Welcome to C++

C++ is a recent addition to the long list of programming languages now available. Experts predict that C++ will become one of the most widely used programming languages within two to three years. Scan your local computer bookstore's shelves and you will see that C++ is taking the programming world by storm. More and more companies are offering C++ compilers. In the world of PCs, both Borland and Microsoft, two of the leading names of PC software, offer full-featured C++ compilers.

Although the C++ language is fairly new, having become popular within the last three years, the designers of C++ compilers are perfecting this efficient, standardized language that should soon be compatible with almost every computer in the world. Whether you are a beginning, an intermediate, or an expert programmer, C++ has the programming tools you need to make your computer do just what *you* want it to do. This chapter introduces you to C++, briefly describes its history, compares C++ to its predecessor C, shows you the advantages of C++, and concludes by introducing you to hardware and software concepts.

What C++ Can Do for You

C++ is currently defined by American Telephone & Telegraph, Incorporated, to achieve conformity between versions of C++.

Imagine a language that makes your computer perform to your personal specifications! Maybe you have looked for a program that keeps track of your household budget—exactly as you prefer—but haven't found one. Perhaps you want to track the records of a small (or large) business with your computer, but you haven't found a program that prints reports exactly as you'd like them. Possibly you have thought of a new and innovative use for a computer and you would like to implement your idea. C++ gives you the power to develop all these uses for your computer.

If your computer could understand English, you would not have to learn a programming language. But because it does not understand English, you must learn to write instructions in a language your computer recognizes. C++ is a powerful programming language. Several companies have written different versions of C++, but almost all C++ languages available today conform to the AT&T standard. *AT&T-compatible* means the C++ language in question conforms to the standard defined by the company that invented the language, namely, American Telephone & Telegraph, Incorporated. AT&T realizes that C++ is still new and has not fully matured. The good people there just completed the AT&T C++ 3.0 standard to which software companies can conform. By developing a uniform C++ language, AT&T helps ensure that programs you write today will most likely be compatible with the C++ compilers of tomorrow.

NOTE: The AT&T C++ standard is only a suggestion. Software companies do not have to follow the AT&T standard, although most choose to do so. No typical computer standards committee has yet adopted a C++ standard language. The committees are currently working on the issue, but they are probably waiting for C++ to entrench the programming community before settling on a standard.

C++ is called a "better C than C."

Companies do not have to follow the AT&T C++ 3.0 standard. Many do, but add their own extensions and create their own version to do more work than the AT&T standard includes. If you are using the AT&T C++ standard, your program should successfully run on any other computer that also uses AT&T C++.

AT&T developed C++ as an improved version of the C programming language. C has been around since the 1970s and has matured into a solid, extremely popular programming language. *ANSI*, the American National Standards Institute, established a standard C programming specification called ANSI C. If your C compiler conforms to ANSI C, your program will work on any other computer that also has ANSI C. This compatibility between computers is so important that AT&T's C++ 3.0 standard includes almost every element of the ANSI C, plus more. In fact, the ANSI C committee often requires that a C++ feature be included in subsequent versions of C. For instance, function prototypes, a feature not found in older versions of ANSI C, is now a requirement for approval by the ANSI committee. Function prototypes did not exist until AT&T required them in their early C++ specification.

C++ By Example teaches you to program in C++. All programs conform to the AT&T C++ 2.1 standard. The differences between AT&T 2.1 and 3.0 are relatively minor for beginning programmers. As you progress in your programming skills, you will want to tackle the more advanced aspects of C++ and Version 3.0 will come more into play later. Whether you use a PC, a minicomputer, a mainframe, or a supercomputer, the C++ language you learn here should work on any that conform to AT&T C++ 2.1 and later.

There is a debate in the programming community as to whether a person should learn C before C++ or learn only C++. Because C++ is termed a "better C," many feel that C++ is an important language in its own right and can be learned just as easily as C. Actually, C++ pundits state that C++ teaches better programming habits than the plain, "vanilla" C. This book is aimed at the beginner programmer, and the author feels that C++ is a great language with which to begin. If you were to first learn C, you would have to "unlearn" a few things when you moved to C++. This book attempts to use the C++ language elements that are better than C. If you are new to programming, you learn C++ from the start. If you have a C background, you learn that C++ overcomes many of C's limitations.

When some people attempt to learn C++ (and C), even if they are programmers in other computer languages, they find that C++ can be cryptic and difficult to understand. This does not have to be the case. When taught to write clear and concise C++ code in an order that builds on fundamental programming concepts,

programmers find that C++ is no more difficult to learn or use than any other programming language. Actually, after you start using it, C++'s modularity makes it even easier to use than most other languages. Once you master the programming elements this book teaches you, you will be ready for the advanced power for which C++ was designed—*object-oriented programming* (OOP). The last chapter of this book, "Introduction to Object-Oriented Programming," offers you the springboard to move to this exciting way of writing programs.

