works the same whether your car has manual steering, power steering, or rack-and-pinion steering. Thus, turning the steering wheel left causes the car to go left no matter what type of steering is used. The benefit of the uniform interface is, of course, that once you know how to operate the steering wheel, you can drive any type of car.

The same principle can also apply to programming. For example, consider a stack (which is a first-in, last-out list). You might have a program that requires three different types of stacks. One stack is used for integer values, one for floating-point values, and one for characters. In this case, the algorithm that implements each stack is the same, even though the data being stored differs. In a non–object-oriented language, you would be required to create three different sets of stack routines, with each set using different names. However, because of polymorphism, in C++ you can create one general set of stack routines that works for all three situations. This way, once you know how to use one stack, you can use them all.

More generally, the concept of polymorphism is often expressed by the phrase "one interface, multiple methods." This means that it is possible to design a generic interface to a group of related activities. Polymorphism helps reduce complexity by allowing the same interface to specify a general class of action. It is the compiler's job to select the specific action (that is, method) as it applies to each situation. You, the programmer, don't need to do this selection manually. You need only remember and utilize the general interface.

Inheritance

Inheritance is the process by which one object can acquire the properties of another object. This is important because it supports the concept of hierarchical classification. If you think about it, most knowledge is made manageable by hierarchical (that is, top-down) classifications. For example, a Red Delicious apple is part of the classification apple, which in turn is part of the fruit class, which is under the larger class food. That is, the food class possesses certain qualities (edible, nutritious, and so on) which also, logically, apply to its subclass, fruit. In addition to these qualities, the fruit class has specific characteristics (juicy, sweet, and so on) that distinguish it from other food. The apple class defines those qualities specific to an apple (grows on trees, not tropical, and so on). A Red Delicious apple would, in turn, inherit all the qualities of all preceding classes and would define only those qualities that make it unique.

Without the use of hierarchies, each object would have to explicitly define all of its characteristics. Using inheritance, an object need only define those qualities that make it unique within its class. It can inherit its general attributes from its parent. Thus, it is the inheritance mechanism that makes it possible for one object to be a specific instance of a more general case.

1. Name the principles of OOP.
2. What is the basic unit of encapsulation in C++?
3. What is the commonly used term for a subroutine in C++?

Answer Key:

1. Encapsulation, polymorphism, and inheritance are the principles of OOP.
2. The class is the basic unit of encapsulation in C++.
3. Function is the commonly used term for a subroutine in C++.

Ask the Expert

Q: You state that object-oriented programming (OOP) is an effective way to manage large programs. However, it seems that OOP might add substantial overhead to relatively small ones. As it relates to C++,

is this true?

A: No. A key point to understand about C++ is that it allows you to write object-oriented programs, but does not require you to do so. This is one of the important differences between C++ and Java/C#, which employ a strict object-model in which every program is, to at least a small extent, object oriented. C++ gives you the option. Furthermore, for the most part, the object-oriented features of C++ are transparent at runtime, so little (if any) overhead is incurred.

CRITICAL SKILL 1.4: A First Simple Program

Now it is time to begin programming. Let's start by compiling and running the short sample C++ program shown here:

```
/*
      This is a simple C++ program.
      Call this file Sample.cpp.
*/

#include <iostream>
using namespace std;

// A C++ program begins at main().
int main()
{
      cout << "C++ is power programming.";

      return 0;
}
```

You will follow these three steps:

1. Enter the program.
2. Compile the program.
3. Run the program.

Before beginning, let's review two terms: source code and object code. Source code is the human-readable form of the program. It is stored in a text file. Object code is the executable form of the program created by the compiler.

Entering the Program

The programs shown in this book are available from Osborne's web site: www.osborne.com. However, if you want to enter the programs by hand, you are free to do so. Typing in the programs yourself often helps you remember the key concepts. If you choose to enter a program by hand, you must use a text editor, not a word processor. Word processors typically store format information along with text. This format information will confuse the C++ compiler. If you are using a Windows platform, then you can use WordPad, or any other programming editor that you like.

The name of the file that holds the source code for the program is technically arbitrary. However, C++ programs are normally contained in files that use the file extension .cpp. Thus, you can call a C++ program file by any name, but it should use the .cpp extension. For this first example, name the source file Sample.cpp so that you can follow along. For most of the other programs in this book, simply use a name of your own choosing.

Compiling the Program

How you will compile Sample.cpp depends upon your compiler and what options you are using. Furthermore, many compilers, such as Microsoft's Visual C++ Express Edition which you can download for free, provide two different ways for compiling a program: the command-line compiler and the Integrated Development Environment (IDE). Thus, it is not possible to give generalized instructions for compiling a C++ program. You must consult your compiler's instructions.

The preceding paragraph notwithstanding, if you are using Visual C++, then the easiest way to compile and run the programs in this book is to use the command-line compilers offered by these environments. For example, to compile Sample.cpp using Visual C++, you will use this command line:

```
C:\...>cl -GX Sample.cpp
```

The -GX option enhances compilation. To use the Visual C++ command-line compiler, you must first execute the batch file VCVARS32.BAT, which is provided by Visual C++. (Visual Studio also provides a ready-to-use command prompt environment that can be activated by selecting Visual Studio Command Prompt from the list of tools shown under the Microsoft Visual Studio entry in the Start | Programs menu of the taskbar.)

