Introduction to Computers and C++ Programming

1.1 Computer Systems 2
Hardware 2
Software 7
High-Level Languages 8
Compilers 9
History Note 12

1.2 Programming and Problem-Solving 13
Algorithms 14
Program Design 15
Object-Oriented Programming 17
The Software Life Cycle 18

1.3 Introduction to C++ 19
Origins of the C++ Language 19
A Sample C++ Program 20
`Pitfall:` Using the Wrong Slash in \n 24
`Programming Tip:` Input and Output Syntax 25
Layout of a Simple C++ Program 25
`Pitfall:` Putting a Space before the `include` File Name 27
Compiling and Running a C++ Program 28
`Programming Tip:` Getting Your Program to Run 28

1.4 Testing and Debugging 31
Kinds of Program Errors 32
`Pitfall:` Assuming Your Program Is Correct 33

Chapter Summary 34
Answers to Self-Test Exercises 35
Programming Projects 37

Introduction to Computers and C++ Programming

The whole of the development and operation of analysis are now capable of being executed by machinery ... As soon as an Analytical Engine exists, it will necessarily guide the future course of science.

CHARLES BABBAGE (1792–1871)

Introduction

In this chapter we describe the basic components of a computer, as well as the basic technique for designing and writing a program. We then show you a sample C++ program and describe how it works.

1.1 Computer Systems

software

hardware

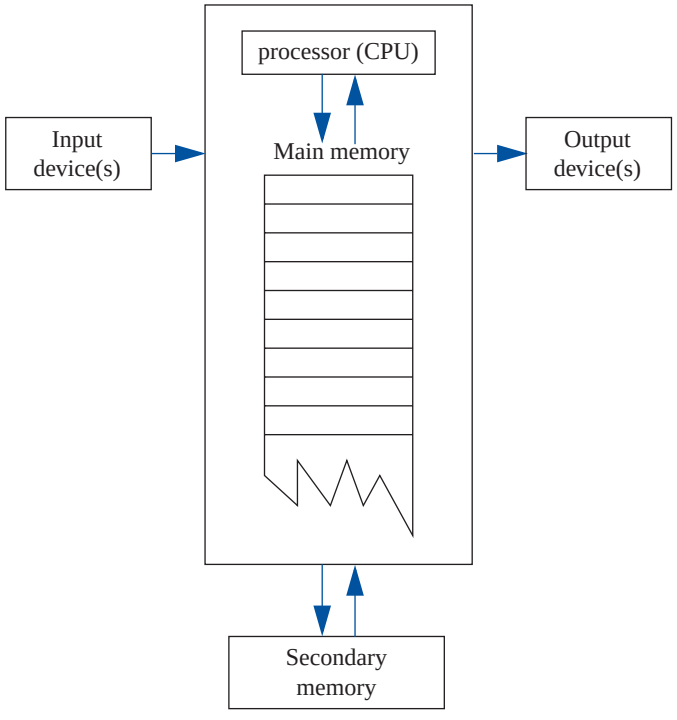
A set of instructions for a computer to follow is called a **program.** The collection of programs used by a computer is referred to as the **software** for that computer. The actual physical machines that make up a computer installation are referred to as **hardware.** As we will see, the hardware for a computer is conceptually very simple. However, computers now come with a large array of software to aid in the task of programming. This software includes editors, translators, and managers of various sorts. The resulting environment is a complicated and powerful system. In this book we are concerned almost exclusively with software, but a brief overview of how the hardware is organized will be useful.

Hardware

PCs, workstations, and mainframes

There are three main classes of computers: *PCs*, *workstations*, and *mainframes*. A **PC** (**personal computer**) is a relatively small computer designed to be used by one person at a time. Most home computers are PCs, but PCs are also widely used in business, industry, and science. A **workstation** is essentially a larger and more powerful PC. You can think of it as an "industrial-strength" PC. A **mainframe** is an even larger computer that typically requires some support staff and generally is shared by more than one user. The distinctions between PCs, workstations, and mainframes are not precise, but the terms are commonly used and do convey some very general information about a computer.