Even if you've never programmed a computer before, you will soon understand that programming in C++ is rewarding. Becoming an expert programmer in C++—or in any other computer language—takes time and dedication. Nevertheless, you can start writing simple programs with little effort. After you learn the fundamentals of C++ programming, you can build on what you learn and hone your skills as you write more powerful programs. You also might see new uses for your computer and develop programs others can use.

The importance of C++ cannot be overemphasized. Over the years, several programming languages were designed to be "the only programming language you would ever need." PL/I was heralded as such in the early 1960s. It turned out to be so large and took so many system resources that it simply became another language programmers used, along with COBOL, FORTRAN, and many others. In the mid-1970s, Pascal was developed for smaller computers. Microcomputers had just been invented, and the Pascal language was small enough to fit in their limited memory space while still offering advantages over many other languages. Pascal became popular and is still used often today, but it never became *the* answer for all programming tasks, and it failed at being "the only programming language you would ever need."

When the mass computer markets became familiar with C in the late 1970s, C also was promoted as "the only programming language you would ever need." What has surprised so many skeptics (including this author) is that C has practically fulfilled this promise! An incredible number of programming shops have converted to C. The appeal of C's efficiency, combined with its portability among computers, makes it the language of choice. Most of

today's familiar spreadsheets, databases, and word processors are written in C. Now that C++ has improved on C, programmers are retooling their minds to think in C++ as well.

The programmer help-wanted ads seek more and more C++ programmers every day. By learning this popular language, you will be learning the latest direction of programming and keeping your skills current with the market. You have taken the first step: with this book, you learn the C++ language particulars as well as many programming tips to use and pitfalls to avoid. This book attempts to teach you to be not just a C++ programmer, but a better programmer by applying the structured, long-term programming habits that professionals require in today's business and industry.

The Background of C++

The UNIX operating system was written almost entirely in C.

Before you jump into C++, you might find it helpful to know a little about the evolution of the C++ programming language. C++ is so deeply rooted in C that you should first see where C began. Bell Labs first developed the C programming language in the early 1970s, primarily so Bell programmers could write their UNIX operating system for a new DEC (Digital Equipment Corporation) computer. Until that time, operating systems were written in assembly language, which is tedious, time-consuming, and difficult to maintain. The Bell Labs people knew they needed a higher-level programming language to implement their project quicker and create code that was easier to maintain.

Because other high-level languages at the time (COBOL, FORTRAN, PL/I, and Algol) were too slow for an operating system's code, the Bell Labs programmers decided to write their own language. They based their new language on Algol and BCPL. Algol is still used in the European markets, but is not used much in America. BCPL strongly influenced C, although it did not offer the various data types that the makers of C required. After a few versions, these Bell programmers developed a language that met their goals well. C is efficient (it is sometimes called a high, low-level language due to its speed of execution), flexible, and contains the proper language elements that enable it to be maintained over time.

In the 1980s, Bjourn Stroustrup, working for AT&T, took the C language to its next progression. Mr. Stroustrup added features to compensate for some of the pitfalls C allowed and changed the way programmers view programs by adding object-orientation to the language. The object-orientation aspect of programming started in other languages, such as Smalltalk. Mr. Stroustrup realized that C++ programmers needed the flexibility and modularity offered by a true OOP programming language.

C++ Compared with Other Languages

C++ requires more stringent data-type checking than does C.

If you have programmed before, you should understand a little about how C++ differs from other programming languages on the market. C++ is efficient and has much stronger typing than its C predecessor. C is known as a *weakly typed* language; variable data types do not necessarily have to hold the same type of data. (Function prototyping and type casting help to alleviate this problem.)

For example, if you declare an integer variable and decide to put a character value in it, C enables you to do so. The data might not be in the format you expect, but C does its best. This is much different from stronger-typed languages such as COBOL and Pascal.

If this discussion seems a little over your head at this point, relax. The upcoming chapters will elaborate on these topics and provide many examples.

C++ is a small, block-structured programming language. It has fewer than 46 keywords. To compensate for its small vocabulary, C++ has one of the largest assortment of *operators* such as `+`, `-`, and `&&` (second only to APL). The large number of operators in C++ might tempt programmers to write cryptic programs that have only a small amount of code. As you learn throughout this book, however, you will find that making the program more readable is more important than saving some bytes. This book teaches you how to use the C++ operators to their fullest extent, while maintaining readable programs.

C++'s large number of operators (almost equal to the number of keywords) requires a more judicious use of an *operator precedence*

table. Appendix D, "C++ Precedence Table," includes the C++ operator precedence table. Unlike most other languages that have only four or five levels of precedence, C++ has 15. As you learn C++, you have to master each of these 15 levels. This is not as difficult as it sounds, but its importance cannot be overstated.

C++ also has no input or output statements. You might want to read that sentence again! C++ has no commands that perform input or output. This is one of the most important reasons why C++ is available on so many different computers. The I/O (input/output) statements of most languages tie those languages to specific hardware. BASIC, for instance, has almost twenty I/O commands—some of which write to the screen, to the printer, to a modem, and so forth. If you write a BASIC program for a microcomputer, chances are good that it cannot run on a mainframe without considerable modification.