The output from a C++ compiler is executable object code. For a Windows environment, the executable file will use the same name as the source file, but have the .exe extension. Thus, the executable version of Sample.cpp will be in Sample.exe.

Run the Program

After a C++ program has been compiled, it is ready to be run. Since the output from a C++ compiler is executable object code, to run the program, simply enter its name at the command prompt. For example, to run Sample.exe, use this command line:

```
C:\...>Sample
```

When run, the program displays the following output:

```
C++ is power programming.
```

If you are using an Integrated Development Environment, then you can run a program by selecting Run from a menu. Consult the instructions for your specific compiler. For the programs in this book, it is usually easier to compile and run from the command line.

One last point: The programs in this book are console based, not window based. That is, they run in a Command Prompt session. C++ is completely at home with Windows programming. Indeed, it is the most commonly used language for Windows development. However, none of the programs in this book use the Windows Graphic User Interface (GUI). The reason for this is easy to understand: Windows programs are, by their nature, large and complex. The overhead required to create even a minimal Windows skeletal program is 50 to 70 lines of code. To write Windows programs that demonstrate the features of C++ would require hundreds of lines of code each. In contrast, console-based programs are much shorter and are the type of programs normally used to teach programming. Once you have mastered C++, you will be able to apply your knowledge to Windows programming with no trouble.

The First Sample Program Line by Line

Although Sample.cpp is quite short, it includes several key features that are common to all C++ programs. Let's closely examine each part of the program. The program begins with the lines

```
/*

This is a simple C++ program.

Call this file Sample.cpp.

*/
```

This is a comment. Like most other programming languages, C++ lets you enter a remark into a program's source code. The contents of a comment are ignored by the compiler. The purpose of a comment is to describe or explain the operation of a program to anyone reading its source code. In the case of this comment, it identifies the program. In more complex programs, you will use comments to help explain what each feature of the program is for and how it goes about doing its work. In other words, you can use comments to provide a "play-by-play" description of what your program does.

In C++, there are two types of comments. The one you've just seen is called a multiline comment. This type of comment begins with a /* (a slash followed by an asterisk). It ends only when a */ is encountered. Anything between these two comment symbols is completely ignored by the compiler. Multiline comments may be one or more lines long. The second type of comment (single-line) is found a little further on in the program and will be discussed shortly.

The next line of code looks like this:

```
#include <iostream>
```

The C++ language defines several headers, which contain information that is either necessary or useful to your program. This program requires the header iostream, which supports the C++ I/O system. This header is provided with your compiler. A header is included in your program using the #include directive. Later in this book, you will learn more about headers and why they are important.

The next line in the program is

```
using namespace std;
```

This tells the compiler to use the std namespace. Namespaces are a relatively recent addition to C++. Although namespaces are discussed in detail later in this book, here is a brief description. A namespace creates a declarative region in which various program elements can be placed. Elements declared in one namespace are separate from elements declared in another. Namespaces help in the organization of large programs. The using statement informs the compiler that you want to use the std namespace. This is the namespace in which the entire Standard C++ library is declared. By using the std namespace, you simplify access to the standard library. (Since namespaces are relatively new, an older compiler may not support them. If you are using an older compiler, see Appendix B, which describes an easy work-around.)

The next line in the program is

```
// A C++ program begins at main().
```

This line shows you the second type of comment available in C++: the single-line comment. Single-line comments begin with // and stop at the end of the line. Typically, C++ programmers use multiline comments when writing larger, more detailed commentaries, and single-line comments when short remarks are needed. This is, of course, a matter of personal style.

The next line, as the preceding comment indicates, is where program execution begins.

```
int main()
```

All C++ programs are composed of one or more functions. As explained earlier, a function is a subroutine. Every C++ function must have a name, and the only function that any C++ program must include is the one shown here, called main( ). The main( ) function is where program execution begins and (most commonly) ends. (Technically speaking, a C++ program begins with a call to main( ) and, in most cases, ends when main( ) returns.) The opening curly brace on the line that follows main( ) marks the start of the main( ) function code. The int that precedes main( ) specifies the type of data returned by main( ). As you will learn, C++ supports several built-in data types, and int is one of them. It stands for integer.

The next line in the program is

```
cout << "C++ is power programming.";
```

This is a console output statement. It causes the message C++ is power programming. to be displayed on the screen. It accomplishes this by using the output operator <<. The << operator causes whatever expression is on its right side to be output to the device specified on its left side. cout is a predefined identifier that stands for console output and generally refers to the computer's screen. Thus, this statement causes the message to be output to the screen. Notice that this statement ends with a semicolon. In fact, all C++ statements end with a semicolon.

The message "C++ is power programming." is a string. In C++, a string is a sequence of characters enclosed between double quotes. Strings are used frequently in C++.

The next line in the program is

```
return 0;
```

This line terminates main( ) and causes it to return the value 0 to the calling process (which is typically the operating system). For most operating systems, a return value of 0 signifies that the program is terminating normally. Other values indicate that the program is terminating because of some error. return is one of C++'s keywords, and it is used to return a value from a function. All of your programs should return 0 when they terminate normally (that is, without error).

The closing curly brace at the end of the program formally concludes the program.

