Introduction to Computers, Problem Solving, and Programming

Chapter Objectives

- To learn the different categories of computers
- To understand the role of each component in a computer
- To understand the purpose of an operating system
- To learn the differences between machine language, assembly language, and higher level languages
- To understand what processes are required to run a C++ program
- To learn how to solve a programming problem in a careful, disciplined way
- To understand and appreciate ethical issues related to the use of computers and programming

SINCE THE 1940s—a period of little more than 70 years—the development of the computer has spurred the growth of technology into realms only dreamed of at the turn of the twentieth century. Computers have changed the way we live and how we do business. Many of us use computers to register for courses, to send and receive electronic mail (e-mail), to shop and bank from home, to retrieve information from the World Wide Web, to research and write term papers, and to do other homework assignments. Computers are a key component of automatic teller machines (ATMs), and computers are built into our cars and most household appliances. Computers can receive, store, process, and output information of all kinds: numbers, text, images, graphics, video, and sound, to name a few.

Although we're often led to believe otherwise, computers cannot "think." Basically, computers are devices for performing computations at incredible speeds (more than a billion instructions per

second) and with great accuracy. But, to accomplish anything useful, a computer must be provided with a list of instructions, or a program. Programs are usually written in special computer programming languages—such as C++, the subject of this book and one of the most versatile programming languages available today.

This chapter introduces the computer and its components and then presents an overview of programming languages. We describe a systematic approach to solving problems called the software development method, and we use it to write a basic C++ program. The chapter ends with a discussion of ethics for programmers.

1.1 Overview of Computers

Most of us deal with computers every day, and we probably use computers for word processing or for surfing the World Wide Web. And some of us may even have studied programming in high school. But it wasn't always this way. Not long ago, most people considered computers to be mysterious devices whose secrets were known only by a few computer wizards.

Early Computers

If we take the literal definition for a computer as "a device for counting or computing," then we could consider the abacus to have been the first computer. The first electronic digital computer was designed in the late 1930s by Dr. John Atanasoff and graduate student Clifford Berry at Iowa State University. They designed their computer to help them perform mathematical computations in nuclear physics.

The first large-scale, general-purpose electronic digital computer, called the ENIAC, was completed in 1946 at the University of Pennsylvania with funding from the U.S. Army. The ENIAC weighed 30 tons and occupied a 30-by-50-foot space. It was used to compute ballistics tables, predict the weather, and make atomic energy calculations. Its designers were J. Presper Eckert and John Mauchley.

stored-program computer (von Neumann architecture)
A computer design based on the concept of storing a computer program along with its data in computer memory.

To program the ENIAC, engineers had to connect hundreds of wires and arrange thousands of switches in a certain way. In 1946, Dr. John von Neumann of Princeton University proposed the concept of a **stored-program** computer—a computer whose program was stored in computer memory. Von Neumann knew that the data stored in computer memory could easily be changed by a program. He reasoned that programs, too, could be stored in computer memory and changed far more easily than by connecting wires and

setting switches. Von Neumann designed a computer based on this idea. His design was a success and greatly simplified computer programming. The **von Neumann architecture** is the basis of the digital computer as we know it today.

Categories of Computers

Early computers used vacuum tubes as their basic electronic component. Technological advances in the design and manufacture of these components led to new generations of computers that were considerably smaller, faster, and less expensive than their predecessors.

In the 1970s, the Altair and Apple computer companies manufactured the first **microcomputers**. The computer processor in a microcomputer is an electronic component called a **microprocessor**, which is about the size of a postage stamp. Because they are so small and relatively inexpensive, microprocessors are found in watches, pocket calculators, cameras, home appliances, and automobiles, as well as in computers.

microcomputer
A computer that uses a very small processor.
microprocessor
The processor found in a microcomputer.
minicomputer
A computer for businesses or research laboratories that can be used by many people simultaneously.
mainframe
A computer with more computational power than a minicomputer that is often used by major corporations.

Most offices have one or more personal computers. Typical models cost less than $2000 and sit on a desk, yet have as much computational power as the giants of 20 years ago that cost more than $100,000 and filled a 9-by-12-foot room. Today's computers come in even smaller models that can fit inside a backpack or a person's hand (see Figure 1.1).

Personal computers are used by one person at a time. Businesses and research labs use larger and faster computers called **minicomputers** and **mainframes**, which can be used by many people simultaneously.

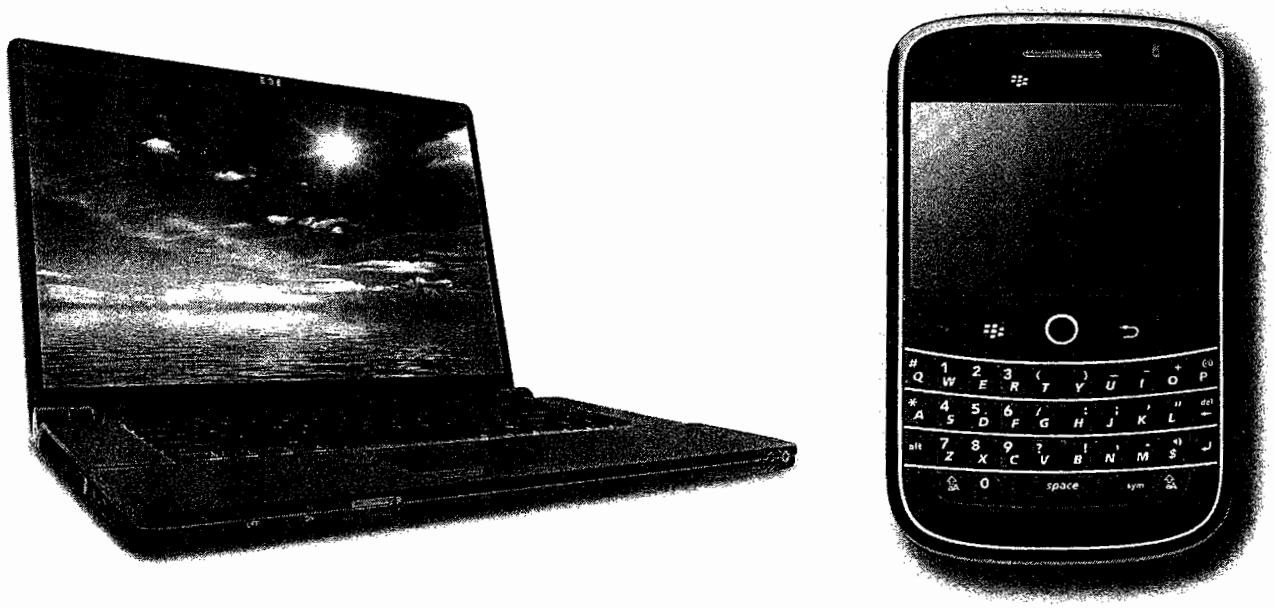


Figure 1.1 Netbook computer and hand-held computer

supercomputer
The most powerful kind of mainframe computer, performing in seconds computations that might take hours or days on other computers.

Supercomputers, the most powerful mainframe computers, can perform in seconds computations that might take hours or even days on other computers.

Sharing Computer Resources

time sharing
A process that allows simultaneous access to a single computer by a number of users.

Time sharing is used on mainframes and minicomputers to allow simultaneous access to the computing resources. The problem with time sharing is that users have to wait for their turn to access the resources. In the early days, these waits could take minutes. And if the computer stops working, all users are affected, as they must wait for the computer to be restarted.

Although microcomputers don't have the huge resources of minicomputers and mainframes, they provide their users with dedicated resources. Also if one microcomputer stops working, others are not affected. The major disadvantage of early personal or workstation computers was that they were isolated from the vast resources of the larger machines. In Section 1.2, we see how computer networks solve this problem.

EXERCISES FOR SECTION 1.1

Self-Check

1. List the different kinds of computers from smallest to largest.
2. Why do you think each computer user in a time-shared environment is unaware that others are also using the computer?
3. Describe the contributions of Atanasoff and Berry, Eckert and Mauchley, and von Neumann.

1.2 Computer Hardware

hardware
The actual computer equipment.
software
The set of programs associated with a computer.
program
A list of instructions that a computer uses to manipulate data to perform a task.

A computer system consists of two major components: **hardware**—the actual equipment used to perform the computations—and **software**—that is, the programs. **Programs** let us communicate with a computer by giving it the instructions it needs to operate. We discuss hardware in this section and software in the next.

Despite their differences in cost, size, and capabilities, modern computers resemble each other in many basic ways. Essentially, most consist of the following hardware components:

- Main memory
- Secondary memory, including storage media such as hard disks, flash drives, and CD-ROMs

- Central processing unit (CPU)
- Input devices, such as a keyboard and mouse
- Output devices, such as a monitor and printer
- Network connection, such as a modem or Ethernet interface

Figure 1.2 shows how these components interact in a computer when a program is executed; the arrows show the direction of information flow.

The program must be transferred from *secondary memory* (or *secondary storage*) to *main memory* before it can be executed. Data must be supplied from some source. The person using a program (the *program user*) may supply data through an *input device* such as a mouse or a keyboard, from a *data file* located in secondary storage, or from a remote machine via the network connection. The data are stored in the computer's *main memory* where they can be accessed and manipulated by the *central processing unit*. The results of this manipulation are stored back in *main memory*. Finally, the information (results) in main memory may be displayed through an *output device* such as a monitor or printer, stored in secondary storage, or sent to another computer via the network. In the remainder of this section, we describe these components in more detail.