C++'s input and output are performed through the abundant use of operators and function calls. With every C++ compiler comes a library of standard I/O functions. I/O functions are *hardware independent*, because they work on any device and on any computer that conform to the AT&T C++ standard.

To master C++ completely, you have to be more aware of your computer's hardware than most other languages would require you to be. You certainly do not have to be a hardware expert, but understanding the internal data representation makes C++ much more usable and meaningful.

It also helps if you can become familiar with binary and hexadecimal numbers. You might want to read Appendix A, "Memory Addressing, Binary, and Hexadecimal Review," for a tutorial on these topics before you start to learn the C++ language. If you do not want to learn these topics, you can still become a good C++ programmer, but knowing what goes on "under the hood" makes C++ more meaningful to you as you learn it.

C++ and Microcomputers

C was a relatively unknown language until it was placed on the microcomputer. With the invention and growth of the microcomputer, C blossomed into a worldwide computer language. C++

extends that use on smaller computers. Most of readers of *C++ By Example* are probably working on a microcomputer-based C++ system. If you are new to computers, this section will help you learn how microcomputers were developed.

In the 1970s, NASA created the *microchip*, a tiny wafer of silicon that occupies a space smaller than a postage stamp. Computer components were placed on these microchips, hence computers required much less space than before. NASA produced these smaller computers in response to their need to send rocket ships to the moon with on-board computers. The computers on Earth could not provide split-second accuracy for rockets because radio waves took several seconds to travel between the Earth and the moon. Through development, these microchips became small enough so the computers could travel with a rocket and safely compute the rocket's trajectory.

The space program was not the only beneficiary of computer miniaturization. Because microchips became the heart of the *micro*computer, computers could now fit on desktops. These microcomputers cost much less than their larger counterparts, so many people started buying them. Thus, the home and small-business computer market was born.

Today, microcomputers are typically called *PCs* from the widespread use of the original IBM PC. The early PCs did not have the memory capacity of the large computers used by government and big business. Nevertheless, PC owners still needed a way to program these machines. BASIC was the first programming language used on PCs. Over the years, many other languages were ported from larger computers to the PC. However, no language was as successful as C in becoming the worldwide standard programming language. C++ seems to be the next standard.

Before diving into C++, you might take a few moments to familiarize yourself with some of the hardware and software components of your PC. The next section, "An Overview of Your Computer," introduces you to computer components that C++ interacts with, such as the operating system, memory, disks, and I/O devices. If you are already familiar with your computer's hardware and software, you might want to skip to Chapter 2, "What Is a Program?," and begin using C++.

An Overview of Your Computer

Your computer system consists of two parts: *hardware* and *software*. The hardware consists of all the physical parts of the machine. Hardware has been defined as "anything you can kick." Although this definition is coarse, it illustrates that your computer's hardware consists of the physical components of your PC. The software is everything else. Software comprises the programs and data that interact with your hardware. The C++ language is an example of software. You can use C++ to create even more software programs and data.

Hardware

Figure 1.1 shows you a typical PC system. Before using C++, you should have a general understanding of what hardware is and how your hardware components work together.

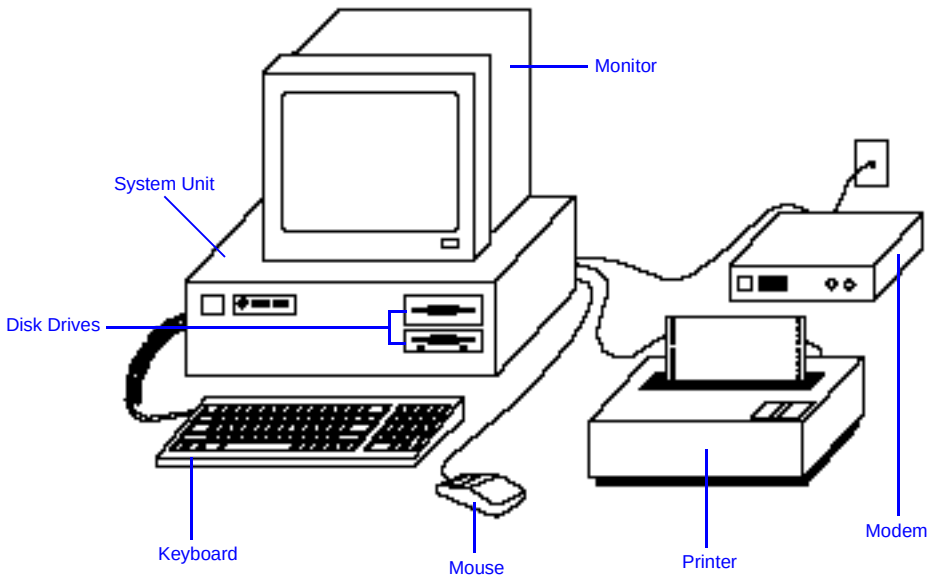


Figure 1.1. A typical PC system.