Handling Syntax Errors

If you have not yet done so, enter, compile, and run the preceding program. As you may know from previous programming experience, it is quite easy to accidentally type something incorrectly when entering code into your computer. Fortunately, if you enter something incorrectly into your program, the compiler will report a syntax error message when it tries to compile it. Most C++ compilers attempt to make sense out of your source code no matter what you have written. For this reason, the error that is reported may not always reflect the actual cause of the problem. In the preceding program, for example, an accidental omission of the opening curly brace after main( ) may cause the compiler to report the cout statement as the source of a syntax error. When you receive syntax error messages, be prepared to look at the last few lines of code in your program in order to find the error.

Ask the Expert

Q: In addition to error messages, my compiler offers several types of warning messages. How do warnings differ from errors, and what type of reporting should I use?

A: In addition to reporting fatal syntax errors, most C++ compilers can also report several types of warning messages. Error messages report things that are unequivocally wrong in your program, such as forgetting a semicolon. Warnings point out suspicious but technically correct code. You, the

programmer, then decide whether the suspicion is justified.

Warnings are also used to report such things as inefficient constructs or the use of obsolete features. Generally, you can select the specific type of warnings that you want to see. The programs in this book are in compliance with Standard C++, and when entered correctly, they will not generate any troublesome warning messages.

For the examples in this book, you will want to use your compiler's default (or "normal") error reporting. However, you should examine your compiler's documentation to see what options you have at your disposal. Many compilers have sophisticated features that can help you spot subtle errors before they become big problems. Understanding your compiler's error reporting system is worth your time and effort.

1. Where does a C++ program begin execution?
2. What is cout?
3. What does #include <iostream> do?

Answer Key:

1. A C++ program begins execution with main( ).
2. cout is a predefined identifier that is linked to console output.
3. It includes the header <iostream>, which supports I/O.

CRITICAL SKILL 1.5: A Second Simple Program

Perhaps no other construct is as fundamental to programming as the variable. A variable is a named memory location that can be assigned a value. Further, the value of a variable can be changed during the execution of a program. That is, the content of a variable is changeable, not fixed.

The following program creates a variable called length, gives it the value 7, and then displays the message "The length is 7" on the screen.

```
// Using a variable.

#include <iostream>
using namespace std;

int main()
{
  int length; // this declares a variable ◄——— Declare a variable.

  length = 7; // this assigns 7 to length ◄——— Assign length a value.

  cout << "The length is ";
  cout << length; // This displays 7 ◄——— Output the value in length.

  return 0;
}
```

As mentioned earlier, the names of C++ programs are arbitrary. Thus, when you enter this program, select a filename to your liking. For example, you could give this program the name VarDemo.cpp.

This program introduces two new concepts. First, the statement

```
int length; // this declares a variable
```

declares a variable called length of type integer. In C++, all variables must be declared before they are used. Further, the type of values that the variable can hold must also be specified. This is called the type of the variable. In this case, length may hold integer values. These are whole number values whose range will be at least –32,768 through 32,767. In C++, to declare a variable to be of type integer, precede its name with the keyword int. Later, you will see that C++ supports a wide variety of built-in variable types. (You can create your own data types, too.)

The second new feature is found in the next line of code:

```
length = 7; // this assigns 7 to length
```

As the comment suggests, this assigns the value 7 to length. In C++, the assignment operator is the single equal sign. It copies the value on its right side into the variable on its left. After the assignment, the variable length will contain the number 7.

The following statement displays the value of length:

```
cout << length; // This displays 7
```

In general, if you want to display the value of a variable, simply put it on the right side of << in a cout statement. In this specific case, because length contains the number 7, it is this number that is displayed

on the screen. Before moving on, you might want to try giving length other values and watching the results.

CRITICAL SKILL 1.6: Using an Operator

Like most other computer languages, C+ supports a full range of arithmetic operators that enable you to manipulate numeric values used in a program. They include those shown here:

+	Addition
–	Subtraction
*	Multiplication
/	Division

These operators work in C++ just like they do in algebra.

The following program uses the * operator to compute the area of a rectangle given its length and the width.

```
// Using an operator.

#include <iostream>
using namespace std;

int main()
{
  int length; // this declares a variable
  int width;  // this declares another variable
  int area;   // this does, too

  length = 7; // this assigns 7 to length
  width = 5;  // this assigns 5 to width

  area = length * width; // compute area

  cout << "The area is ";
  cout << area; // This displays 35

  return 0;
}
```

Assign the product of **length** and **width** to **area**.

This program declares three variables: length, width, and area. It assigns the value 7 to length and the value 5 to width. It then computes the product and assigns that value to area. The program outputs the following:

```
The area is 35
```

In this program, there is actually no need for the variable area. For example, the program can be rewritten like this:

```
// A simplified version of the area program.

#include <iostream>
using namespace std;

int main()
{
  int length; // this declares a variable
  int width;  // this declares another variable

  length = 7; // this assigns 7 to length

  width = 5;  // this assigns 5 to width

  cout << "The area is ";
  cout << length * width; // This displays 35 <-- Output length * width directly.

  return 0;
}
```

In this version, the area is computed in the cout statement by multiplying length by width. The result is then output to the screen.

One more point before we move on: It is possible to declare two or more variables using the same declaration statement. Just separate their names by commas. For example, length, width, and area could have been declared like this:

```
int length, width, area; // all declared using one statement
```

Declaring two or more variables in a single statement is very common in professionally written C++ code.

1. Must a variable be declared before it is used?
2. Show how to assign the variable min the value 0.
3. Can more than one variable be declared in a single declaration statement?