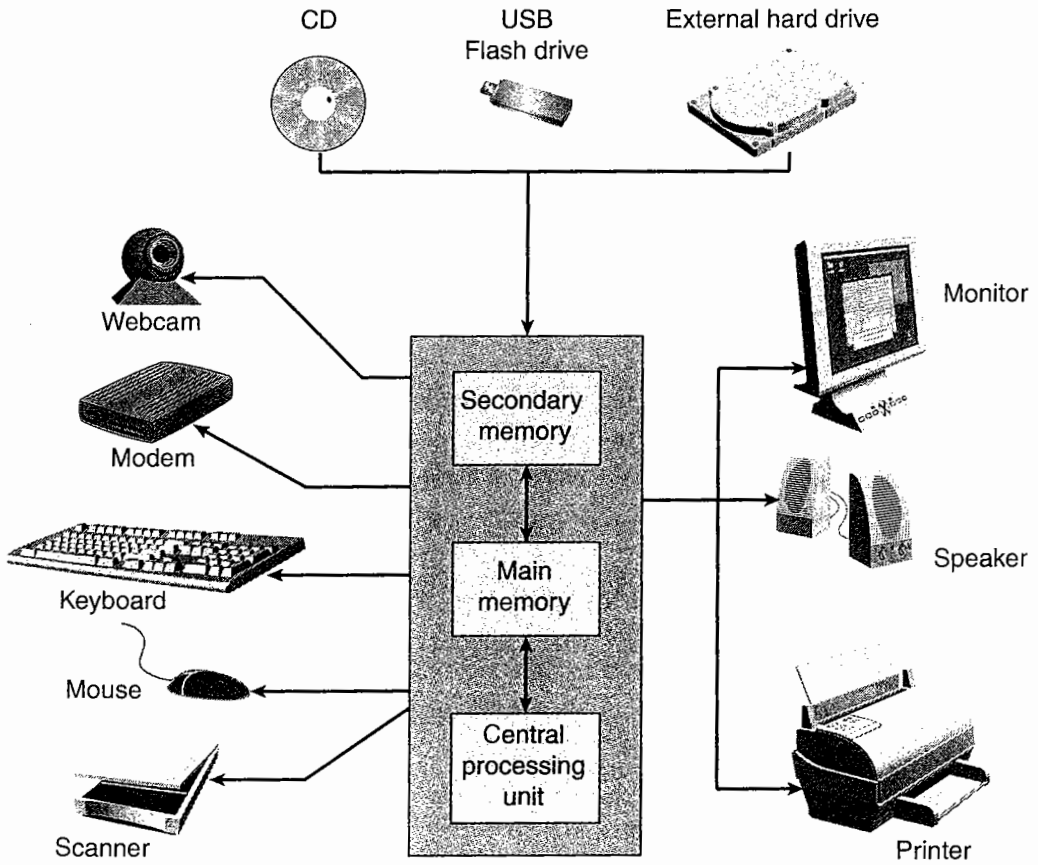


Figure 1.2 Computer components

Memory

Memory is an essential component in any computer. Before discussing the types of memory—main and secondary—let's look at what it consists of and how the computer works with it.

Anatomy of Memory

memory cell
an individual storage location in memory.

address of a memory cell
the relative position of a memory cell in the computer's main memory.

contents of a memory cell
the information stored in a memory cell, either a program instruction or data.

Imagine the memory of a computer as a sequence of storage locations called **memory cells**. To store and access information, the computer must have some way of identifying the individual memory cells, so each memory cell has a unique **address** that indicates its position in memory. Figure 1.3 shows a computer memory consisting of 1000 memory cells with addresses 0 through 999. Most computers have millions of individual memory cells, each with its own address.

The data stored in a memory cell are called the **contents** of the cell. In Figure 1.3, the contents of memory cell 3 is the number −26 and the contents of memory cell 4 is the letter H.

Memory cells can also contain program instructions. Cells 6 through 8 in Figure 1.3 store instructions to add two numbers (from cells 1 and 3) and store the result in memory cell 5. You'll recall that in a von Neumann computer, a program's instructions must be stored in main memory before they

Address	Contents
0	-27.2
1	354
2	0.005
3	-26
4	H
5	400
6	RTV 001
7	ADD 003
8	STO 005
⋮	⋮
998	X
999	75.62

Figure 1.3 A thousand memory cells in main memory

can be executed. Whenever we open a new program, we change the computer's operation by storing the new instructions in memory.

Bytes and Bits

A memory cell is actually a grouping of smaller units called bytes. A **byte** is the amount of storage required to store a single character, such as the letter H in cell 4 of Figure 1.3. The number of bytes a memory cell can contain varies from computer to computer.

byte
The amount of storage required to store a single character.

A byte is composed of even smaller units of storage called bits (see Figure 1.4). There are eight bits to a byte. "**Bit**" derives from the words *bi*nary digi*t* and is the smallest element a computer can deal with. Binary refers to a number system based on two numbers, 0 and 1; therefore, a bit is either a 0 or a 1.

bit (binary digit)
A binary unit representing a 0 or a 1.

Storing and Retrieving Information in Memory

A particular pattern of zeros and ones (that is, bits) represents each value in memory, whether it's a number, a letter, or an instruction such as ADD 003. A computer can either store or retrieve a value. When **storing** a value, the computer sends electronic signals to set each bit of a selected memory cell to either 0 or 1; storing a value destroys the previous contents of the cell. When **retrieving** a value from a memory cell, the computer copies the pattern of 0s and 1s stored in that cell to another storage area for processing; copying does *not* destroy the contents of the cell whose value is retrieved. This process is the same regardless of the kind of information—character, number, or program instruction—to be stored or retrieved.

storing a value in memory
Setting the individual bits of a memory cell to 0 or 1, destroying its previous contents.

retrieving a value from memory
Copying the contents of a particular memory cell to another storage area for processing.

Main Memory

Main memory, made of electronic circuitry on small computer chips, stores programs, data, and results. Most computers have two types of main memory: **random access memory (RAM)** for temporary storage of programs, data, and results, and **read-only memory (ROM)**, which stores programs or data permanently.

RAM temporarily stores programs while they are being executed by the computer. It also temporarily stores information—for example, numbers,

random access memory (RAM)
The part of main memory that temporarily stores programs, data, and results.

read-only memory (ROM)
The part of main memory that permanently stores programs or data.

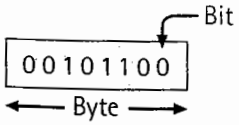


Figure 1.4 Relationship between a byte and a bit

names, or pictures—that is being processed by the computer. RAM is usually **volatile memory**, which means that when you switch off the computer, you will lose everything stored in RAM. To prevent this, you should store the contents of RAM in semipermanent secondary memory (discussed below) before switching off your computer.

volatile memory
Memory whose contents disappear when you switch off the computer.

In contrast, ROM stores information permanently. The computer can retrieve (or read) information but it can't store (or write) information in ROM—hence its name, read-only memory. Because they're not volatile, the instructions and data stored in ROM don't disappear when you switch off the computer. Most modern computers contain an internal ROM that stores the instructions needed to get the computer running when you first switch it on.

Usually a computer contains much more RAM than internal ROM. The amount of RAM can often be increased (up to a specified maximum), but the amount of internal ROM is usually fixed. When we refer to main memory in this text, we mean RAM because that is the part of main memory that is normally accessible to the programmer.

RAM is relatively fast memory, but is limited in size and is not permanent memory. These features are in contrast to secondary memory, which, although slower than RAM, is larger and more permanent.

Secondary Memory

Computer systems provide storage in addition to main memory for two reasons. First, computers need storage that is permanent or semipermanent so that information can be retained during a power loss or when the computer is turned off. Second, systems typically store more information than will fit in memory.

secondary storage
Units such as disks or flash drives that retain data even when the power to the drive is off.

disk
Thin platter of metal or plastic on which data are represented by magnetized spots arranged in tracks.

Figure 1.5 shows some of the most frequently used **secondary storage** devices and storage media. Most personal computers use two types of disk drives as their secondary storage devices—hard drives and optical drives. Hard **disks** are attached to their disk drives and are coated with a magnetic material. Each data bit is a magnetized spot on the disk, and the spots are arranged in concentric circles called tracks. The disk drive read/write head accesses data by moving across the spinning disk to the correct track and

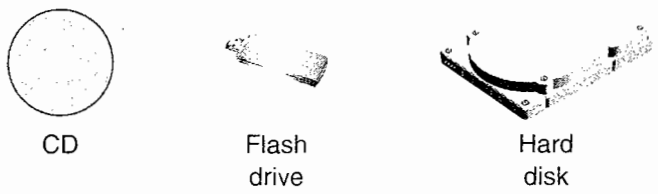


Figure 1.5 Secondary storage media

then sensing the spots as they move by. The hard disks in personal computers usually hold from one to several hundred gigabytes (GB) of data, but clusters of hard drives that store data for an entire network may provide many terabytes (TB) of storage (see Table 1.1).

Most of today's personal computers are equipped with **optical drives** for storing and retrieving data on compact disks (CDs) or digital versatile disks (DVDs) that can be removed from the drive. A CD is a silvery plastic platter on which a laser records data as a sequence of tiny pits in a spiral track on one side of the disk. One CD can hold 680 MB of data. A DVD uses smaller pits packed in a tighter spiral, allowing storage of 4.7 GB of data on one layer. Some DVDs can hold four layers of data—two on each side—for a total capacity of 17 GB, sufficient storage for as much as nine hours of studio-quality video and multichannel audio.

optical drive
Device that uses a laser to access or store data on a CD or DVD.

Flash drives such as the one pictured in Figure 1.5 use flash memory packaged in small plastic cases about three inches long that can be plugged into any of a computer's USB (Universal Serial Bus) ports. Unlike hard drives and optical drives that must spin their disks for access to data, flash drives have no moving parts and all data transfer is by electronic signal only. In flash memory, bits are represented as electrons trapped in microscopic chambers of silicon dioxide. Typical USB flash drives store 1 to a few GB of data, but 64-GB drives are also available.

flash drive
Device that plugs into USB port and stores data bits as trapped electrons.

Information stored on a disk is organized into separate collections called **files**. One file may contain a C++ program. Another file may contain the data to be processed by that program (a *data file*). A third file could contain the results generated by a program (an *output file*). The names of all files stored on a disk are listed in the disk's **directory**. This directory may be broken into one or more levels of subdirectories or folders, where each **subdirectory** stores the names of files that relate to the same general topic. For example, you might have separate subdirectories of files that contain homework assignments and programs for each course you are taking this

file
Named collection of data stored on a disk.

directory
A list of the names of files stored on a disk.

subdirectory
A list of the names of files that relate to a particular topic.

Table 1.1 Terms Used to Quantify Storage Capacities

Term	Abbreviation	Equivalent to	Comparison to Power of 10
Byte	B	8 bits	
Kilobyte	KB	1,024 (2^{10}) bytes	$> 10^3$
Megabyte	MB	1,048,576 (2^{20}) bytes	$> 10^6$
Gigabyte	GB	1,073,741,824 (2^{30}) bytes	$> 10^9$
Terabyte	TB	1,099,511,627,776 (2^{40}) bytes	$> 10^{12}$

semester. The details of how files are named and grouped in directories vary with each computer system. Follow the naming conventions that apply to your system.

Central Processing Unit

central processing unit (CPU)
Coordinates all computer operations and performs arithmetic and logical operations on data.

The **central processing unit (CPU)** has two roles: coordinating all computer operations and performing arithmetic and logical operations on data. The CPU follows instructions in computer programs to determine which operations to carry out and in what order. It then transmits control signals to the other computer components. For example, when instructed to read a data item, the CPU sends the necessary control signals to the input device.

fetching an instruction
Retrieving an instruction from main memory.

To process a program stored in main memory, the CPU retrieves each instruction in sequence (called **fetching an instruction**), decodes the instruction, and then retrieves any data needed to carry out that instruction. Next, the CPU processes the data it retrieved and stores the results in main memory.