The System Unit and Memory

The *system unit* is the large, box-shaped component of the computer. This unit houses the PC's microprocessor. You might hear the microprocessor called the *CPU*, or *central processing unit*. The CPU acts like a traffic cop, directing the flow of information throughout your computer system. The CPU is analogous also to the human brain. When you use a computer, you are actually interacting with its CPU. All the other hardware exists so the CPU can send information to you (through the monitor or the printer), and you can give instructions to the CPU (through the keyboard or the mouse).

The CPU also houses the computer's internal *memory*. Although the memory has several names, it is commonly referred to as *RAM* (random-access memory). RAM is where the CPU looks for software and data. When you run a C++ program, for example, you are instructing your computer's CPU to look in RAM for that program and carry out its instructions. C++ uses RAM space when it is loaded.

A byte is a single character of memory.

RAM is used for many things and is one of the most important components of your computer's hardware. Without RAM, your computer would have no place for its instructions and data. The amount of RAM can also affect the computer's speed. In general, the more RAM your computer has, the more work it can do and the faster it can process data.

The amount of RAM is measured by the number of characters it can hold. PCs generally hold approximately 640,000 characters of RAM. A character in computer terminology is called a *byte*, and a byte can be a letter, a number, or a special character such as an exclamation point or a question mark. If your computer has 640,000 bytes of RAM, it can hold a total of 640,000 characters.

All the zeros following RAM measurements can become cumbersome. You often see the shortcut notation *K* (which comes from the metric system's *kilo*, meaning *1000*) in place of the last three zeros. In computer terms, K means exactly 1024 bytes; but this number is usually rounded to 1000 to make it easier to remember. Therefore, 640K represents approximately 640,000 bytes of RAM. For more information, see the sidebar titled "The Power of Two."

The limitations of RAM are similar to the limitations of audio cassette tapes. If a cassette is manufactured to hold 60 minutes of

music, it cannot hold 75 minutes of music. Likewise, the total number of characters that compose your program, the C++ data, and your computer's system programs cannot exceed the RAM's limit (unless you save some of the characters to disk).

You want as much RAM as possible to hold C++, data, and the system programs. Generally, 640K is ample room for anything you might want to do in C++. Computer RAM is relatively inexpensive, so if your computer has less than 640K bytes of memory, you should consider purchasing additional memory to increase the total RAM to 640K. You can put more than 640K in most PCs. There are two types of additional RAM: *extended* memory and *expanded* memory (they both offer memory capacity greater than 640K). You can access this extra RAM with some C++ systems, but most beginning C++ programmers have no need to worry about RAM beyond 640K.

The Power of Two

Although K means approximately 1000 bytes of memory, K equates to 1024. Computers function using *on* and *off* states of electricity. These are called *binary* states. At the computer's lowest level, it does nothing more than turn electricity on and off with many millions of switches called *transistors*. Because these switches have two possibilities, the total number of states of these switches—and thus the total number of states of electricity—equals a number that is a power of 2.

The closest power of 2 to 1000 is 1024 (2 to the 10th power). The inventors of computers designed memory so that it is always added in kilobytes, or multiples of 1024 bytes at a time. Therefore, if you add 128K of RAM to a computer, you are actually adding a total of 131,072 bytes of RAM (128 times 1024 equals 131,072).

Because K actually means *more* than 1000, you always have a little more memory than you bargained for! Even though your computer might be rated at 640K, it actually holds more than 640,000 bytes (655,360 to be exact). See Appendix A, "Memory Addressing, Binary, and Hexadecimal Review," for a more detailed discussion of memory.

The computer stores C++ programs to RAM as you write them. If you have used a word processor before, you have used RAM. As you type words in your word-processed documents, your words appear on the video screen and also go to RAM for storage.

Despite its importance, RAM is only one type of memory in your computer. RAM is *volatile;* when you turn the computer off, all RAM is erased. Therefore, you must store the contents of RAM to a nonvolatile, more permanent memory device (such as a disk) before you turn off your computer. Otherwise, you lose your work.

Disk Storage

A *disk* is another type of computer memory, sometimes called *external memory*. Disk storage is nonvolatile. When you turn off your computer, the disk's contents do not go away. This is important. After typing a long C++ program in RAM, you do not want to retype the same program every time you turn your computer back on. Therefore, after creating a C++ program, you save the program to disk, where it remains until you're ready to retrieve it again.

Disk storage differs from RAM in ways other than volatility. Disk storage cannot be processed by the CPU. If you have a program or data on disk that you want to use, you must transfer it from the disk to RAM. This is the only way the CPU can work with the program or data. Luckily, most disks hold many times more data than the RAM's 640K. Therefore, if you fill up RAM, you can store its contents on disk and continue working. As RAM continues to fill up, you or your C++ program can keep storing the contents of RAM to the disk.