Answer Key:

1. Yes, in C++ variables must be declared before they are used.
2. min = 0;
3. Yes, two or more variables can be declared in a single declaration statement.

CRITICAL SKILL 1.7: Reading Input from the Keyboard

The preceding examples have operated on data explicitly specified in the program. For example, the area program just shown computes the area of a rectangle that is 7 by 5, and these dimensions are part of the program itself. Of course, the calculation of a rectangle's area is the same no matter what its size, so the program would be much more useful if it would prompt the user for the dimensions of the rectangle, allowing the user to enter them using the keyboard.

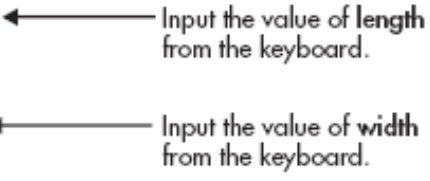
To enable the user to enter data into a program from the keyboard, you will use the >> operator. This is the C++ input operator. To read from the keyboard, use this general form

```
cin >> var;
```

Here, cin is another predefined identifier. It stands for console input and is automatically supplied by C++. By default, cin is linked to the keyboard, although it can be redirected to other devices. The variable that receives input is specified by var.

Here is the area program rewritten to allow the user to enter the dimensions of the rectangle:

```
/*
   An interactive program that computes
   the area of a rectangle.
*/

#include <iostream>
using namespace std;

int main()
{
  int length; // this declares a variable
  int width;  // this declares another variable

  cout << "Enter the length: ";
  cin >> length; // input the length
  cout << "Enter the width: ";
  cin >> width;  // input the width

  cout << "The area is ";
  cout << length * width; // display the area

  return 0;
}
```



Here is a sample run:

```
Enter the length: 8
Enter the width: 3
The area is 24
```

Pay special attention to these lines:

```
cout << "Enter the length: ";
cin >> length; // input the length
```

The cout statement prompts the user. The cin statement reads the user's response, storing the value in length. Thus, the value entered by the user (which must be an integer in this case) is put into the variable that is on the right side of the >> (in this case, length). Thus, after the cin statement executes, length will contain the rectangle's length. (If the user enters a nonnumeric response, length will be zero.) The statements that prompt and read the width work in the same way.

Some Output Options

So far, we have been using the simplest types of cout statements. However, cout allows much more sophisticated output statements. Here are two useful techniques. First, you can output more than one

piece of information using a single cout statement. For example, in the area program, these two lines are used to display the area:

```
cout << "The area is ";

cout << length * width;
```

These two statements can be more conveniently coded, as shown here:

```
cout << "The area is " << length * width;
```

This approach uses two output operators within the same cout statement. Specifically, it outputs the string “The area is” followed by the area. In general, you can chain together as many output operations as you like within one output statement. Just use a separate << for each item.

Second, up to this point, there has been no occasion to advance output to the next line— that is, to execute a carriage return–linefeed sequence. However, the need for this will arise very soon. In C++, the carriage return–linefeed sequence is generated using the newline character. To put a newline character into a string, use this code: \n (a backslash followed by a lowercase n). To see the effect of the \n, try the following program:

```
/*
      This program demonstrates the \n code, which generates a new
      line.
*/

#include <iostream>
using namespace std;
int main()

{
      cout << "one\n";
      cout << "two\n";
      cout << "three";
      cout << "four";

      return 0;

}
```

This program produces the following output:

```
one
two
threefour
```

The newline character can be placed anywhere in the string, not just at the end. You might want to try experimenting with the newline character now, just to make sure you understand exactly what it does.

1. What is C++'s input operator?
2. To what device is cin linked by default?
3. What does \n stand for?

Answer Key:

1. The input operator is >>.
2. cin is linked to the keyboard by default.
3. The \n stands for the newline character.

Another Data Type

In the preceding programs, variables of type int were used. However, a variable of type int can hold only whole numbers. Thus, it cannot be used when a fractional component is required. For example, an int variable can hold the value 18, but not the value 18.3. Fortunately, int is only one of several data types defined by C++. To allow numbers with fractional components, C++ defines two main flavors of floating-point types: float and double, which represent single- and double-precision values, respectively. Of the two, double is probably the most commonly used.

To declare a variable of type double, use a statement similar to that shown here:

```
double result;
```

Here, result is the name of the variable, which is of type double. Because result has a floating-point type, it can hold values such as 88.56, 0.034, or –107.03.

To better understand the difference between int and double, try the following program:

```
/*
   This program illustrates the differences
   between int and double.
*/

#include <iostream>
using namespace std;

int main() {
  int ivar;    // this declares an int variable
  double dvar; // this declares a floating-point variable

  ivar = 100; // assign ivar the value 100

  dvar = 100.0; // assign dvar the value 100.0

  cout << "Original value of ivar: " << ivar << "\n";
  cout << "Original value of dvar: " << dvar << "\n";

  cout << "\n"; // print a blank line ←——— Output a blank line.

  // now, divide both by 3
  ivar = ivar / 3;
  dvar = dvar / 3.0;

  cout << "ivar after division: " << ivar << "\n";
  cout << "dvar after division: " << dvar << "\n";

  return 0;
}
```

The output from this program is shown here:

```
Original value of ivar: 100
Original value of dvar: 100

ivar after division: 33
dvar after division: 33.3333
```

Ask the Expert

Q: Why does C++ have different data types for integers and floating-point values? That is, why aren't all numeric values just the same type?