The CPU can perform arithmetic operations such as addition, subtraction, multiplication, and division. It can also compare the contents of two memory cells—for example, deciding which contains the larger value or if the values are equal—and make decisions based on the results of that comparison.

integrated circuit (IC)
An electronic device containing a large number of circuits/components housed inside a silicon case.
microprocessor
A central processing unit packaged in an integrated circuit.
register
A high-speed memory location inside the CPU.

The circuitry of a modern CPU is housed in a single **integrated circuit** (IC) or chip—millions of miniature circuits manufactured on a sliver of silicon. An integrated circuit that is a full central processing unit is called a **microprocessor**. A CPU's current instruction and data values are stored temporarily inside the CPU in special high-speed memory locations called **registers**.

Input/Output Devices

keyboard
A computer input device for typing sequences of letter or digit characters.
cursor
A moving place marker that indicates the position on the screen where the next character will be displayed.
function keys
A special keyboard key used to select a particular operation; the operation selected depends on the program being used.

We use input/output (I/O) devices to communicate with the computer—to enter data for a computation and to observe the results of that computation. The most common input device is a keyboard and the most common output device is a monitor.

A computer **keyboard** resembles a typewriter keyboard (see Figure 1.6). When you press a letter or digit key, that character is sent to main memory and is displayed on the monitor at the position of the **cursor**, a moving place marker. A computer keyboard has extra keys for special purposes. For example, on the keyboard shown in Figure 1.6, the 12 keys in the top row labeled F1 through F12 are **function keys**. The functions of these keys depend on the program currently being executed; that is, pressing F4 in one program will usually not produce the same results as pressing F4 in another program.

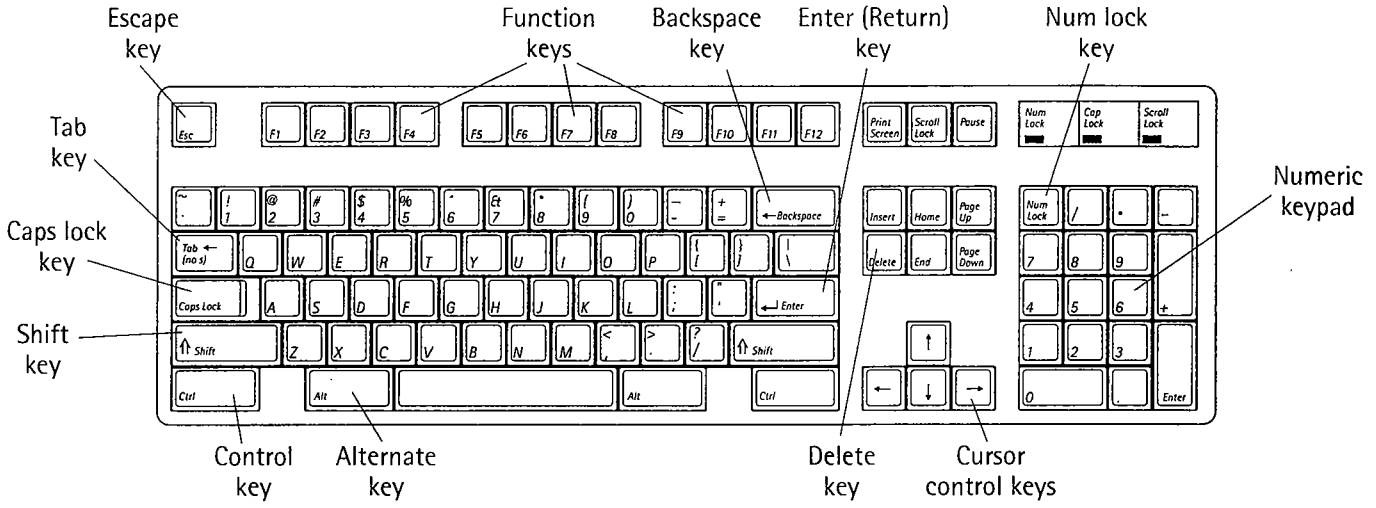


Figure 1.6 Computer keyboard

Other special keys let you delete characters, move the cursor, and "enter" a line of data you typed at the keyboard.

Another common input device is a **mouse**, a handheld pointing device. Moving the mouse on your desk causes the mouse cursor, a small arrow or symbol on the screen, to move in the same direction as the mouse. You can use the mouse to select an operation by pointing and clicking: moving the mouse cursor to a word or picture (called an **icon**) that represents the computer operation and then pressing a mouse button to start the operation selected.

mouse
A handheld input device that moves a cursor on the computer screen to select an operation.

icon
A picture representing a computer operation that is activated by clicking a mouse.

A **scanner** is used to process a page of text or a picture and to generate a sequence of bits as its digital image. This digital image is sent to memory so it can be processed and stored on disk. Some printers include a scanner.

scanner
A device for generating a digital image of a document or picture.

A Webcam (abbreviation for Web camera) is a device that sends a video to a computer. Webcams are often used to send a video of the person using the computer over the Internet.

A **monitor** displays the output of currect computer operations. Once the image disappears from the monitor screen, it is lost. If you want a hard copy (a printed version) of some information, you must send that information to a different output device, a printer.

monitor
A computer output device for providing a temporary display of information.

Computer Networks

Network technology was invented to connect computers together and let them share resources. Unlike a mainframe, which is a single computer shared by many users, a **network** is made of many computers that share resources. Within an organization, a **local area network (LAN)** lets many personal computers access sharable resources from a larger computer called

network
An interconnected collection of computers that share resources.

local area network (LAN)
A network of computers in a single organization.

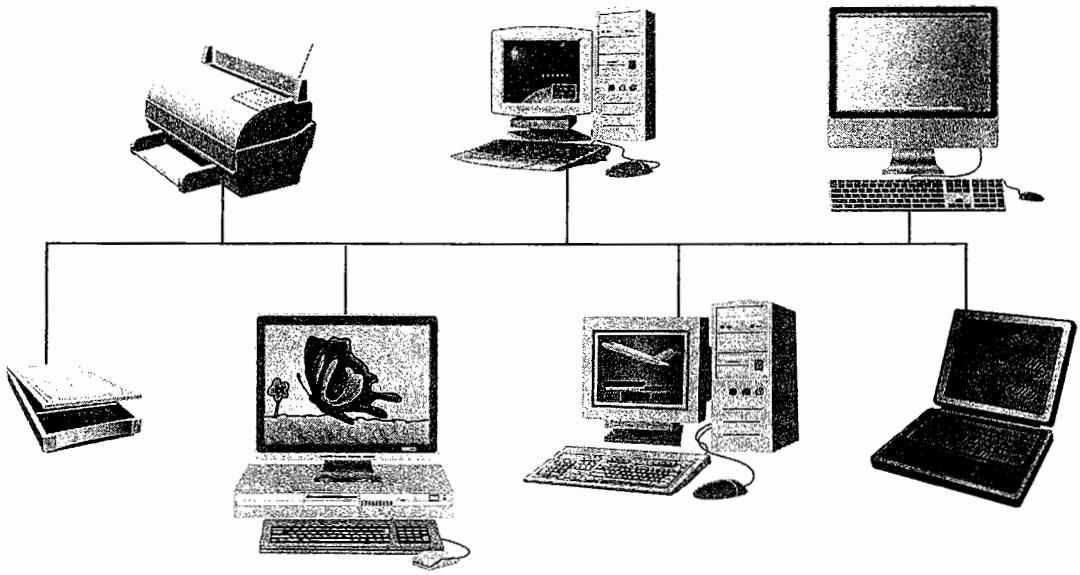
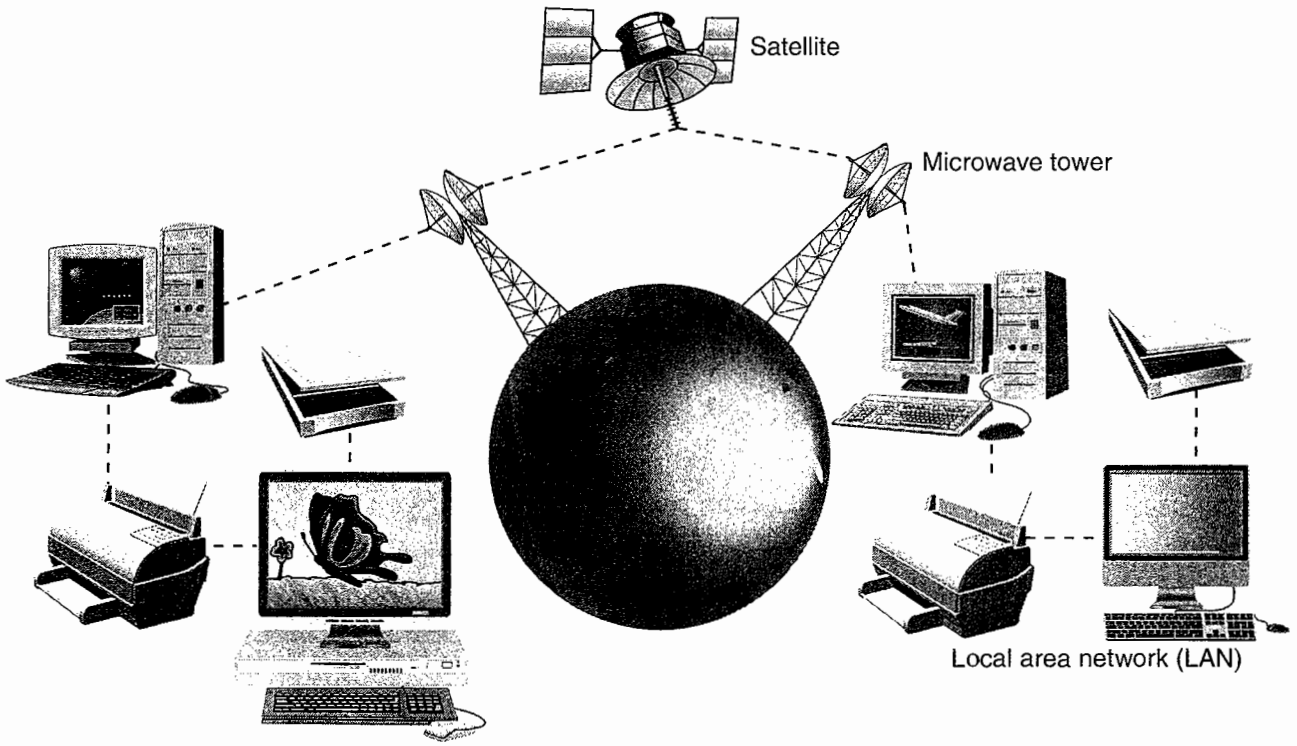


Figure 1.7 Local area network

a **server** (see Figure 1.7). A network that links individual computers and local area networks over a large geographic area is called a **wide area network (WAN)** (see Figure 1.8). The best-known WAN is the **Internet**, which is composed of university, corporate, government, and public-access networks. The World Wide Web is accessed via the Internet.



igure 1.8 A wide area network with satellite relays of microwave signals

erver
A computer that provides esources to other com- puters in a network.
vide area network WAN)
A network such as the nternet that connects computers and LANs over large geographic area.
nternet
An interconnected group- ng of computer networks rom all over the world; rovides access to the Vorld Wide Web.

