The chief merit of language is clearness.
—Galen

Our life is frittered away by detail. … Simplify, simplify.
—Henry David Thoreau

He had a wonderful talent for packing thought close, and rendering it portable.
—Thomas B. Macaulay

Man is still the most extraordinary computer of all.
—John F. Kennedy

**OBJECTIVES**

In this chapter you will learn:

- Basic hardware and software concepts.
- Basic object-technology concepts, such as classes, objects, attributes, behaviors, encapsulation and inheritance.
- The different types of programming languages.
- Which programming languages are most widely used.
- A typical C++ program development environment.
- The history of the industry-standard object-oriented system modeling language, the UML.
- The history of the Internet and the World Wide Web.
- To test-drive C++ applications in two popular C++ environments—GNU C++ running on Linux and Microsoft’s Visual C++®.NET running on Windows® XP.
Chapter 1  Introduction to Computers, the Internet and World Wide Web

1.1 Introduction

Welcome to C++! We have worked hard to create what we hope you will find to be an informative, entertaining and challenging learning experience. C++ is a powerful computer programming language that is appropriate for technically oriented people with little or no programming experience and for experienced programmers to use in building substantial information systems. 

C++ How to Program, Fifth Edition, is an effective learning tool for each of these audiences.

The core of the book emphasizes achieving program clarity through the proven techniques of object-oriented programming—this is an “early classes and objects” book—non-programmers will learn programming the right way from the beginning. The presentation is clear, straightforward and abundantly illustrated. We teach C++ features in the context of complete working C++ programs and show the outputs produced when those programs are run on a computer—we call this the live-code approach. The example programs are included on the CD that accompanies this book, or you may download them from www.deitel.com or www.prenhall.com/deitel.

The early chapters introduce the fundamentals of computers, computer programming and the C++ computer programming language, providing a solid foundation for the deeper treatment of C++ in the later chapters. Experienced programmers tend to read the early chapters quickly, then find the treatment of C++ in the remainder of the book both rigorous and challenging.
1.2 What Is a Computer?

Most people are at least somewhat familiar with the exciting things computers do. Using this textbook, you will learn how to command computers to do those things. Computers (often referred to as hardware) are controlled by software (i.e., the instructions you write to command the computer to perform actions and make decisions). C++ is one of today’s most popular software development languages. This text provides an introduction to programming in the version of C++ standardized in the United States through the American National Standards Institute (ANSI) and worldwide through the efforts of the International Organization for Standardization (ISO).

Computer use is increasing in almost every field of endeavor. Computing costs have been decreasing dramatically due to rapid developments in both hardware and software technologies. Computers that might have filled large rooms and cost millions of dollars a few decades ago can now be inscribed on silicon chips smaller than a fingernail, costing a few dollars each. (Those large computers were called mainframes and are widely used today in business, government and industry.) Fortunately, silicon is one of the most abundant materials on earth—it’s an ingredient in common sand. Silicon chip technology has made computing so economical that about a billion general-purpose computers are in use worldwide, helping people in business, industry and government, and in their personal lives.

Over the years, many programmers learned the programming methodology called structured programming. You will learn structured programming and an exciting newer methodology, object-oriented programming. Why do we teach both? Object orientation is the key programming methodology used by programmers today. You will create and work with many software objects in this text. You will discover however, that their internal structure is often built using structured-programming techniques. Also, the logic of manipulating objects is occasionally expressed with structured programming.

You are embarking on a challenging and rewarding path. As you proceed, if you have any questions, please send e-mail to deitel@deitel.com

We will respond promptly. To keep up to date with C++ developments at Deitel & Associates, please register for our free e-mail newsletter, the Deitel® Buzz Online, at www.deitel.com/newsletter/subscribe.html

We hope that you will enjoy learning with C++ How to Program, Fifth Edition.
Chapter 1  Introduction to Computers, the Internet and World Wide Web

A computer consists of various devices referred to as hardware (e.g., the keyboard, screen, mouse, hard disk, memory, DVDs and processing units). The programs that run on a computer are referred to as software. Hardware costs have been declining dramatically in recent years, to the point that personal computers have become a commodity. In this book, you will learn proven methods that are reducing software development costs—object-oriented programming and (in our optional Software Engineering Case Study in Chapters 2–7, 9 and 13) object-oriented design.