A: C++ supplies different data types so that you can write efficient programs. For example, integer arithmetic is faster than floating-point calculations. Thus, if you don't need fractional values, then you don't need to incur the overhead associated with types float or double. Also, the amount of memory required for one type of data might be less than that required for another. By supplying different types, C++ enables you to make the best use of system resources. Finally, some algorithms require (or at least benefit from) the use of a specific type of data. C++ supplies a number of built-in types to give you the greatest flexibility.

As you can see, when ivar is divided by 3, a whole-number division is performed and the outcome is 33—the fractional component is lost. However, when dvar is divided by 3, the fractional component is preserved.

There is one other new thing in the program. Notice this line:

```
cout << "\n"; // print a blank line
```

It outputs a newline. Use this statement whenever you want to add a blank line to your output.

Project 1-1 Converting Feet to Meters

Although the preceding sample programs illustrate several important features of the C++ language, they are not very useful. You may not know much about C++ at this point, but you can still put what you have learned to work to create a practical program. In this project, we will create a program that converts feet to meters. The program prompts the user for the number of feet. It then displays that value converted into meters.

A meter is equal to approximately 3.28 feet. Thus, we need to use floating-point data. To perform the conversion, the program declares two double variables. One will hold the number of feet, and the second will hold the conversion to meters.

Step by Step

1. Create a new C++ file called FtoM.cpp. (Remember, in C++ the name of the file is arbitrary, so you can use another name if you like.)

2. Begin the program with these lines, which explain what the program does, include the iostream header, and specify the std namespace.

```
/*
   Project 1-1
   This program converts feet to meters.
   Call this program FtoM.cpp.
*/
```

```
#include <iostream>
using namespace std;
```

3. Begin main( ) by declaring the variables f and m:

```
int main()
{
double f; // holds the length in feet
double m; // holds the conversion to meters
```

4. Add the code that inputs the number of feet:

```
cout << "Enter the length in feet: ";
cin >> f; // read the number of feet
```

5. Add the code that performs the conversion and displays the result:

```
m = f / 3.28; // convert to meters
cout << f << " feet is " << m << " meters.";
```

6. Conclude the program, as shown here:

```
return 0; }
```

7. Your finished program should look like this:

```
/*
   Project 1-1
   This program converts feet to meters.
   Call this program FtoM.cpp.
*/

#include <iostream>
using namespace std;

int main()
{
   double f; // holds the length in feet
   double m; // holds the conversion to meters

   cout << "Enter the length in feet: ";
   cin >> f; // read the number of feet

   m = f / 3.28; // convert to meters
   cout << f << " feet is " << m << " meters.";
```

```
    return 0;
}
```

8. Compile and run the program. Here is a sample run:

```
Enter the length in feet: 5 5 feet is 1.52439 meters.
```

9. Try entering other values. Also, try changing the program so that it converts meters to feet.

1. What is C++'s keyword for the integer data type?
2. What is double?
3. How do you output a newline?

Answer Key:

1. The integer data type is int.
2. double is the keyword for the double floating-point data type.
3. To output a newline, use \n.

CRITICAL SKILL 1.8: Two Control Statements

Inside a function, execution proceeds from one statement to the next, top to bottom. It is possible, however, to alter this flow through the use of the various program control statements supported by C++. Although we will look closely at control statements later, two are briefly introduced here because we will be using them to write sample programs.

The if Statement

You can selectively execute part of a program through the use of C++'s conditional statement: the if. The if statement works in C++ much like the IF statement in any other language. For example, it is syntactically identical to the if statements in C, Java, and C#. Its simplest form is shown here:

if(condition) statement;

where condition is an expression that is evaluated to be either true or false. In C++, true is nonzero and false is zero. If the condition is true, then the statement will execute. If it is false, then the statement will

not execute. For example, the following fragment displays the phrase 10 is less than 11 on the screen because 10 is less than 11.

```
if(10 < 11) cout << "10 is less than 11";
```

However, consider the following:

```
if(10 > 11) cout << "this does not display";
```

In this case, 10 is not greater than 11, so the cout statement is not executed. Of course, the operands inside an if statement need not be constants. They can also be variables.

C++ defines a full complement of relational operators that can be used in a conditional expression. They are shown here:

Operator	Meaning
<	Less than
<=	Less than or equal
>	Greater than
>=	Greater than or equal
==	Equal to
!=	Not equal

Notice that the test for equality is the double equal sign. Here is a program that illustrates the if statement:

```
// Demonstrate the if.

#include <iostream>
using namespace std;
```

```
int main() {
  int a, b, c;

  a = 2;
  b = 3;

  if(a < b) cout << "a is less than b\n"; <-- An if statement

  // this won't display anything
  if(a == b) cout << "you won't see this\n";

  cout << "\n";

  c = a - b; // c contains -1

  cout << "c contains -1\n";
  if(c >= 0) cout << "c is non-negative\n";
  if(c < 0) cout << "c is negative\n";

  cout << "\n";

  c = b - a; // c now contains 1
  cout << "c contains 1\n";
  if(c >= 0) cout << "c is non-negative\n";
  if(c < 0) cout << "c is negative\n";

  return 0;
}
```

The output generated by this program is shown here:

```
a is less than b

c contains -1
c is negative

c contains 1
c is non-negative
```

The for Loop

You can repeatedly execute a sequence of code by creating a loop. C++ supplies a powerful assortment of loop constructs. The one we will look at here is the for loop. If you are familiar with C# or Java, then you will be pleased to know that the for loop in C++ works the same way it does in those languages. The simplest form of the for loop is shown here:

for(initialization; condition; increment) statement;

Here, initialization sets a loop control variable to an initial value. condition is an expression that is tested each time the loop repeats. As long as condition is true (nonzero), the loop keeps running. The

increment is an expression that determines how the loop control variable is incremented each time the loop repeats.