If your computer is connected to a modem, you can connect to the Internet through a telephone line or television or fiber-optic cable. A **modem** (**mo**dulator **dem**odulator) converts binary computer data into a form that can be transmitted to another computer. At the computer on the receiving end, another modem converts the signal received back to binary data. Early modems for telephone lines transmitted at only 300 baud (300 bits per second). Today's telephone modems transmit over 50,000 bits per second, or if you have a digital subscriber line (**DSL connection**), the associated modem can transmit 1.5 million bits per second while allowing you to use the same line simultaneously for voice calls. Another high-speed option is **cable Internet access**, which brings Internet data to your computer along a channel just like a television channel, using the same coaxial cable that carries cable TV.

modem
A device that converts binary data into a form that can be transmitted between computers over telephone lines or coaxial cable.

DSL connection (digital subscriber line)
A high-speed Internet connection that uses a telephone line and does not interfere with simultaneous voice communication on the same line.

cable Internet access
Two-way high-speed transmission of Internet data over the coaxial cable that carries cable television signals.

The World Wide Web

The **World Wide Web (the Web)** was introduced in 1989; it is the newest and the most popular feature on the Internet. The Web was developed at CERN (the European Laboratory for Particle Physics) as an effective and uniform way of accessing all the information on the Internet. Today you can use the Web to send, view, retrieve, and search for information or to create a Web page. (There are no controls or checks for accuracy on the Web; you should keep this in mind when reading or using such information.)

World Wide Web
A collection of interconnected documents that may be accessed from virtually any computer in the world.

To access and navigate the Web, you need a **Web browser**. A Web browser is a program with a **graphical user interface (GUI)** that displays the text and graphics in a Web document and activates the **hyperlinks** to other documents. Clicking on a link to a Web document causes that document to be transferred from the computer where it is stored to your own computer.

Web browser
A program that lets users display and view a Web document and activate links to other documents.

graphical user interface (GUI)
Displayed pictures and menus that allow users to select commands and data.

hyperlink
A connection in a Web document to other related documents that users can activate by pressing a mouse button.

EXERCISES FOR SECTION 1.2

Self-Check

1. What are the contents of memory cells 1 and 998 in Figure 1.3? What memory cells contain the letter H and the number 75.62?
2. Explain the purpose of main memory, secondary memory, CPU, and the disk. What input and output devices will be used with your computer? What is a computer network?
3. List the following in order of smallest to largest: byte, bit, main memory, WAN, memory cell, secondary memory, LAN.

4. What do you think each of the instructions in memory cells 6 though 8 in Figure 1.3 means?
5. Which came first, the Internet or the World Wide Web?
6. Indicate whether each of these devices usually provides temporary, semi-permanent, or permanent storage: ROM, RAM, flash drive, CD, hard disk.
7. Which device from the list in Exercise 6 does a word processor use to store a letter while it is being typed? To store the letter after you are finished? To store a software package you purchase? To store some very large files you no longer need?
8. Explain the use of a Web browser and hyperlinks in a Web document.
9. If a computer were instructed to sum the contents of memory cells 1 and 3 of Figure 1.3 and store the result in memory cell 5, what would be stored in memory cell 5?
10. One bit can have two values, 0 or 1. A combination of 2 bits can have values 00, 01, 10, 11. List the values you can have with 3 bits. Do the same for 4 bits. Write a formula that tells you how many values you can have with n bits.

1.3 Computer Software

In the previous section, we surveyed the components of a computer system, collectively called hardware. We also covered the basic operations by which a computer accomplishes tasks: repeated fetching and execution of instructions. In this section, we focus on these all-important lists of instructions, called computer programs or computer software. We will first consider the software that makes the hardware accessible to the user, then look at the various levels of computer languages in which software is written and at the process of creating and running a new program.

Operating System

operating system (OS) Software that controls a user's interaction with the computer hardware and software and that manages the computer resources.

The computer programs that control the interaction of the user with the computer hardware compose the **operating system (OS)**. The operating system may be compared to the conductor of an orchestra, for it is responsible for directing all computer operations and managing all computer resources. Usually, part of the operating system is stored permanently in a read-only memory (ROM) chip so it will be available as soon as the computer is turned on. (A computer can look at the values in ROM but can't write new values to the chip.) This portion of the OS contains the

instructions that will load into RAM the rest of the operating system code, which typically resides on disk. Loading the operating system into RAM is called **booting the computer**.

booting the computer
Starting the computer by loading part of the operating system from disk into memory (RAM) and executing it.

Among the operating system's many responsibilities are the following:

1. Communicating with the computer user: receiving commands and carrying them out or rejecting them with an error message.
2. Managing allocation of memory, processor time, and other resources for various tasks.
3. Collecting input from the keyboard, mouse, and other input devices, and providing this data to the currently running program.
4. Conveying program output to the screen, printer, or other output device.
5. Accessing data from secondary storage.
6. Writing data to secondary storage.

In addition, the OS of a computer with multiple users must verify each user's right to use the computer and ensure that each user can access only data for which he or she has authorization.

Table 1.2 lists some widely used operating systems. An OS with a command-line interface displays a brief message called a **prompt** that indicates readiness to receive input; the user can then type a command at the keyboard. Listing 1.1 shows the entry of a UNIX command (ls temp/misc) requesting a list of the names of all the files (`Gridvar.cpp`, `Gridvar.exe`, `Gridok.dat`) in subdirectory `misc` of directory `temp`. Here, the prompt is

prompt
A message displayed by the computer indicating its readiness to receive data or a command from the user.

Table 1.2 Some Widely Used Operating Systems Characterized by Interface Type

Command-Line Interface	Graphical User Interface
UNIX	Macintosh OS
MS-DOS	Windows 7
VMS	Windows Vista
	OS/2 Warp
	UNIX + X Window System
	Linux

Listing 1.1 Entering a UNIX command to display a directory

```
mycomputer:~> ls temp/misc
Gridvar.cpp     Gridvar.exe     Gridok.dat

mycomputer:~>
```

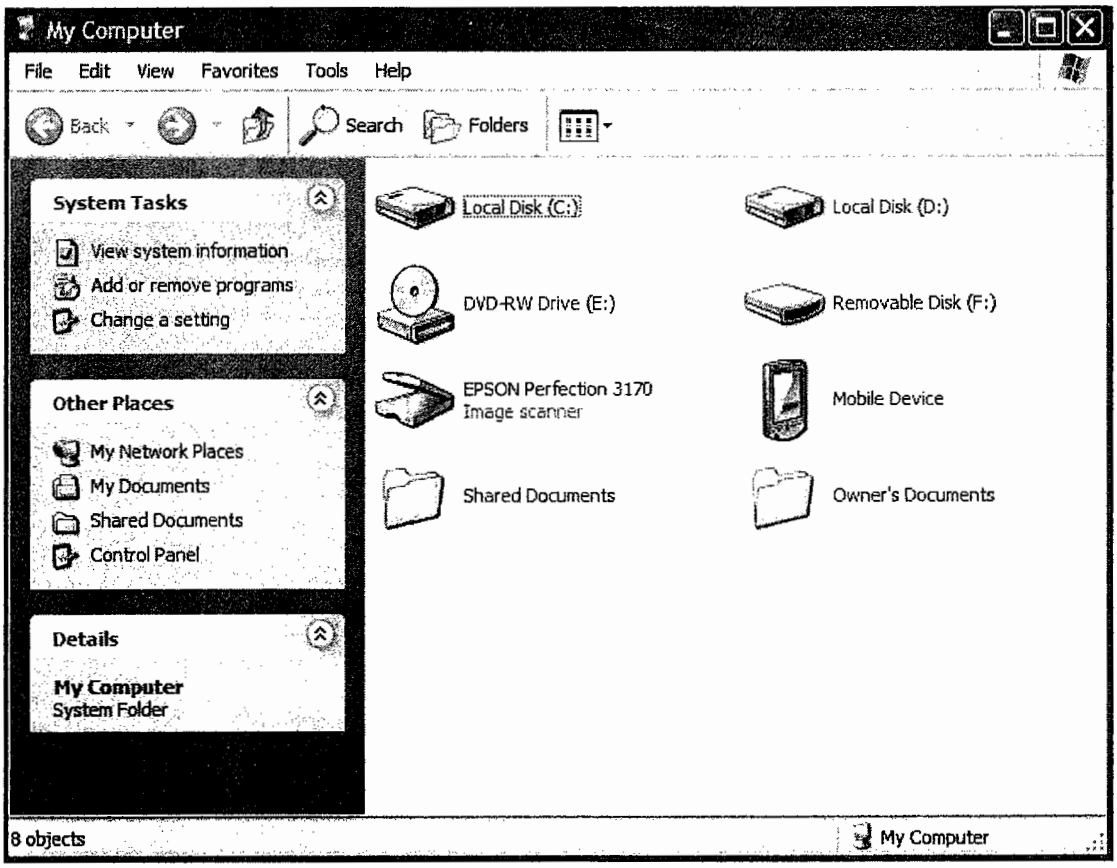
`mycomputer:~>`. (In this and all subsequent listings showing program runs, input typed by the user is shown in color to distinguish it from computer-generated text.)

In contrast, operating systems with a graphical user interface provide the user with a system of icons and menus. To issue commands, the user moves the mouse or touch-pad cursor to point to the appropriate icon or menu selection and pushes a button once or twice. Figure 1.9 shows the window that pops up when you double-click on the My Computer icon on the desktop of a Windows GUI. You can view the directories of the hard drive (C:), backup drive (D:), optical drive (E:), or flash drive (F:) by double-clicking the appropriate icon.

Application Software

pplication programs
oftware used for a
pecific task such as word
rocessing, accounting, or
latabase management.

Application programs are designed to accomplish specific tasks. For example, a word-processing application such as Microsoft Word or WordPerfect creates a document, a spreadsheet application such as Excel automates tedious numerical calculations and generates charts that depict data, and



igure 1.9 Accessing secondary storage devices through Windows

a database management application such as Access or dBASE enables data storage and quick keyword-based access to large collections of records.