This process might sound complicated, but you have only to understand that data must be transferred to RAM before your computer can process it, and saved to disk before you shut your computer off. Most the time, a C++ program runs in RAM and retrieves data from the disk as it needs it. In Chapter 30, "Sequential Files," you learn that working with disk files is not difficult.

There are two types of disks: *hard disks* and *floppy disks.* Hard disks (sometimes called *fixed disks*) hold much more data and are many times faster to work with than floppy disks. Most of your C++ programs and data should be stored on your hard disk. Floppy disks

are good for backing up hard disks, and for transferring data and programs from one computer to another. (These removable floppy disks are often called *diskettes*.) Figure 1.2 shows two common sizes, the 5 1/4-inch disk and the 3 1/2-inch disk. These disks can hold from 360K to 1.4 million bytes of data.

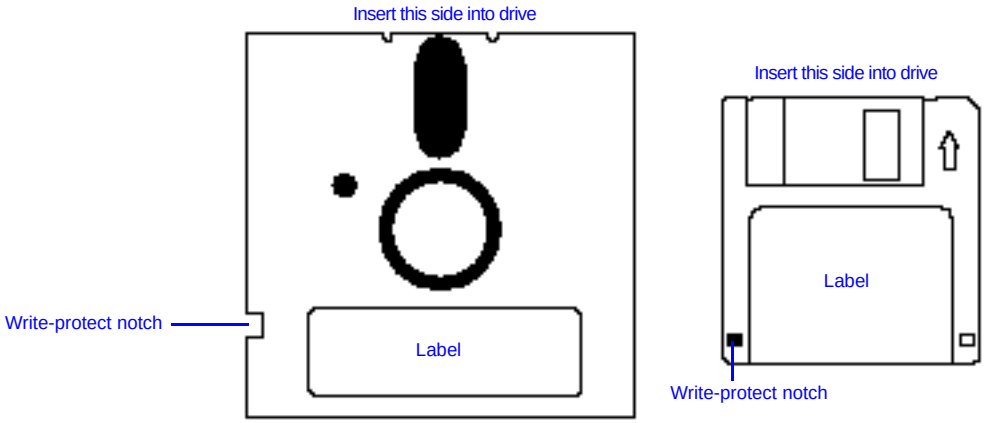


Figure 1.2. 5 1/4-inch disk and 3 1/2-inch disk.

Before using a new box of disks, you have to format them (unless you buy disks that are already formatted). Formatting prepares the disks for use on your computer by writing a pattern of paths, called *tracks*, where your data and programs are stored. Refer to the operating system instruction manual for the correct formatting procedure.

Disk drives house the disks in your computer. Usually, the disk drives are stored in your system unit. The hard disk is sealed inside the hard disk drive, and you never remove it (except for repairs). In general, the floppy disk drives also are contained in the system unit, but you insert and remove these disks manually.

Disk drives have names. The computer's first floppy disk drive is called drive A. The second floppy disk drive, if you have one, is called drive B. The first hard disk (many computers have only one) is called drive C. If you have more than one hard disk, or if your hard disk is logically divided into more than one, the others are named drive D, drive E, and so on.

Disk size is measured in bytes, just as RAM is. Disks can hold many millions of bytes of data. A 60-million-byte hard disk is common. In computer terminology, a million bytes is called a *megabyte*, or *M*. Therefore, if you have a 60-megabyte hard disk, it can hold approximately 60 million characters of data before it runs out of space.

The Monitor

The television-like screen is called the *monitor*. Sometimes the monitor is called the *CRT* (which stands for the primary component of the monitor, the *cathode-ray tube*). The monitor is one place where the output of the computer can be sent. When you want to look at a list of names and addresses, you could write a C++ program to list the information on the monitor.

The advantage of screen output over printing is that screen output is faster and does not waste paper. Screen output, however, is not permanent. When text is *scrolled* off-screen (displaced by additional text coming on-screen), it is gone and you might not always be able to see it again.

All monitors have a *cursor*, which is a character such as a blinking underline or a rectangle. The cursor moves when you type letters on-screen, and always indicates the location of the next character to be typed.

Monitors that can display pictures are called *graphics monitors*. Most PC monitors are capable of displaying graphics and text, but some can display only text. If your monitor cannot display colors, it is called a *monochrome* monitor.

Your monitor plugs into a *display adapter* located in your system unit. The display adapter determines the amount of resolution and number of possible on-screen colors. *Resolution* refers to the number of row and column intersections. The higher the resolution, the more rows and columns are present on your screen and the sharper your text and graphics appear. Some common display adapters are MCGA, CGA, EGA, and VGA.

The Printer

The printer provides a more permanent way of recording your computer's results. It is the "typewriter" of the computer. Your printer can print C++ program output to paper. Generally, you can print anything that appears on your screen. You can use your printer to print checks and envelopes too, because most types of paper work with computer printers.

The two most common PC printers are the *dot-matrix* printer and the *laser* printer. A dot-matrix printer is inexpensive, fast, and uses a series of small dots to represent printed text and graphics. A laser printer is faster than a dot-matrix, and its output is much sharper because a laser beam burns toner ink into the paper. For many people, a dot-matrix printer provides all the speed and quality they need for most applications. C++ can send output to either type of printer.