Display 1.1 Main Components of a Computer



network

A **network** consists of a number of computers connected, so they may share resources, such as printers, and may share information. A network might contain a number of workstations and one or more mainframes, as well as shared devices such as printers.

For our purposes in learning programming, it will not matter whether you are working on a PC, a mainframe, or a workstation. The basic configuration of the computer, as we will view it, is the same for all three types of computers.

The hardware for most computer systems is organized as shown in Display 1.1. The computer can be thought of as having five main components: the *input device(s)*, the *output device(s)*, the *processor* (also called the *CPU*), the *main memory*, and the *secondary memory*. The processor, main memory, and sometimes even secondary memory are normally housed in a single cabinet. The processor and main memory form the heart of a computer and can be thought of as an integrated unit.

Other components connect to the main memory and operate under the direction of the processor. The arrows in Display 1.1 indicate the direction of information flow.

input devices

An **input device** is any device that allows a person to communicate information to the computer. Your primary input devices are likely to be a keyboard and a mouse.

output devices

An **output device** is anything that allows the computer to communicate information to you. The most common output device is a display screen, referred to as a *monitor.* Quite often, there is more than one output device. For example, in addition to the monitor, your computer probably has a printer for producing output on paper. The keyboard and monitor are sometimes thought of as a single unit called a *terminal*.

In order to store input and to have the equivalent of scratch paper for performing calculations, computers are provided with *memory.* The program that the computer executes is also stored in this memory. A computer has two forms of memory, called *main memory* and *secondary memory.* The program that is being executed is kept in main memory, and main memory is, as the name implies, the most important memory.

main memory

Main memory consists of a long list of numbered locations called *memory locations*; the number of memory locations varies from one computer to another, ranging from a few thousand to many millions, and sometimes even into the billions. Each memory location contains a string of zeros and ones. The contents of these locations can change. Hence, you can think of each memory location as a tiny blackboard on which the computer may write and erase. In most computers, all memory locations contain the same number of zero/one digits. A digit that can assume only the values zero or one is called a **binary digit** or a **bit.** The memory locations in most computers contain eight bits (or some multiple of eight bits). An eight-bit portion of memory is called a **byte,** so we may refer to these numbered memory locations as *bytes*. To rephrase the situation, you can think of the computer's main memory as a long list of numbered memory locations called *bytes*. The number that identifies a byte is called its **address.** A data item, such as a number or a letter, can be stored in one of these bytes, and the address of the byte is then used to find the data item when it is needed.

bit

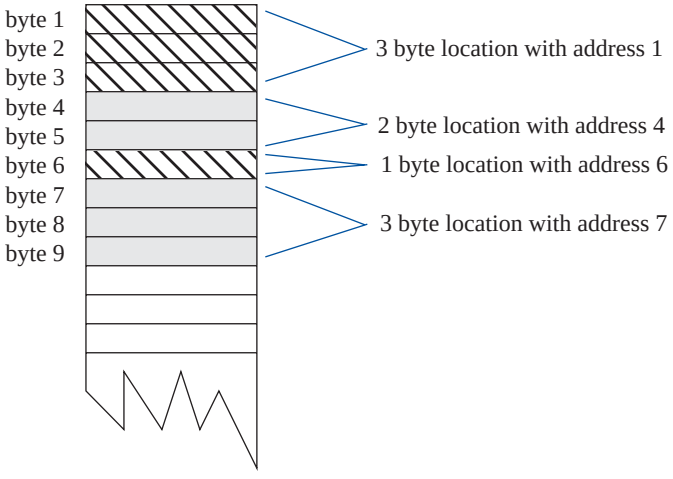
byte

address

If the computer needs to deal with a data item (such as a large number) that is too large to fit in a single byte, it will use several adjacent bytes to hold the data item. In this case the entire chunk of memory that holds the data item is still called a **memory location.** The address of the first of the bytes that make up this memory location is used as the address for this larger memory location. Thus, as a practical matter, you can think of the computer's main memory as a long list of memory locations of *varying sizes*. The size of each of these locations is expressed in bytes and the address of the first byte is used as the address (name) of that memory location. Display 1.2 shows a picture of a hypothetical computer's main memory. The sizes of the memory locations are not fixed, but can change when a new program is run on the computer.