The following program demonstrates the for. It prints the numbers 1 through 100 on the screen.

```
// A program that illustrates the for loop.

#include <iostream>
using namespace std;

int main()
{
  int count;

  for(count=1; count <= 100; count=count+1)
    cout << count << " ";

  return 0;
}
```

This is a for loop.

In the loop, count is initialized to 1. Each time the loop repeats, the condition

```
count <= 100
```

is tested. If it is true, the value is output and count is increased by one. When count reaches a value greater than 100, the condition becomes false, and the loop stops running. In professionally written C++ code, you will almost never see a statement like

```
count=count+1
```

because C++ includes a special increment operator that performs this operation more efficiently. The increment operator is ++ (two consecutive plus signs). The ++ operator increases its operand by 1. For example, the preceding for statement will generally be written like this:

```
for(count=1; count <= 100; count++) cout << count << " ";
```

This is the form that will be used throughout the rest of this book.

C++ also provides a decrement operator, which is specified as – –. It decreases its operand by 1.

1. What does the **if** statement do?

2. What does the **for** statement do?
3. What are C++'s relational operators?

Answer Key:

1. if is C++'s conditional statement.
2. The for is one of C++'s loop statements.
3. The relational operators are ==, !=, <, >, <=, and >=.

CRITICAL SKILL 1.9: Using Blocks of Code

Another key element of C++ is the code block. A code block is a grouping of two or more statements. This is done by enclosing the statements between opening and closing curly braces. Once a block of code has been created, it becomes a logical unit that can be used any place that a single statement can. For example, a block can be a target of the if and for statements. Consider this if statement:

```
if(w < h) {
      v = w * h;
      w = 0;
}
```

Here, if w is less than h, then both statements inside the block will be executed. Thus, the two statements inside the block form a logical unit, and one statement cannot execute without the other also executing. The key point here is that whenever you need to logically link two or more statements, you do so by creating a block. Code blocks allow many algorithms to be implemented with greater clarity and efficiency.

Here is a program that uses a block of code to prevent a division by zero:

```
// Demonstrate a block of code.

#include <iostream>
using namespace std;

int main() {
  double result, n, d;

  cout << "Enter value: ";
  cin >> n;

  cout << "Enter divisor: ";
  cin >> d;

  // the target of this if is a block
  if(d != 0) {
    cout << "d does not equal zero so division is OK" << "\n";
    result = n / d;
    cout << n << " / " << d << " is " << result;
  }

  return 0;
}
```

The target of this if is the entire block.

Here is a sample run:

```
Enter value: 10
Enter divisor: 2
d does not equal zero so division is OK
10 / 2 is 5
```

In this case, the target of the if statement is a block of code and not just a single statement. If the condition controlling the if is true (as it is in the sample run), the three statements inside the block will be executed. Try entering a zero for the divisor and observe the result. In this case, the code inside the block is bypassed.

As you will see later in this book, blocks of code have additional properties and uses. However, the main reason for their existence is to create logically inseparable units of code.

Ask the Expert

Q: Does the use of a code block introduce any runtime inefficiencies? In other words, do the { and } consume any extra time during the execution of my program?

A: No. Code blocks do not add any overhead whatsoever. In fact, because of their ability to simplify the coding of certain algorithms, their use generally increases speed and efficiency.

Semicolons and Positioning

In C++, the semicolon signals the end of a statement. That is, each individual statement must end with a semicolon. As you know, a block is a set of logically connected statements that is surrounded by opening and closing braces. A block is not terminated with a semicolon. Since a block is a group of statements, with a semicolon after each statement, it makes sense that a block is not terminated by a semicolon; instead, the end of the block is indicated by the closing brace.

C++ does not recognize the end of the line as the end of a statement—only a semicolon terminates a statement. For this reason, it does not matter where on a line you put a statement.

For example, to C++

```
x = y;
y = y + 1;
cout << x << " " << y;
```

is the same as

```
x = y; y = y + 1; cout << x << " " << y;
```

Furthermore, the individual elements of a statement can also be put on separate lines. For example, the following is perfectly acceptable:

```
cout << "This is a long line. The sum is : " << a + b + c +
        d + e + f;
```

Breaking long lines in this fashion is often used to make programs more readable. It can also help prevent excessively long lines from wrapping.

Indentation Practices

You may have noticed in the previous examples that certain statements were indented. C++ is a free-form language, meaning that it does not matter where you place statements relative to each other on a line. However, over the years, a common and accepted indentation style has developed that allows for very readable programs. This book follows that style, and it is recommended that you do so as well. Using this style, you indent one level after each opening brace and move back out one level after each closing brace. There are certain statements that encourage some additional indenting; these will be covered later.

1. How is a block of code created? What does it do?
2. In C++, statements are terminated by a _____.
3. All C++ statements must start and end on one line. True or false?