The Keyboard

Figure 1.3 shows a typical PC keyboard. Most the keys are the same as those on a standard typewriter. The letter and number keys in the center of the keyboard produce their indicated characters on-screen. If you want to type an uppercase letter, be sure to press one of the Shift keys before typing the letter. Pressing the CapsLock key shifts the keyboard to an uppercase mode. If you want to type one of the special characters above a number, however, you must do so with the Shift key. For instance, to type the percent sign (%), you would press Shift-5.

Like the Shift keys, the Alt and Ctrl keys can be used with some other keys. Some C++ programs require that you press Alt or Ctrl before pressing another key. For instance, if your C++ program prompts you to press Alt-F, you should press the Alt key, then press F while still holding down Alt, then release both keys. Do not hold them both down for long, however, or the computer keeps repeating your keystrokes as if you typed them more than once.

The key marked Esc is called the *escape* key. In many C++ programs, you can press Esc to "escape," or exit from, something you started and then wanted to stop. For example, if you prompt your C++ compiler for help and you no longer need the help

message, you can press Esc to remove the help message from the screen.

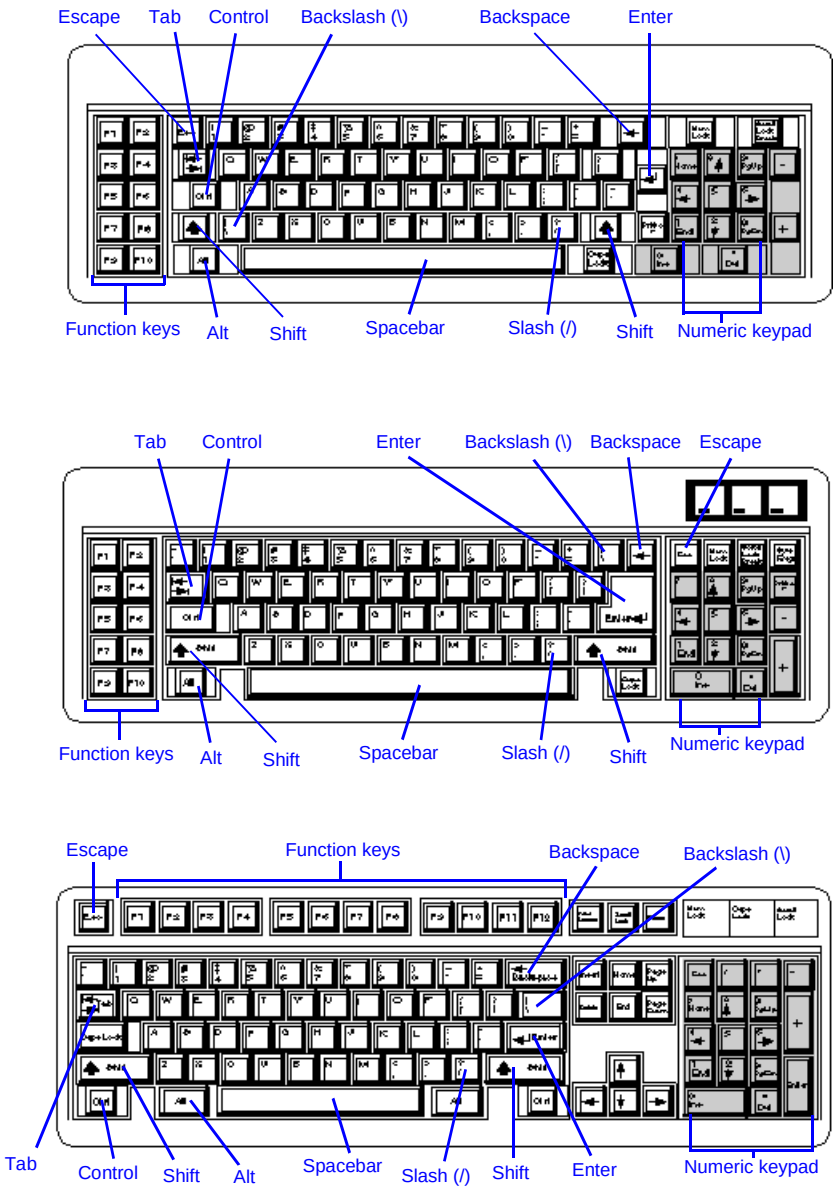


Figure 1.3. The various PC keyboards.

The group of numbers and arrows on the far right of the keyboard is called the *numeric keypad*. People familiar with a 10-key adding machine usually prefer to type numbers from the keypad rather than from the top row of the alphabetic key section. The numbers on the keypad work only when you press the NumLock key. If you press NumLock a second time, you disable these number keys and make the arrow keys work again. To prevent confusion, many keyboards have separate arrow keys and a keypad used solely for numbers.