memory location

Display 1.2 Memory Locations and Bytes



Bytes and Addresses

Main memory is divided into numbered locations called **bytes.** The number associated with a byte is called its **address.** A group of consecutive bytes is used as the location for a data item, such as a number or letter. The address of the first byte in the group is used as the address of this larger memory location.

The fact that the information in a computer's memory is represented as zeros and ones need not be of great concern to you when programming in C++ (or in most any other programming language). There is, however, one point about this use of zeros and ones that will concern us as soon as we start to write programs. The computer needs to interpret these strings of zeros and ones as numbers, letters, instructions, or other types of information. The computer performs these interpretations automatically according to certain coding schemes. A different code is used for each different type of item that is stored in the computer's memory: one code for letters, another for whole numbers, another for fractions, another for instructions, and so on. For example, in one commonly used set of codes, 01000001 is the code for the letter A and also for the number 65. In order to know what the string 01000001 in a particular location

Why Eight?

A **byte** is a memory location that can hold eight bits. What is so special about eight? Why not ten bits? There are two reasons why eight is special. First, eight is a power of 2. (8 is 2^3.) Since computers use bits, which only have two possible values, powers of two are more convenient than powers of 10. Second, it turns out that it requires eight bits (one byte) to code a single character (such as a letter or other keyboard symbol).

stands for, the computer must keep track of which code is currently being used for that location. Fortunately, the programmer seldom needs to be concerned with such codes and can safely reason as though the locations actually contained letters, numbers, or whatever is desired.

The memory we have been discussing up until now is the main memory. Without its main memory, a computer can do nothing. However, main memory is only used while the computer is actually following the instructions in a program. The computer also has another form of memory called *secondary memory* or *secondary storage*. (The words *memory* and *storage* are exact synonyms in this context.)

secondary memory

Secondary memory is the memory that is used for keeping a permanent record of information after (and before) the computer is used. Some alternative terms that are commonly used to refer to secondary memory are *auxiliary memory*, *auxiliary storage*, *external memory*, and *external storage*.

files

Information in secondary storage is kept in units called **files,** which can be as large or as small as you like. A program, for example, is stored in a file in secondary storage and copied into main memory when the program is run. You can store a program, a letter, an inventory list, or any other unit of information in a file.

Several different kinds of secondary memory may be attached to a single computer. The most common forms of secondary memory are *hard disks*, *diskettes*, and *CDs*. (Diskettes are also sometimes referred to as *floppy disks*.) **CDs** (compact disks) used on computers are basically the same as those used to record and play music. CDs for computers may be read-only so that your computer can read, but cannot change, the data on the CD; CDs for computers can also be read/write CDs, which can have their data changed by the computer. Information is stored on hard disks and diskettes in basically the same way as it is stored on CDs. **Hard disks** are fixed in place and are normally not removed from the disk drive. **Diskettes** and CDs can be easily removed from the disk drive and carried to another computer. Diskettes and CDs have the advantages of being inexpensive and portable, but hard disks hold more data and operate faster. Other forms of secondary memory are also available, but this list covers most forms that you are likely to encounter.