Answer Key:

1. A block is started by a {. It is ended by a }. A block creates a logical unit of code.
2. semicolon
3. False.

Project 1-2 Generating a Table of Feet to Meter Conversions

This project demonstrates the for loop, the if statement, and code blocks to create a program that displays a table of feet-to-meters conversions. The table begins with 1 foot and ends at 100 feet. After every 10 feet, a blank line is output. This is accomplished through the use of a variable called counter that counts the number of lines that have been output. Pay special attention to its use.

Step by Step

1. Create a new file called FtoMTable.cpp.
2. Enter the following program into the file:

```
/*
   Project 1-2

   This program displays a conversion table of feet to meters.

   Call this program FtoMTable.cpp.
*/

#include <iostream>
using namespace std;

int main() {
  double f; // holds the length in feet
  double m; // holds the conversion to meters
  int counter;

  counter = 0; <-- Line counter is initially set to zero.

  for(f = 1.0; f <= 100.0; f++) {
    m = f / 3.28; // convert to meters
    cout << f << " feet is " << m << " meters.\n";

    counter++; <-- Increment the line counter with each loop iteration.

    // every 10th line, print a blank line
    if(counter == 10) { <-- If counter is 10, output a blank line.
      cout << "\n"; // output a blank line
      counter = 0; // reset the line counter
    }
  }

  return 0;
}
```

3. Notice how counter is used to output a blank line after each ten lines. It is initially set to zero outside the for loop. Inside the loop, it is incremented after each conversion. When counter equals 10, a blank line is output, counter is reset to zero, and the process repeats.

4. Compile and run the program. Here is a portion of the output that you will see. Notice that results that don't produce an even result include a fractional component.

```
1 feet is 0.304878 meters.
2 feet is 0.609756 meters.
3 feet is 0.914634 meters.
4 feet is 1.21951 meters.
5 feet is 1.52439 meters.
6 feet is 1.82927 meters.
7 feet is 2.13415 meters.
```

```
8 feet is 2.43902 meters.
9 feet is 2.7439 meters.
10 feet is 3.04878 meters.
11 feet is 3.35366 meters.
12 feet is 3.65854 meters.
13 feet is 3.96341 meters.
14 feet is 4.26829 meters.
15 feet is 4.57317 meters.
16 feet is 4.87805 meters.
17 feet is 5.18293 meters.
18 feet is 5.4878 meters.
19 feet is 5.79268 meters.
20 feet is 6.09756 meters.
21 feet is 6.40244 meters.
22 feet is 6.70732 meters.
23 feet is 7.0122 meters.
24 feet is 7.31707 meters.
25 feet is 7.62195 meters.
26 feet is 7.92683 meters.
27 feet is 8.23171 meters.
28 feet is 8.53659 meters.
29 feet is 8.84146 meters.
30 feet is 9.14634 meters.
31 feet is 9.45122 meters.
32 feet is 9.7561 meters.
33 feet is 10.061 meters.
34 feet is 10.3659 meters.
35 feet is 10.6707 meters.
36 feet is 10.9756 meters.
37 feet is 11.2805 meters.
38 feet is 11.5854 meters.
39 feet is 11.8902 meters.
40 feet is 12.1951 meters.
```

5. On your own, try changing this program so that it prints a blank line every 25 lines.

CRITICAL SKILL 1.10: Introducing Functions

A C++ program is constructed from building blocks called functions. Although we will look at the function in detail in Module 5, a brief overview is useful now. Let's begin by defining the term function: a function is a subroutine that contains one or more C++ statements.

Each function has a name, and this name is used to call the function. To call a function, simply specify its name in the source code of your program, followed by parentheses. For example, assume some function named MyFunc. To call MyFunc, you would write

```
MyFunc();
```

When a function is called, program control is transferred to that function, and the code contained within the function is executed. When the function's code ends, control is transferred back to the caller. Thus, a function performs a task for other parts of a program.

Some functions require one or more arguments, which you pass when the function is called. Thus, an argument is a value passed to a function. Arguments are specified between the opening and closing parentheses when a function is called. For example, if MyFunc( ) requires an integer argument, then the following calls MyFunc( ) with the value 2:

```
MyFunc(2);
```

When there are two or more arguments, they are separated by commas. In this book, the term argument list will refer to comma-separated arguments. Remember, not all functions require arguments. When no argument is needed, the parentheses are empty.

A function can return a value to the calling code. Not all functions return values, but many do. The value returned by a function can be assigned to a variable in the calling code by placing the call to the function on the right side of an assignment statement. For example, if MyFunc( ) returned a value, it could be called as shown here:

```
x = MyFunc(2);
```

This statement works as follows. First, MyFunc( ) is called. When it returns, its return value is assigned to x. You can also use a call to a function in an expression. For example,

```
x = MyFunc(2) + 10;
```

In this case, the return value from MyFunc( ) is added to 10, and the result is assigned to x. In general, whenever a function's name is encountered in a statement, it is automatically called so that its return value can be obtained.

To review: an argument is a value passed into a function. A return value is data that is passed back to the calling code.

Here is a short program that demonstrates how to call a function. It uses one of C++'s built-in functions, called abs( ), to display the absolute value of a number. The abs( ) function takes one argument, converts it into its absolute value, and returns the result.

```
// Use the abs() function.

#include <iostream>
#include <cstdlib>
using namespace std;

int main()
{
  int result;

  result = abs(-10); <-- The calls the abs( ) function and assigns its return value to result.

  cout << result;

  return 0;
}
```

Here, the value –10 is passed as an argument to abs( ). The abs( ) function receives the argument with which it is called and returns its absolute value, which is 10 in this case. This value is assigned to result. Thus, the program displays "10" on the screen.