The arrows help you move the cursor from one area of the screen to another. To move the cursor toward the top of the screen, you have to press the up arrow continuously. To move the cursor to the right, you press the right-arrow, and so on. Do not confuse the Backspace key with the left-arrow. Pressing Backspace moves the cursor backward one character at a time—erasing everything as it moves. The left-arrow simply moves the cursor backward, without erasing.

The keys marked Insert and Delete (Ins and Del on some keyboards) are useful for editing. Your C++ program editor probably takes advantage of these two keys. Insert and Delete work on C++ programs in the same way they work on a word processor's text. If you do not have separate keys labeled Insert and Delete, you probably have to press NumLock and use the keypad key 0 (for Insert) and period (for Delete).

PgUp and PgDn are the keys to press when you want to scroll the screen (that is, move your on-screen text either up or down). Your screen acts like a camera that pans up and down your C++ programs. You can move the screen down your text by pressing PgDn, and up by pressing PgUp. (Like Insert and Delete, you might have to use the keypad for these operations.)

The keys labeled F1 through F12 (some keyboards go only to F10) are called *function keys*. The function keys are located either across the top of the alphabetic section or to the left of it. These keys perform an advanced function, and when you press one of them, you usually want to issue a complex command, such as searching for a specific word in a program. The function keys in your C++ program, however, do not necessarily produce the same results as they might in another program, such as a word processor. In other words, function keys are *application-specific*.

CAUTION: Computer keyboards have a key for number 1, so do not substitute the lowercase *l* to represent the number 1, as you might on a typewriter. To C++, a *1* is different from the letter *l*. You should be careful also to use *0* when you mean zero, and *O* when you want the uppercase letter *O*.

The Mouse

The mouse is a relatively new input device. The mouse moves the cursor to any on-screen location. If you have never used a mouse before, you should take a little time to become skillful in moving the cursor with it. Your C++ editor (described in Chapter 2, "What is a Program?") might use the mouse for selecting commands from its menus.

Mouse devices have two or three buttons. Most of the time, pressing the third button produces the same results as simultaneously pressing both keys on a two-button mouse.

The Modem

A modem can be used to communicate between two distant computers.

A PC *modem* enables your PC to communicate with other computers over telephone lines. Some modems, called *external modems,* sit in a box outside your computer. *Internal modems* reside inside the system unit. It does not matter which one you have, because they operate identically.

Some people have modems so they can share data between their computer and that of a long-distance friend or off-site coworker. You can write programs in C++ that communicate with your modem.

A Modem by Any Other Name...

The term *digital computer* comes from the fact that your computer operates on binary (on and off) digital impulses of electricity. These digital states of electricity are perfect for your computer's equipment, but they cannot be sent over normal telephone lines. Telephone signals are called *analog* signals, which are much different from the binary digital signals in your PC.

Therefore, before your computer can transmit data over a telephone line, the information must be *modulated* (converted) to analog signals. The receiving computer must have a way to *demodulate* (convert back) those signals to digital.

The modem is the means by which computer signals are modulated and demodulated from digital to analog and vice versa. Thus, the name of the device that *mo*dulates and *dem*odulates these signals is *modem*.

Software

No matter how fast, large, and powerful your computer's hardware is, its software determines what work is done and how the computer does it. Software is to a computer what music is to a stereo system. You store software on the computer's disk and load it in your computer's memory when you are ready to process the software, just as you store music on a tape and play it when you want to hear music.

Programs and Data

No doubt you have heard the phrase, *data processing.* This is what computers actually do: they take data and manipulate it into

meaningful output. The meaningful output is called *information.* Figure 1.4 shows the *input-process-output* model, which is the foundation of everything that happens in your computer.

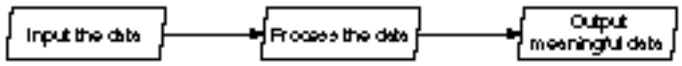


Figure 1.4. Data processing at its most elementary level.

In Chapter 2, "What Is a Program?," you learn the mechanics of programs. For now, you should know that the programs you write in C++ process the data that you input in the programs. Both data and programs compose the software. The hardware acts as a vehicle to gather the input and produce the output. Without software, computers would be worthless, just as an expensive stereo would be useless without some way of playing music so you can hear it.

The input comes from input devices, such as keyboards, modems, and disk drives. The CPU processes the input and sends the results to the output devices, such as the printer and the monitor. A C++ payroll program might receive its input (the hours worked) from the keyboard. It would instruct the CPU to calculate the payroll amounts for each employee in the disk files. After processing the payroll, the program could print the checks.

MS-DOS

MS-DOS (Microsoft disk operating system) is a system that lets your C++ programs interact with hardware. MS-DOS (commonly called DOS) is always loaded into RAM when you turn on your computer. DOS controls more than just the disks; DOS is there so your programs can communicate with all the computer's hardware, including the monitor, keyboard, and printer.