Computer users typically buy application software on CDs or they download software files from a Web site. The software is saved on the hard disk and then *installed* which makes it ready to use. When buying software, verify that the program you are purchasing is compatible with both your operating system and your computer hardware. Programmers use programming languages, the subject of the next section, to write most commercial software.

installing software
Making an application available on a computer by copying it from a CD or downloading it from a Web site.

Programming Languages

Developing new software requires writing lists of instructions for a computer to execute. Software developers rarely write instructions in **machine language**, a language of binary numbers directly understood by a computer. A drawback of machine language is that it is not standardized: every type of CPU has a different machine language. This same drawback also applies to the somewhat more readable **assembly language**, a language in which computer operations are represented by mnemonic codes rather than binary numbers, and variables can be given names rather than binary memory addresses.

machine language
A list of binary instructions for a particular CPU.

assembly language
A language whose instructions are in the form of mnemonic codes and variable names.

Table 1.3 shows a small machine language program fragment that adds two numbers and an equivalent fragment in assembly language. Notice that each assembly language instruction corresponds to exactly one machine language instruction. (The assembly language memory cells labeled A and B are space for data; they are not instructions.) The symbol ? indicates that we don't know the contents of the memory cells with addresses 00000100 and 00000101.

To write programs that are independent of the CPU on which they will be executed, software designers use **high-level languages**, which combine algebraic expressions and English words. For example, the machine/assembly

high-level language
A machine programming language that combines algebraic expressions and English words.

Table 1.3 A Program in Machine and Assembly Language

Memory Addresses	Machine Language Instructions	Assembly Language Instructions
00000000	00010101	RTV A
00000001	00010110	ADD B
00000010	00110101	STO A
00000011	01110111	HLT
00000100	?	A ?
00000101	?	B ?

language program fragment shown in Table 1.3 would be a single statement in a high-level language:

```
a = a + b;
```

This statement means "Add the data in memory cells a and b, and store the result in memory cell a (replacing its previous value)."

Many high-level languages are available. Table 1.4 lists some of the most widely used ones along with the application areas that first popularized them.

Object-Oriented Programming

We will focus on C++, an object-oriented programming language derived from C. C++ was developed by Bjarne Stroustrup of AT&T's Bell Laboratories in the mid-1980s and was formally standardized in 1998. Object-oriented programming languages are popular because they make it easier to reuse and adapt previously written software. Another object-oriented language listed in Table 1.4 is Java, which is widely used on the Web.

An object is an entity that has particular properties. Some of these properties can be encoded into a computer program as data and some can be encoded as **methods** for operating on the data.

method
An operation that can be performed on an object's data.

As an example of an object, consider a hypothetical automobile. You may visualize wheels, a steering wheel, a body shape, and a color. These are the attributes, or data, of the automobile. You can also imagine the actions associated with operating an automobile, such as starting the engine, driving forward or in reverse, and applying the brakes. These activities are analogous to methods. The attributes and methods we use to characterize an automobile are an **abstraction**, or model, of an automobile.

abstraction
A representation or model of a physical object.

Table 1.4 Common High-Level Languages

High-Level Language	Original Purpose
BASIC	Teaching college students how to use the computer in their courses
C	Writing system software
C++	Extension of C supporting object-oriented programming
COBOL	Performing business data processing
FORTRAN (**For**mula **tran**slation)	Performing engineering and scientific applications
Java	A highly portable object-oriented language used for programming on the Web
Lisp	Performing artificial intelligence applications that require manipulating abstract symbols

The description of an automobile thus provides a definition of a hypothetical automobile. As such, it describes a **class**—the class *automobile*. The class definition can be used as a template to construct actual **objects**, or instances of the class, such as your car and your parents' car. Both cars have all the attributes of an automobile, but they differ in detail—for example, one may be red and the other white.

class
An entity that defines the properties of a hypothetical object.
object
A member, or instance, of a class that has all the properties described by the class definition.

The distinction between the terms *class* and *object* sometimes gets a bit blurry. A class definition, or class, describes the properties (attributes) of an abstract or hypothetical object. Actual objects are instances of the class. An object's data fields provide storage for information, and an object's methods process or manipulate this information. Some of the data stored in an object may be computed using a method. For example, fuel efficiency can be computed from the power of the engine and the weight of the car (along with some other factors). Table 1.5 shows some attributes of the class *automobile* and two objects of that class.

Classes can have other classes as components. For example, a car has an engine and wheels; both components can be defined as separate classes. Finally, classes can be organized into a hierarchy of **subclasses** and **superclasses**, where a subclass has all the properties of a superclass and some additional properties that are not part of the superclass. Figure 1.10 shows that the class *automobile* is a subclass of the class *vehicle* (an automobile is a vehicle that has passenger seats), and the class *vehicle* is a superclass of the class *automobile*. The class *truck* is also a subclass of the class *vehicle* (a truck is a vehicle that has a truck bed), and the class *vehicle* is also a superclass of the class *truck*. The class *sports car* is a subclass of the class *automobile* (a sports car is an automobile with a powerful engine), so it is also a subclass of the class *vehicle* but not of the class *truck*. The classes *vehicle* and *automobile* are both superclasses of the class *sports car*.

subclass
A class that is derived from a superclass but with additional attributes.
superclass
A class that serves as a base class for other classes with additional attributes.

Objects in a subclass inherit properties (both data and methods) from a superclass. For example, all three subclasses shown in Figure 1.10 have a driver's seat and can be driven (properties inherited from class *vehicle*).

These basic principles help programmers organize their solutions to problems. In particular, rather than starting each program from scratch, programmers use object-oriented programming because it encourages reuse

Table 1.5 The Relationship Between a Class and Objects of the Class

Class `automobile`	Object `yourCar`	Object parentsCar
Color	Red	White
Make	Toyota	Buick
Model	Coupe	Sedan
Year	1999	2003

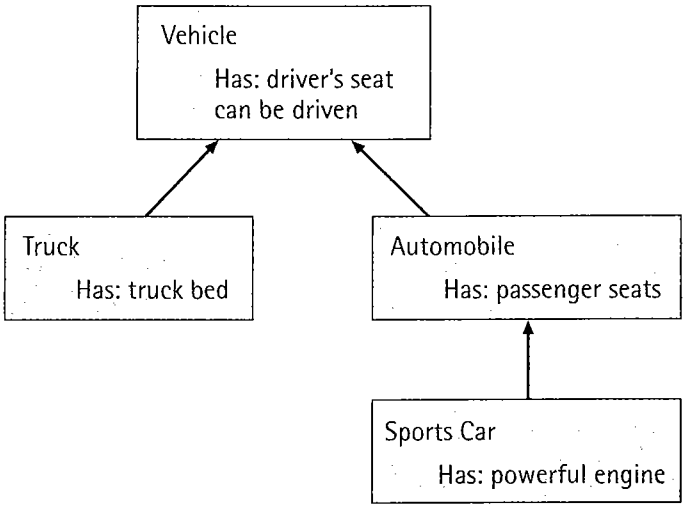


Figure 1.10 A class hierarchy

of code (programs) that are already written. They can use existing classes as components of new classes, or they can create new classes that are subclasses of existing classes. Either way, methods that were programmed for the existing classes can be used with objects of the new classes. In this book, we will focus on using existing classes as components of new classes rather than creating subclasses of existing classes.

EXERCISES FOR SECTION 1.3

Self-Check

1. What do you think the six high-level language statements below mean?
   a. `profit = gross - net;`
   b. `fahren = 1.8 * celsius + 32;`
   c. `fraction = percent / 100.0;`
   d. `sum = sum + x;`
   e. `newPrincipal = oldPrincipal * (1.0 + interest);`
   f. `celsius = 5 * (fahren - 32.0) / 9;`
2. List two reasons why it would be preferable to write a program in C++ rather than machine language.
3. Explain the relationship between the data and methods of a class.
4. What are two ways that object-oriented programming facilitates code reuse by programmers?
5. What is an abstraction in programming?
6. Is an object an instance of a class or vice versa?

1.4 Processing a High-Level Language Program

Although programmers find it much easier to express problem solutions in high-level languages, computers do NOT understand these languages. Before a high-level language program can be executed, it must be translated into the target computer's machine language. The program that does this translation is called a **compiler**. Figure 1.11 illustrates the role of the compiler in the process of developing and testing a high-level language program. Both the input to and (when successful) the output from the compiler are programs.

compiler
Software that translates a high-level language program into machine language.

The input to the compiler is a **source file** containing the text of a high-level language program. The software developer creates this file with a word processor or **editor**. The source file is in text format, which means that it is a collection of character codes. For example, you might type a program into a file called `myprog.cpp`. The compiler will scan this source file, checking to see if it follows the high-level language's rules of **syntax** (grammar). If the program is syntactically correct, the compiler saves, in an **object file**, the machine language instructions that carry out the program's purpose. For program `myprog.cpp`, the object file created might be named `myprog.obj`. This file's format is binary, which means that you should not send it to a printer, display it on your monitor, or try to work with it in a word processor because it will appear to be meaningless garbage. If the source program contains syntax errors, the compiler lists these errors but does not create an object file. The developer must return to the editor, correct the errors, and recompile the program.

source file
File containing a program written in a high-level language; the input for a compiler.

editor
Software used to create, edit (change), and store a source file on disk.

syntax
Grammar rules of a programming language.

object file
File of machine language instructions that is the output of a compiler.

Although an object file contains machine language instructions, not all of the instructions are complete. High-level languages provide the software developer with many named chunks of code for operations that he or she will likely need. Almost all high-level language programs use at least one of these code chunks, which reside in other object files available to the system. The **linker program** combines code from other object files with the new object file, creating a complete machine language program that is ready to run. For your sample program, the linker might name the executable file it creates `myprog.exe`.