Notice one other thing about the preceding program: it includes the header cstdlib. This is the header required by abs( ). Whenever you use a built-in function, you must include its header.

In general, there are two types of functions that will be used by your programs. The first type is written by you, and main( ) is an example of this type of function. Later, you will learn how to write other functions of your own. As you will see, real-world C++ programs contain many user-written functions.

The second type of function is provided by the compiler. The abs( ) function used by the preceding program is an example. Programs that you write will generally contain a mix of functions that you create and those supplied by the compiler.

When denoting functions in text, this book has used and will continue to use a convention that has become common when writing about C++. A function will have parentheses after its name. For example, if a function's name is getval, then it will be written getval( ) when its name is used in a sentence. This notation will help you distinguish variable names from function names in this book.

The C++ Libraries

As just explained, abs( ) is provided with your C++ compiler. This function and many others are found in the standard library. We will be making use of library functions in the example programs throughout this book.

C++ defines a large set of functions that are contained in the standard function library. These functions perform many commonly needed tasks, including I/O operations, mathematical computations, and

string handling. When you use a library function, the C++ compiler automatically links the object code for that function to the object code of your program.

Because the C++ standard library is so large, it already contains many of the functions that you will need to use in your programs. The library functions act as building blocks that you simply assemble. You should explore your compiler's library documentation. You may be surprised at how varied the library functions are. If you write a function that you will use again and again, it too can be stored in a library.

In addition to providing library functions, every C++ compiler also contains a class library, which is an object-oriented library. However, you will need to wait until you learn about classes and objects before you can make use of the class library.

1. What is a function?
2. A function is called by using its name. True or false?
3. What is the C++ standard function library?

Answer Key:

1. A function is a subroutine that contains one or more C++ statements.
2. True.
3. The C++ standard function library is a collection of functions supplied by all C++ compilers.

CRITICAL SKILL 1.11: The C++ Keywords

There are 63 keywords currently defined for Standard C++. These are shown in Table 1-1. Together with the formal C++ syntax, they form the C++ programming language. Also, early versions of C++ defined the overload keyword, but it is obsolete. Keep in mind that C++ is a case-sensitive language, and it requires that all keywords be in lowercase.

asm	auto	bool	break
case	catch	char	class
const	const_cast	continue	default
delete	do	double	dynamic_cast
else	enum	explicit	export
extern	false	float	for
friend	goto	if	inline
int	long	mutable	namespace
new	operator	private	protected
public	register	reinterpret_cast	return
short	signed	sizeof	static
static_cast	struct	switch	template
this	throw	true	try
typedef	typeid	typename	union
unsigned	using	virtual	void
volatile	wchar_t	while	

Table 1-1 The C++ Keywords

CRITICAL SKILL 1.12: Identifiers

In C++, an identifier is a name assigned to a function, variable, or any other user-defined item. Identifiers can be from one to several characters long. Variable names can start with any letter of the alphabet or an underscore. Next comes a letter, a digit, or an underscore. The underscore can be used to enhance the readability of a variable name, as in line_count. Uppercase and lowercase are seen as different; that is, to C++, myvar and MyVar are separate names. There is one important identifier restriction: you cannot use any of the C++ keywords as identifier names. In addition, predefined identifiers such as cout are also off limits.

Here are some examples of valid identifiers:

Test	x	y2	MaxIncr
up	_top	my_var	simpleInterest23

Remember, you cannot start an identifier with a digit. Thus, 98OK is invalid. Good programming practice dictates that you use identifier names that reflect the meaning or usage of the items being named.

Progress Check

1. Which is the keyword, for, For, or FOR?
2. A C++ identifier can contain what type of characters?
3. Are index21 and Index21 the same identifier?

Answer Key:

1. The keyword is for. In C++, all keywords are in lowercase.
2. A C++ identifier can contain letters, digits, and the underscore.
3. No, C++ is case sensitive.

Module 1 Mastery Check

1. It has been said that C++ sits at the center of the modern programming universe. Explain this statement.
2. A C++ compiler produces object code that is directly executed by the computer. True or false?
3. What are the three main principles of object-oriented programming?
4. Where do C++ programs begin execution?
5. What is a header?
6. What is <iostream>? What does the following code do?

```
#include <iostream>
```

7. What is a namespace?
8. What is a variable?
9. Which of the following variable names is/are invalid?

a. `count`

b. `_count`

c. `count27`

d. `67count`

e. `if`

10. How do you create a single-line comment? How do you create a multiline comment?

11. Show the general form of the if statement. Show the general form of the for loop.

12. How do you create a block of code?

13. The moon's gravity is about 17 percent that of Earth's. Write a program that displays a table that shows Earth pounds and their equivalent moon weight. Have the table run from 1 to 100 pounds. Output a newline every 25 pounds.

14. A year on Jupiter (the time it takes for Jupiter to make one full circuit around the Sun) takes about 12 Earth years. Write a program that converts Jovian years to Earth years. Have the user specify the number of Jovian years. Allow fractional years.

15. When a function is called, what happens to program control?

16. Write a program that averages the absolute value of five values entered by the user. Display the result.