CDs, disks, and diskettes

RAM

Main memory is often referred to as **RAM** or **random access memory.** It is called *random access* because the computer can immediately access the data in any memory location. Secondary memory often requires **sequential access,** which means that the computer must look through all (or at least very many) memory locations until it finds the item it needs.

processor, chip

The **processor** (also know as the **central processing unit,** or **CPU)** is the "brain" of the computer. When a computer is advertised, the computer company will tell you what *chip* it contains. The **chip** is the processor. The processor follows the instructions in a program and performs the calculations specified by the program. The processor is, however, a very simple brain. All it can do is follow a set of simple instructions provided by the programmer. Typical processor instructions say things like "Interpret the zeros and ones as numbers, and then add the number in memory location 37 to the number in memory location 59, and put the answer in location 43," or "Read a letter of input, convert it to its code as a string of zeros and ones, and place it in memory location 1298." The processor can add, subtract, multiply, and divide and can move things from one memory location to another. It can interpret strings of zeros and ones as letters and send the letters to an output device. The processor also has some primitive ability to rearrange the order of instructions. processor instructions vary somewhat from one computer to another. The processor of a modern computer can have as many as several hundred available instructions. However, these instructions are typically all about as simple as those we have just described.

Software

operating system

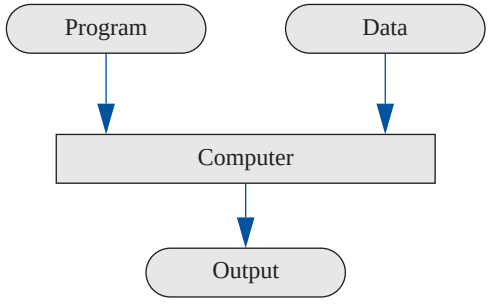
You do not normally talk directly to the computer, but communicate with it through an *operating system*. The **operating system** allocates the computer's resources to the different tasks that the computer must accomplish. The operating system is actually a program, but it is perhaps better to think of it as your chief servant. It is in charge of all your other servant programs, and it delivers your requests to them. If you want to run a program, you tell the operating system the name of the file that contains it, and the operating system runs the program. If you want to edit a file, you tell the operating system the name of the file and it starts up the editor to work on that file. To most users the operating system is the computer. Most users never see the computer without its operating system. The names of some common operating systems are *UNIX*, *DOS*, *Linux*, *Windows*, *Macintosh*, and *VMS*.

program

A **program** is a set of instructions for a computer to follow. As shown in Display 1.3, the input to a computer can be thought of as consisting of two parts, a program and some data. The computer follows the instructions in the program, and in that way, performs some process. The **data** is what we conceptualize as the input to the program. For example, if the program adds two numbers, then the two numbers

data

Display 1.3 Simple View of Running a Program



are the data. In other words, the data is the input to the program, and both the program and the data are input to the computer (usually via the operating system). Whenever we give a computer both a program to follow and some data for the program, we are said to be **running the program** on the data, and the computer is said to **execute the program** on the data. The word *data* also has a much more general meaning than the one we have just given it. In its most general sense it means any information available to the computer. The word is commonly used in both the narrow sense and the more general sense.

running a program
executing a program

High-Level Languages

There are many languages for writing programs. In this text we will discuss the C++ programming language and use it to write our programs. C++ is a high-level language, as are most of the other programming languages you are likely to have heard of, such as C, Java, Pascal, Visual Basic, FORTRAN, COBOL, Lisp, Scheme, and Ada. **High-level languages** resemble human languages in many ways. They are designed to be easy for human beings to write programs in and to be easy for human beings to read. A high-level language, such as C++, contains instructions that are much more complicated than the simple instructions a computer's processor (CPU) is capable of following.

high-level language

The kind of language a computer can understand is called a **low-level language.** The exact details of low-level languages differ from one kind of computer to another. A typical low-level instruction might be the following:

low-level language

```
ADD X Y Z
```

This instruction might mean "Add the number in the memory location called X to the number in the memory location called Y, and place the result in the memory location called Z." The above sample instruction is written in what is called **assembly language.** Although assembly language is almost the same as the language understood by the computer, it must undergo one simple translation before the computer can understand it. In order to get a computer to follow an assembly language instruction, the words need to be translated into strings of zeros and ones. For example, the word ADD might translate to 0110, the X might translate to 1001, the Y to 1010, and the Z to 1011. The version of the above instruction that the computer ultimately follows would then be:

assembly language

```
0110 1001 1010 1011
```

Assembly language instructions and their translation into zeros and ones differ from machine to machine.