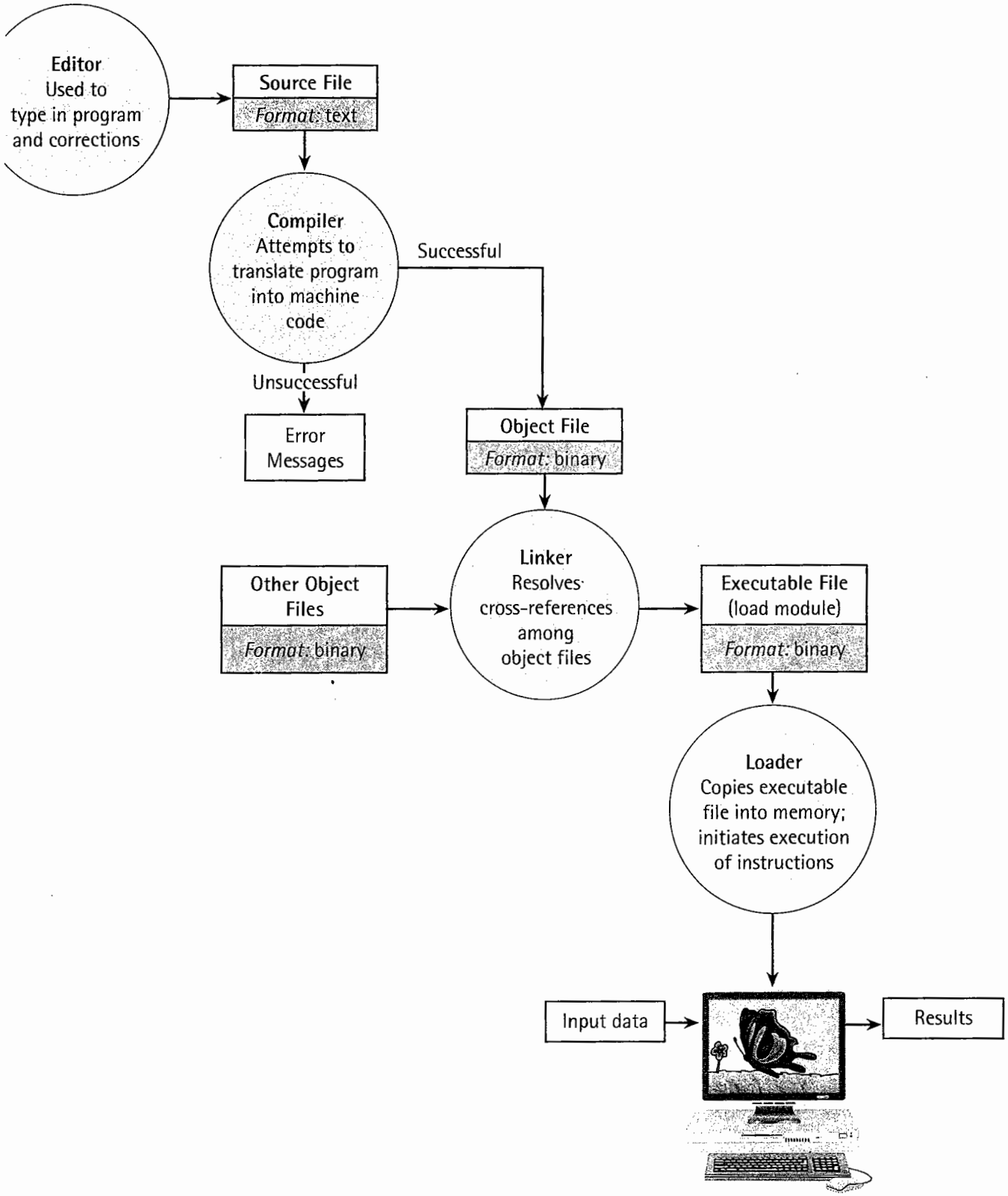
linker
Software that combines object files to create an executable machine language program.

As long as `myprog.exe` is just stored on your disk, it does nothing. To run it, the **loader** must copy all its instructions into memory and direct the CPU to begin execution with the first instruction. As the program runs, it takes input data from one or more sources and sends results to output and/or secondary storage devices.

loader
Software that copies an executable machine language program into memory and starts its execution.

Some computer systems require the user to ask the OS to separately carry out each step illustrated in Figure 1.11. However, many high-level language compilers are now sold as part of an **integrated development environment (IDE)**, a package that combines a simple editor with a compiler, linker, and loader. IDEs give the developer menus from which to select steps, and if the developer tries a step that is out of sequence, the environment simply fills in the missing steps automatically.

integrated development environment (IDE)
Software package combining an editor, a compiler, a linker, a loader, and tools for finding errors.



gure 1.11 Preparing a high-level language program for execution

One caution: the IDE might not automatically save to disk the source, object, and executable files created by the programmer; it may simply leave these programs in memory. This approach saves the time and disk space needed to make copies, and keeps the code available in memory for application of the next step in the translation/execution process. But the developer can risk losing the only copy of the source file if a serious program error causes termination of the IDE program. To prevent such a loss, users must explicitly save the source file to disk after every modification before attempting to run the program.

Executing a Program

To execute a program, the CPU must examine each program instruction in memory and send out the command signals required to carry out the instruction. Although normally the instructions are executed in sequence, as we will discuss later, it is possible to have the CPU skip over some instructions or execute some instructions more than once.

While a program runs, data can be entered into memory and manipulated in some specified way. Special program instructions are used for entering a program's data (called **input data**) into memory. After the input data have been processed, instructions for displaying or printing values in memory can be executed to display the program results. The lines displayed by a program are called the **program output**.

input data
The data values that are entered by a program.

program output
The lines displayed by a program.

Let's look at the example shown in Figure 1.12—executing a water bill program stored in memory. Step 1 of the program enters into memory data that describe the amount of water used. In Step 2, the program manipulates the data and stores the results of the computations in memory. In the final step, the computed results are displayed as a water bill.

EXERCISES FOR SECTION 1.4

Self-Check

1. Would a syntax error be found in a source program or an object program? What system program would find a syntax error if one existed? What system program would you use to correct it?
2. Explain the differences between a source program, an object program, and an executable program. Which do you create, and which does the compiler create? Which does the linker or loader create?
3. What is an IDE? What does the program developer need to be concerned about when using an IDE?

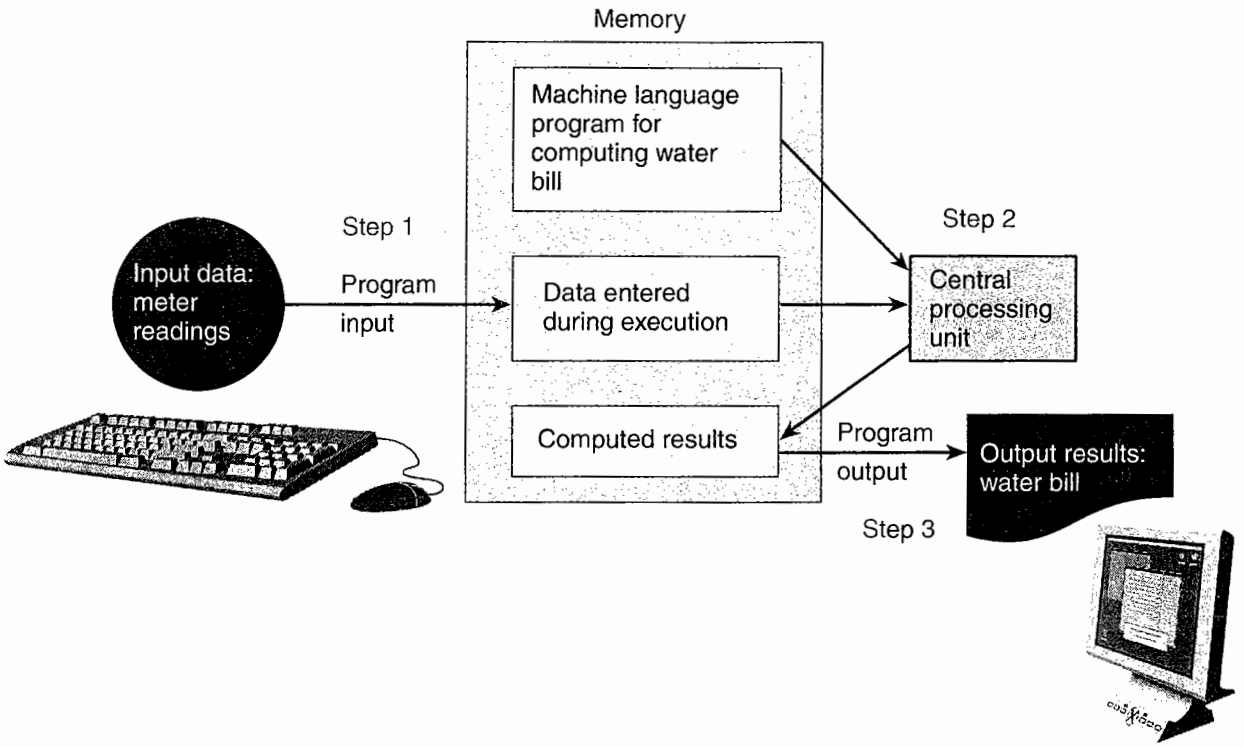


Figure 1.12 Flow of information during program execution

4. Explain how you could lose your source program if an error occurs while running a program using an IDE.
5. Explain the role of the compiler, linker, loader, and editor.

1.5 The Software Development Method

Programming is a problem-solving activity. If you are a good problem solver, you have the potential to become a good programmer, so one goal of this book is to help you improve your problem-solving ability. Problem-solving methods vary with subject area. Business students learn to solve problems with a *systems approach*, while engineering and science students use the *engineering and scientific method*. Programmers use the *software development method*.

We will focus on the first five steps listed below. The last step applies to commercial software, not student programs.

1. Specify the problem requirements.
2. Analyze the problem.
3. Design the algorithm to solve the problem.
4. Implement the algorithm.

5. Test and verify the completed program.
6. Maintain and update the program.

PROBLEM

Specifying the problem requirements forces you to state the problem clearly and unambiguously to gain a precise understanding of what is required for its solution. Your objective is to eliminate unimportant aspects and zero in on the root problem. This goal may not be as easy to achieve as it sounds. You may, for instance, need more information from the person who posed the problem.

ANALYSIS

Analyzing the problem involves identifying the problem *inputs* (the data you have to work with), *outputs* (the desired results), and any additional requirements for or constraints on the solution. At this stage, you should also determine the format in which the results should be displayed (for example, as a table with specific column headings) and develop a list of problem variables and their relationships. These relationships may be expressed as formulas.

If Steps 1 and 2 are not done properly, you will solve the wrong problem. Read the problem statement carefully (1) to obtain a clear idea of the problem and (2) to determine the inputs and outputs. It may be helpful to underline phrases in the problem statement that identify the inputs and outputs, as in the following example:

> Compute and display the *total cost of apples* given the number of *pounds of apples* purchased and the *cost per pound of apples*.

Next, summarize the information contained in the underlined phrases:

> ***Problem Inputs***
>
> quantity of apples purchased (in pounds)
> cost per pound of apples (in dollars per pound)
>
> ***Problem Output***
>
> total cost of apples (in dollars)

Once you know the problem inputs and outputs, develop a list of formulas that specify relationships between them. The general formula

> Total cost = Unit cost × Number of units

computes the total cost of any item purchased. Using the variables for our particular problem, we get the formula

> Total cost of apples = Cost per pound × Pounds of apples

abstraction
The process of modeling a problem to extract the essential variables and their relationships.

This process of modeling a problem to extract the essential variables and their relationships is called **abstraction**.