Figure 1.5 illustrates the concept of DOS as the "go-between" with your computer's hardware and software. Because DOS understands how to control every device hooked to your computer, it stays in RAM and waits for a hardware request. For instance, printing the words `"C++ is fun!"` on your printer takes many computer instructions. However, you do not have to worry about all

those instructions. When your C++ program wants to print something, it tells DOS to print it. DOS always knows how to send information to your printer, so it takes your C++ program requests and does the work of routing that data to the printer.

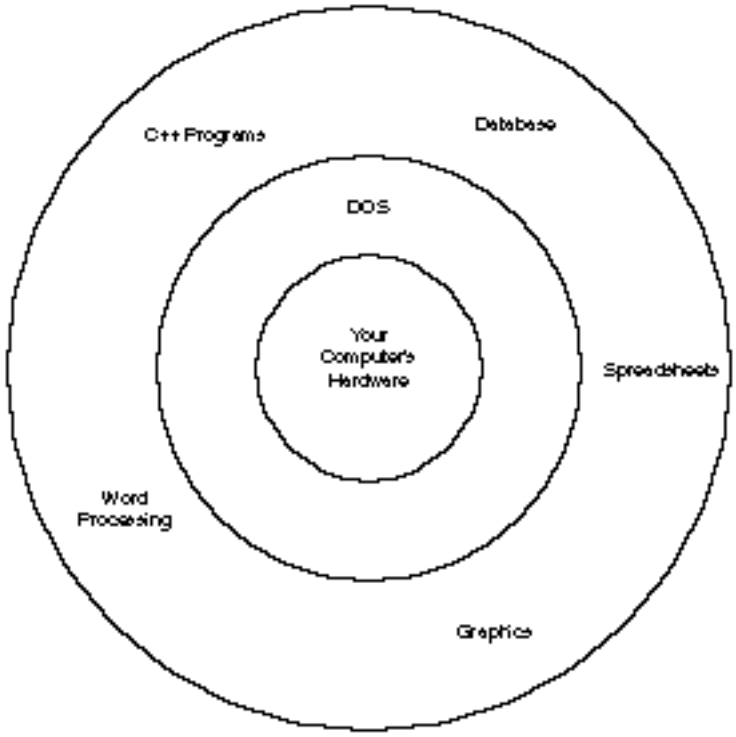


Figure 1.5. DOS interfaces between hardware and software.

Many people program computers for years and never take the time to learn why DOS is there. You do not have to be an expert in DOS, or even know more than a few simple DOS commands, to be proficient with your PC. Nevertheless, DOS does some things that C++ cannot do, such as formatting disks and copying files to your disks. As you learn more about the computer, you might see the need to better understand DOS. For a good introduction to using DOS, refer to the book *MS-DOS 5 QuickStart* (Que).

NOTE: As mentioned, DOS always resides in RAM and is loaded when you start the computer. This is done automatically, so you can use your computer and program in C++ without worrying about how to transfer DOS to RAM. It is important to remember that DOS always uses some of your total RAM.

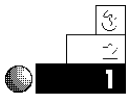
Figure 1.6 shows you the placement of DOS, C++, and your C++ data area in RAM. This formation is a typical way to represent RAM—several boxes stacked on top of each other. Each memory location (each byte) has a unique *address*, just as everybody's residence has a unique address. The first address in memory begins at 0, the second RAM address is 1, and so on until the last RAM location, many thousands of bytes later.

Figure 1.6. After MS-DOS and a C++ program, there is less RAM for data.

Your operating system (whether you use MS-DOS, PC DOS, DR DOS, or UNIX) takes part of the first few thousand bytes of memory. The amount of RAM that DOS takes varies with each computer's configuration. When working in C++, the C++ system sits on top of DOS, leaving you with the remainder of RAM for your program and data. This explains why you might have a total of 512K of RAM and still not have enough memory to run some programs—DOS is using some of the RAM for itself.

Review Questions

The answers to each chapter's review questions are in Appendix B, aptly named "Answers to Review Questions."



1. What is the name of one of the programming languages from which C was developed?
2. True or false: C++ is known as a "better C."
3. In what decade was C++ developed?
4. True or false: C++ is too large to fit on many microcomputers.
5. Which usually holds more data: RAM or the hard disk?
6. What device is needed for your PC to communicate over telephone lines?

7. Which of the following device types best describes the mouse?
   a. Storage
   b. Input
   c. Output
   d. Processing
8. What key would you press to turn off the numbers on the numeric keypad?

9. What operating system is written almost entirely in C?
10. Why is RAM considered volatile?
11. True or false: The greater the resolution, the better the appearance of graphics on-screen.
12. How many bytes is 512K?
13. What does *modem* stand for?

Summary

C++ is an efficient, powerful, and popular programming language. Whether you are new to C++ or an experienced programmer, C++ is all you need to program the computer to work the way you want it to.

This chapter presented the background of C++ by walking you through the history of its predecessor, the C programming language. C++ adds to C and offers some of the most advanced programming language commands that exist today.

The rest of this book is devoted to teaching you C++. Chapter 2, "What Is a Program?," explains program concepts so you can begin to write C++ programs.