Programs written in the form of zeros and ones are said to be written in **machine language,** because that is the version of the program that the computer (the machine) actually reads and follows. Assembly language and machine language are almost the same thing, and the distinction between them will not be important to us. The important distinction is that between machine language and high-level languages like C++: Any high-level language program must be translated into machine language before the computer can understand and follow the program.

machine language

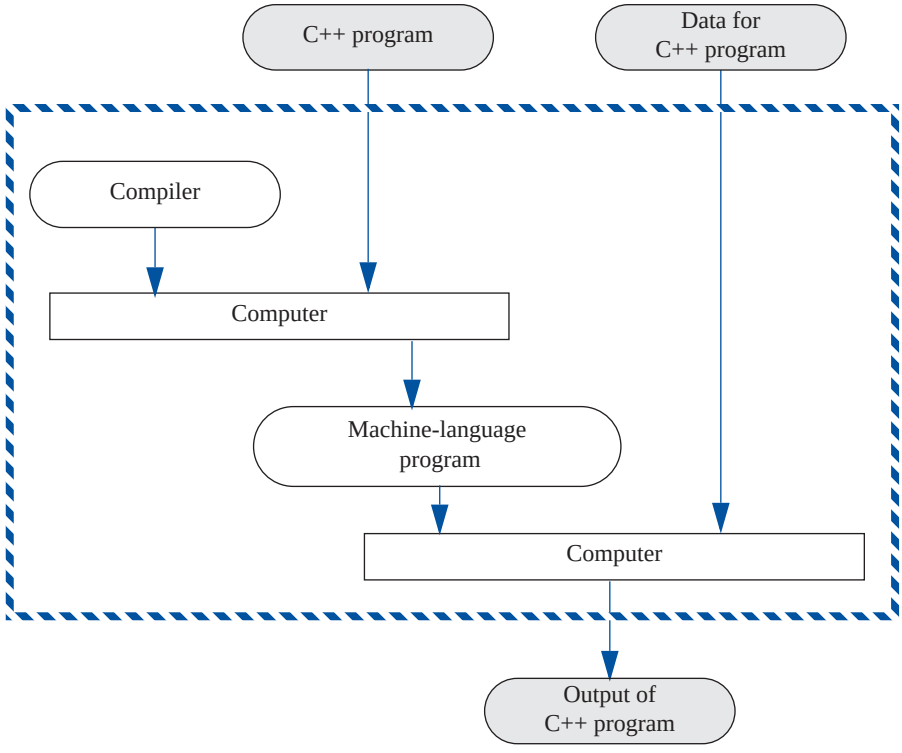
Compilers

A program that translates a high-level language like C++ to a machine language is called a **compiler.** A compiler is thus a somewhat peculiar sort of program, in that its input or data is some other program, and its output is yet another program. To avoid confusion, the input program is usually called the **source program** or **source code,** and the translated version produced by the compiler is called the **object program** or **object code.** The word **code** is frequently used to mean a program or a part of a program, and this usage is particularly common when referring to object programs. Now, suppose you want to run a C++ program that you have written. In order to get the computer to follow your C++ instructions, proceed as follows. First, run the compiler using your C++ program as data. Notice that in this case, your C++ program is not being treated as a set of instructions. To the compiler, your C++ program is just a long string of characters. The output will be another long string of characters, which is the machine-language equivalent of your C++ program. Next, run this machine-language program on what we normally think of as the data for the C++ program. The output will be what we normally conceptualize as the output of the C++ program. The basic process is easier to visualize if you have two computers available, as diagrammed in

compiler

source program
object program
code

Display 1.4 Compiling and Running a C++ Program (Basic Outline)



Display 1.4. In reality, the entire process is accomplished by using one computer two times.