DESIGN

algorithm
A list of steps for solving a problem.

top-down design
Breaking a problem into its major subproblems and then solving the subproblems.

Designing the algorithm to solve the problem requires you to develop a list of steps (an **algorithm**) to solve the problem and then verify that the algorithm solves the problem as intended. Writing the algorithm is often the hardest part of the problem-solving process. Don't try to solve every detail of the problem at the beginning; instead, use top-down design. In **top-down design** (also called *divide and conquer*), you first list the major steps, or subproblems, that need to be solved, then solve the original problem by solving each of its subproblems. Most computer algorithms consist of at least the following subproblems.

ALGORITHM FOR A PROGRAMMING PROBLEM

1. Get the data.
2. Perform the computations.
3. Display the results.

algorithm refinement
Developing a detailed list of steps to solve a particular step in an algorithm.

Once you know the subproblems, you can attack each one individually. The perform-the-computations step, for example, may need to be broken down into a more detailed list of steps through a process called **algorithm refinement**.

You are using top-down design when creating an outline for a term paper. You first create an outline of the major topics, then refine it by filling in subtopics for each major topic. Once the outline is complete, you begin writing the text for each subtopic.

desk-checking
The step-by-step simulation of the computer execution of an algorithm.

Desk-checking is an important, and often overlooked, part of algorithm design. To desk check an algorithm, you must carefully perform each algorithm step (or its refinements) just as a computer would and verify that the algorithm works as intended. You'll save time and effort if you locate algorithm errors early in the problem-solving process.

IMPLEMENTATION

Implementing the algorithm (Step 4 in the software development method) involves writing it as a program—converting each algorithm step into one or more statements in a programming language.

TESTING

Testing and verifying the program requires testing the completed program to verify that it works as desired. Don't rely on just one test case; run the program several times using different sets of data, making sure that it works correctly for every situation provided for in the algorithm.

MAINTENANCE

Maintaining and updating the program involves modifying a program to remove previously undetected errors and to keep it up-to-date as government regulations or company policies change. Many organizations maintain a program for five years or more, often after the programmers who originally coded it have left.

A disciplined approach is essential if you want to create programs that are easy to read, understand, and maintain. You must follow accepted program style guidelines (which will be stressed in this book) and avoid tricks and programming shortcuts.

Caution: Failure Is Part of the Process

Although a step-by-step approach to problem solving is helpful, it does not guarantee that following these steps will result in a correct solution the first time, every time. The importance of verification highlights an essential truth of problem solving: The first (and also the second, third, or twentieth) attempt at a solution *may be wrong*. Probably the most important distinction between outstanding problem solvers and less proficient ones is that the former are not discouraged by initial failures. Instead, they see the faulty and near-correct early solutions as pointing toward a better understanding of the problem. One of the most inventive problem solvers of all time, Thomas Edison, is noted for his positive interpretation of the thousands of failed experiments that contributed to his incredible record of invention; he always saw those failures in terms of the helpful data they yielded about what did *not* work.

EXERCISES FOR SECTION 1.5

Self-Check

1. List the steps of the software development method.
2. Do you desk-check an algorithm before you refine it or after you refine it? Explain your answer.

1.6 Applying the Software Development Method

Throughout this book, we use the first five steps of the software development method to solve programming problems. These example problems, presented as *Case Studies*, begin with a statement of the *problem*. This is

followed by an *analysis*, where we identify the data requirements for the problem: the problem inputs and the desired outputs. Next, we formulate the *design* of the initial algorithm and then we refine it. Finally, we *implement* the algorithm as a C++ program. We also provide a sample run of the program and discuss how to *test* the program.

Now let's walk through a sample case study. This example includes a running commentary on the process, which you can use as a model in solving other problems.

case study Converting Miles to Kilometers

PROBLEM

Your summer internship has you working with a surveyor. Part of your job is to study some maps that give distances in kilometers and some that use miles. The surveyor prefers to deal in metric measurements. Write a program that performs the necessary conversion.

ANALYSIS

The first step in solving this problem is to determine what you are asked to do. You must convert from one system of measurement to another, but are you supposed to convert from kilometers to miles, or vice versa? The problem states that you prefer to deal in metric measurements, so you must convert distance measurements in miles to kilometers. Therefore, the problem input is *distance in miles* and the problem output is *distance in kilometers*. To write the program, you need to know the relationship between miles and kilometers. Consulting a metric table shows that one mile equals 1.609 kilometers.

The data requirements and relevant formulas are listed below. The memory cell that will contain the problem input is identified by `miles`, and `kms` identifies the memory cell that will contain the program result, or the problem output.

DATA REQUIREMENTS

Problem Input

`miles` *the distance in miles*

Problem Output

`kms` *the distance in kilometers*

Relevant Formula

1 mile = 1.609 kilometers

DESIGN

The next step is to formulate the algorithm that solves the problem. Begin by listing the three major steps, or subproblems, of the algorithm.

ALGORITHM

1. Get the distance in miles.
2. Convert the distance to kilometers.
3. Display the distance in kilometers.

Now decide whether any steps of the algorithm need further refinement or whether they are perfectly clear as stated. Step 1 (getting the data) and Step 3 (displaying a value) are basic steps and require no further refinement. Step 2 is fairly straightforward, but some detail might help:

Step 2 Refinement

2.1 The distance in kilometers is 1.609 times the distance in miles.

We list the complete algorithm with refinements below to show you how it all fits together. The refinement of Step 2 is numbered as Step 2.1 and is indented under Step 2.

ALGORITHM WITH REFINEMENTS

1. Get the distance in miles.
2. Convert the distance to kilometers.
   2.1 The distance in kilometers is 1.609 times the distance in miles.
3. Display the distance in kilometers.

Let's desk check the algorithm before going further. If Step 1 gets a distance of 10.0 miles, Step 2.1 would convert it to 1.609×10.00 or 16.09 kilometers. This correct result would be displayed by Step 3.

IMPLEMENTATION

To implement the solution, you must write the algorithm as a C++ program. You would first tell the C++ compiler about the problem data requirements—that is, what memory cell names you are using and what kind of data will be stored in each memory cell. Next, convert each algorithm step into one or more C++ statements. If an algorithm step has been refined, convert the refinements, not the original step, into C++ statements.

Listing 1.2 shows the C++ program along with a sample execution. The algorithm steps are in lines that begin with `//`; the program statements follow the algorithm steps. The sample execution is highlighted in the color screen. The input data typed in by the program user are in color. Don't worry about understanding the details of this program yet; we explain the program in the next chapter.

TESTING

How do you know the sample run is correct? You should always examine program results carefully to make sure that they make sense. In this run, a distance of 10.0 miles is converted to 16.09 kilometers, as it should be. To verify that the program works properly, enter a few more test values of miles. You don't need to try more than a few test cases to verify that a simple program like this is correct.

Listing 1.2 Converting miles to kilometers

```
// miles.cpp
// Converts distance in miles to kilometers.

#include <iostream>
using namespace std;

int main()                                  // start of main function
{
      const float KM_PER_MILE = 1.609;      // 1.609 km in a mile
      float miles,                          // input: distance in miles
            kms;                            // output: distance in kilometers

      // Get the distance in miles.
      cout << "Enter the distance in miles: ";
      cin >> miles;

      // Convert the distance to kilometers.
      kms = km_per_mile * miles;

      // Display the distance in kilometers.
      cout << "The distance in kilometers is " << kms << endl;

      return 0;                             //Exit the main function
}

Enter the distance in miles: 10.0
The distance in kilometers is 16.09
```

EXERCISES FOR SECTION 1.6

Self-Check

1. Change the algorithm for the metric conversion program to convert distance in kilometers to miles.
2. List the data requirements, formulas, and algorithm for a program that converts a volume from quarts to liters.

1.7 Professional Ethics for Computer Programmers

We end this introductory chapter with a discussion of professional ethics for computer programmers. Like other professionals, computer programmers and software system designers (called software engineers) need to follow certain standards of professional conduct.

Privacy and Misuse of Data

As part of their jobs, programmers may have access to large data banks or databases containing sensitive information on financial transactions or personnel, or information that is classified as "secret" or "top secret." Programmers should always behave in a socially responsible manner and not retrieve information that they are not entitled to see. They should not use information to which they are given access for their own personal gain, or do anything that would be considered illegal, unethical, or harmful to others. Just as doctors and lawyers must keep patient information confidential, programmers must respect an individual's rights to privacy.

A programmer who changes information in a database containing financial records for his or her own personal gain—for example, changes the amount of money in a bank account—is guilty of **computer theft** or **computer fraud**. This is a felony that can lead to fines and imprisonment.

computer theft (computer fraud)
Illegally obtaining money by falsifying information in a computer database.

Computer Hacking

You may have heard about "computer hackers" who break into secure data banks by using their own computer to call the computer that controls access to the data bank. Classified or confidential information retrieved in this way has been sold to intelligence agencies of other countries. Other hackers have tried to break into computers to retrieve information for their own amusement or as a prank, or just to demonstrate that they can do it. Regardless of the intent, this activity is illegal, and the government will prosecute anyone who does it. Your university probably addresses this kind of activity in your student handbook. The punishment is likely similar to that for other criminal activity, because that is exactly what it is.

Another illegal activity sometimes practiced by hackers is attaching harmful code, called a **virus**, to another program so that the virus code copies itself throughout a computer's disk memory. A virus can cause sporadic activities to disrupt the operation of the host computer—for example, unusual messages may appear on the screen at certain times—or cause the host computer to erase portions of its own disk memory, destroying valuable information and programs. Viruses are spread from one computer

virus
Code attached to another program that spreads through a computer's disk memory, disrupting the computer or erasing information.

to another in various ways—for example, if you copy a file that originated on another computer that has a virus, or if you open an e-mail message that is sent from an infected computer. A computer **worm** is a virus that can replicate itself on other network computers, causing these computers to send multiple messages over the network to disrupt its operation or shut it down. Certainly, data theft and virus propagation should not be considered harmless pranks; they are illegal and carry serious penalties.

worm
A virus that can disrupt a network by replicating itself on other network computers.

Plagiarism and Software Piracy

Using someone else's programs without permission is also unprofessional behavior. Although it is certainly permissible to use modules from libraries that have been developed for reuse by their own company's programmers, you cannot use another programmer's personal programs or programs from another company without getting permission beforehand. Doing so could lead to a lawsuit, with you or your company having to pay damages.

Modifying another student's code and submitting it as your own is a fraudulent practice—specifically, plagiarism—and is no different than copying paragraphs of information from a book or journal article and calling it your own. Most universities have severe penalties for plagiarism that may include failing the course and/or being dismissed from the university. Be aware that even if you modify the code slightly or substitute your own comments or different variable names, you are still guilty of plagiarism if you are using another person's ideas and code. To avoid any question of plagiarism, find out beforehand your instructor's rules about working with others on a project. If group efforts are not allowed, make sure that you work independently and submit only your own code.

Many commercial software packages are protected by copyright laws against **software piracy**—the practice of illegally copying software for use on another computer. If you violate this law, your company or university can be fined heavily for allowing this activity to occur. Besides the fact that software piracy is against the law, using software copied from another computer increases the possibility that your computer will receive a virus. For all these reasons, you should read the copyright restrictions on each software package and adhere to them.

software piracy
Violating copyright agreements by illegally copying software for use in another computer.

Misuse of a Computer Resource

Computer system access privileges or user account codes are private property. These privileges are usually granted for a specific purpose—for example, for work to be done in a particular course or for work to be done during the time you are a student at your university. The privilege should be protected; it should not be loaned to or shared with anyone else and should

not be used for any purpose for which it was not intended. When you leave the institution, this privilege is normally terminated and any accounts associated with the privilege will be closed.

Computers, computer programs, data, and access (account) codes are like any other property. If they belong to someone else and you are not explicitly given permission to use them, then do not use them. If you are granted a use privilege for a specific purpose, do not abuse the privilege or it will be taken away.

Legal issues aside, it is important that we apply the same principles of right and wrong to computerized property and access rights as to all other property rights and privileges. If you are not sure about the propriety of something you want to do, ask first. As students and professionals in computing, we set an example for others. If we set a bad example, others are sure to follow.

EXERCISES FOR SECTION 1.7

Self-Check

1. Some computer users will not open an e-mail message unless they know the person who sent it. Why might someone adopt this policy?
2. Find out the penalty for plagiarism at your school.
3. Why is it a good policy to be selective about opening e-mail attachments?
4. Define the terms *virus* and *worm*.

Chapter Review

1. The basic components of a computer are main and secondary memory, the CPU, and input and output devices.
2. Main memory is organized into individual storage locations called memory cells.
   - Each memory cell has a unique address.
   - A memory cell is a collection of bytes; a byte is a collection of eight bits.
   - A memory cell is never empty, but its initial contents may be meaningless to your program.
   - The current contents of a memory cell are destroyed whenever new information is stored in that cell.
   - Programs must be copied into the memory of the computer before they can be executed.
   - Data cannot be manipulated by the computer until they are first stored in memory.

7. What processes are needed to transform a C++ program to a machine language program that is ready for execution?
8. Explain the relationship between memory cells, bytes, and bits.
9. What is the difference between a command-line interface and a GUI? Which are you using? Which kind is used on a PC?
10. Name three high-level languages and describe their main usage.
11. What are the differences between RAM and ROM? Which contains the instructions that execute when you first boot your computer? Which can be extended? Which contains information that does not disappear when you turn your computer off?
12. What is the major reason for programming in an object-oriented language?
13. In the following problem statement, identify the problem inputs, problem outputs, and some relevant formulas.

    You are writing a program to help balance your checkbook. This program displays the new balance in your account after you enter the starting balance and a transaction. Each transaction has two data items: an indication of whether the transaction is a deposit or a withdrawal and the dollar amount of the transaction.

Answers to Quick-Check Exercises

1. Compiler, high-level, machine, assembler, machine
2. false
3. translation, linking, loading, execution
4. source
5. compiler, source
6. object
7. executable
8. editor
9. less, smaller
10. problem specification, problem analysis, program design, program implementation, program testing
11. main (a, b, f), secondary (c, d, e, g)
12. A class describes the properties of a category of objects whereas an object is a specific instance of a class.
13. methods, data or attributes
14. sub, super

Bjarne Stroustrup

Bjarne Stroustrup is the designer of the C++ programming language. He is currently the College of Engineering Chair in Computer Science professor at Texas A&M University and an AT&T Labs' Fellow. He is a member of the National Academy of Engineering, an ACM Fellow, and an IEEE Fellow. In 1993 Stroustrup won the ACM Grace Murray Hopper award for his early work on C++, and in 1990 he was named one of America's twelve top young scientists by Fortune *magazine. Stroustrup has also authored several books, including* The C++ Programming Language *and* The Design and Evolution of C++, *which are the definitive C++ reference books.*

What is your educational background? Why did you decide to study computer science?

I have a master's degree in mathematics with computer science from the University of Århus in Denmark and a Ph.D. in computer science from Cambridge University in England. I liked math, but wanted something that had a practical application. Thinking about that led me to computers.

What was your first job in the computer industry? What did it entail?

I first had a series of programming jobs while I was studying in Århus, which helped me avoid serious study debts. I programmed ledger [billing] systems for small firms such as lumberyards, accounting systems, payroll systems, etc. Interestingly enough, this involved my collaborating with both computer salesmen and people running businesses. (They knew their businesses, but nothing about computers.)

In such collaboration, I designed a system and modified it for better usability. In addition, I did the complete detailed design, implementation, and documentation. That taught me a lot—especially that a program is only a part of a larger system and that people directly depend on it.

My first full-time job was as a researcher at AT&T Bell Labs. I experimented with distributed systems and eventually with programming and programming languages as tools for building systems.

What drove the development of C++? Which specific issues were you trying to address when you developed the language?

I was working with some problems in distributed systems and networking. I needed a tool that would help me to structure my programs well and also allow me to write efficient code. I designed C++ to give me the design techniques of Simula and the low-level flexibility and efficiency of C. The further evolution of C++ has been dominated by the same aim of enabling code that is simultaneously elegant and efficient.

What is a typical day like for you?

Usually the first thing I do when I get up (at about 7 A.M.) and the last thing I do before I go to bed (after 11 P.M.) is look at

my e-mail. In between, things vary. I read a bit, go to a few talks, and write a lot—both code and text. I avoid meetings whenever I can. As a professor, I naturally lecture a couple of times a week during term-time and see a lot of students.

I try to do something that has nothing to do with work every day. I spend time with my family, read a lot of literature and history, and run a few miles when time and the weather allows it.

Which person in the computer science field has inspired you?

I have always had a deep respect for Kristen Nygaard. He is one of the designers of Simula [the first programming language to support object-oriented programming, designed in the mid-1960s], a gentleman, and a thoroughly enjoyable person to be with. Together with his friend and colleague, Ole-Johan Dahl, he received the 2002 Turing award.

Do you have any advice for students learning C++?

Study systems and programming, not just programming languages. A language is a tool, so it should be studied in the context of problems that deserve to be solved. Studying a language in isolation is sterile.

Start C++ with an up-to-date ISO-standard-conforming implementation and use standard library facilities, such as strings, vectors, maps, and iostreams, right from the beginning. Do not fiddle with pointers, C-style strings, and other low-level facilities until you understand the basics of scope, functions, looks, variables, etc. and until you have a real need. Focus on concepts and techniques. Learn programming language features as needed to express ideas. Don't think that a single programming language is all you'll ever need to know.

What advice do you have for students entering the computer science field?

Don't just study computers. Learn something that will give you an idea of what to use computers for. Consider making computer science a minor or the major with some other interesting field as a minor.

Study something just because it is hard and interesting at least once in your life; not everything has to be obviously useful or directly applicable to something specific.

How do you see C++ evolving over the next few years?

In 2003, the standards committee issued a set of minor corrections and clarifications. That shows the dedication to stability and detail that is necessary for a mature language. However, no living language can remain unchanged for more than a few years, and a more significant revision of the standard, called C++0x, is in progress. The emphasis will be on providing a larger and better standard library. The changes to the core language will be close to 100% compatible and focus on providing better support for systems programming and library building. You can get an idea of some of the libraries considered for the next standard by looking at boost.org. These include smart pointers, regular expressions, threads, and file system access. The language extensions currently under consideration focus on better support for generic programming and on making the language (and standard library) rules more regular and easier to understand. The emphasis on performance and suitability for systems programming will be maintained. See my home pages for more information about C++0x.

3. Information in secondary memory is of two types: program files and data files. Secondary memory stores information in semipermanent form and is less expensive than main memory.
4. A computer cannot think for itself; a programming language is used to instruct it in a precise and unambiguous manner to perform a task.
5. The three categories of programming languages are machine language (meaningful to the computer), high-level language (meaningful to the programmer), and assembly language, which is similar to machine language except that it uses special mnemonic codes for operations and names for memory cells instead of numeric addresses.
6. Several system programs are used to prepare a high-level language program for execution. An *editor* enters a high-level language program into memory. A *compiler* translates a high-level language program (the source program) into machine language (the object program). The *linker* links this object program to other object files, creating an executable program, and the *loader* loads the executable program into memory. Sometimes these steps are combined into an Integrated Development Environment (IDE).
7. Programming a computer can be fun, if you're patient, organized, and careful. The software development method for solving problems using a computer can be of considerable help in your programming work. We emphasize five major steps in this problem-solving process:
   - Problem specification
   - Problem analysis
   - Program design
   - Program implementation
   - Program testing
8. Through the operating system, you can issue commands to the computer and manage files.
9. Follow ethical standards of conduct in everything you do pertaining to computers. This means don't copy software that is copyright protected, don't hack into someone else's computer, don't send files that may be infected to others, and don't submit someone else's work as your own or lend your work to another student.

Quick-Check Exercises

1. A _____ may translate statements in _____ language into several statements in _____ language while a statement in _____ language usually is translated into one statement in _____ language.
2. After a program has been executed, all program results are automatically displayed. True or false?

3. Specify the correct order for these operations: execution, translation, loading, linking.
4. A high-level language program is saved on disk as a(n) _____ file.
5. The _____ finds syntax errors in the _____ file.
6. Before linking, a machine-language program is saved on disk as a(n) _____ file.
7. After linking, a machine-language program is saved on disk as a(n) _____ file.
8. The _____ program is used to create and save the source file.
9. Computers are becoming (more/less) expensive and (bigger/smaller) in size.
10. List the five steps of the software development method that you should use to write your programs.
11. Determine whether each characteristic below applies to main memory or secondary memory.
    a. Faster to access
    b. Volatile
    c. May be extended almost without limit
    d. Less expensive
    e. Used to store files
    f. Central processor accesses it to obtain the next machine-language instruction for execution
    g. Provides semipermanent data storage
12. In object-oriented programming, explain the difference between the terms *class* and *object*.
13. The _____ of an object operate on the _____ of the object.
14. In a class hierarchy, a _____ class inherits data and methods from its _____ class.

Review Questions

1. List at least three kinds of information stored in a computer.
2. List two functions of the CPU.
3. List two input devices, two output devices, and two secondary storage devices.
4. A computer can think. True or false?
5. What programs are combined in an IDE?
6. Describe two advantages of programming in a high-level language such as C++.