The complete process of translating and running a C++ program is a bit more complicated than what we showed in Display 1.4. Any C++ program you write will

Compiler

A **compiler** is a program that translates a high-level language program, such as a C++ program, into a machine-language program that the computer can directly understand and execute.

use some operations (such as input and output routines) that have already been programmed for you. These items that are already programmed for you (like input and output routines) are already compiled and have their object code waiting to be combined with your program's object code to produce a complete machine-language program that can be run on the computer. Another program, called a **linker,** combines the object code for these program pieces with the object code that the compiler produced from your C++ program. The interaction of the compiler and the linker are diagrammed in Display 1.5. In routine cases, many systems will do this linking for you automatically. Thus, you may not need to worry about linking in very simple cases.

linking

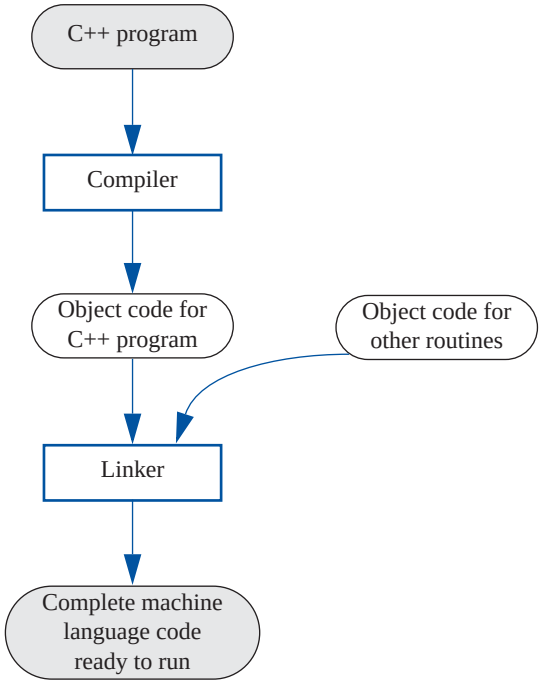
Linking

The object code for your C++ program must be combined with the object code for routines (such as input and output routines) that your program uses. This process of combining object code is called **linking** and is done by a program called a **linker.** For simple programs, linking may be done for you automatically.

SELF-TEST EXERCISES

1. What are the five main components of a computer?
2. What would be the data for a program to add two numbers?
3. What would be the data for a program that assigns letter grades to students in a class?
4. What is the difference between a machine-language program and a high-level language program?
5. What is the role of a compiler?
6. What is a source program? What is an object program?
7. What is an operating system?
8. What purpose does the operating system serve?
9. Name the operating system that runs on the computer you use to prepare programs for this course.
10. What is linking?
11. Find out whether linking is done automatically by the compiler you use for this course.

Display 1.5 Preparing a C++ Program for Running



History Note

Charles Babbage

The first truly programmable computer was designed by **Charles Babbage,** an English mathematician and physical scientist. Babbage began the project sometime before 1822 and worked on it for the rest of his life. Although he never completed the construction of his machine, the design was a conceptual milestone in the history of computing. Much of what we know about Charles Babbage and his computer design comes from the writings of his colleague **Ada Augusta.** Ada Augusta was the daughter of the poet Byron and was the Countess of Lovelace. Ada Augusta is frequently given the title of the first computer programmer. Her comments, quoted in the opening of the next section, still apply to the process of solving problems on a computer. Computers are not magic and do not, at least as yet, have the ability to formulate sophisticated solutions to all the problems we encounter. Computers simply do what the programmer orders them to do. The solutions to problems are

Ada Augusta