1.3 Computer Organization
Regardless of differences in physical appearance, virtually every computer may be envisioned as divided into six logical units or sections:

1. **Input unit.** This is the “receiving” section of the computer. It obtains information (data and computer programs) from input devices and places this information at the disposal of the other units for processing. Most information is entered into computers through keyboards and mouse devices. Information also can be entered in many other ways, including by speaking to your computer, scanning images and having your computer receive information from a network, such as the Internet.

2. **Output unit.** This is the “shipping” section of the computer. It takes information that the computer has processed and places it on various output devices to make the information available for use outside the computer. Most information output from computers today is displayed on screens, printed on paper or used to control other devices. Computers also can output their information to networks, such as the Internet.

3. **Memory unit.** This is the rapid-access, relatively low-capacity “warehouse” section of the computer. It stores computer programs while they are being executed. It retains information that has been entered through the input unit, so that it will be immediately available for processing when needed. The memory unit also retains processed information until it can be placed on output devices by the output unit. Information in the memory unit is typically lost when the computer’s power is turned off. The memory unit is often called either memory or primary memory. [Historically, this unit has been called “core memory,” but that term is fading from use today.]

4. **Arithmetic and logic unit (ALU).** This is the “manufacturing” section of the computer. It is responsible for performing calculations, such as addition, subtraction, multiplication and division. It contains the decision mechanisms that allow the computer, for example, to compare two items from the memory unit to determine whether they are equal.

5. **Central processing unit (CPU).** This is the “administrative” section of the computer. It coordinates and supervises the operation of the other sections. The CPU tells the input unit when information should be read into the memory unit, tells the ALU when information from the memory unit should be used in calculations and tells the output unit when to send information from the memory unit to certain output devices. Many of today’s computers have multiple CPUs and, hence, can perform many operations simultaneously—such computers are called multiprocessors.

Copyright © 1992-2005 by Deitel & Associates, Inc. All Rights Reserved
6. **Secondary storage unit.** This is the long-term, high-capacity “warehousing” section of the computer. Programs or data not actively being used by the other units normally are placed on secondary storage devices, such as your hard drive, until they are again needed, possibly hours, days, months or even years later. Information in secondary storage takes much longer to access than information in primary memory, but the cost per unit of secondary storage is much less than that of primary memory. Other secondary storage devices include CDs and DVDs, which can hold hundreds of millions of characters and billions of characters, respectively.

### 1.4 Early Operating Systems

Early computers could perform only one **job** or task at a time. This is often called single-user **batch processing**. The computer runs a single program at a time while processing data in groups or **batches**. In these early systems, users generally submitted their jobs to a computer center on decks of punched cards and often had to wait hours or even days before printouts were returned to their desks.

Software systems called **operating systems** were developed to make using computers more convenient. Early operating systems smoothed and speeded up the transition between jobs, and hence increased the amount of work, or **throughput**, computers could process.

As computers became more powerful, it became evident that single-user batch processing was inefficient, because so much time was spent waiting for slow input/output devices to complete their tasks. It was thought that many jobs or tasks could **share** the resources of the computer to achieve better utilization. This is achieved by **multiprogramming**. Multiprogramming involves the simultaneous operation of many jobs that are competing to share the computer’s resources. With early multiprogramming operating systems, users still submitted jobs on decks of punched cards and waited hours or days for results.

In the 1960s, several groups in industry and the universities pioneered **timesharing** operating systems. Timesharing is a special case of multiprogramming in which users access the computer through terminals, typically devices with keyboards and screens. Dozens or even hundreds of users share the computer at once. The computer actually does not run them all simultaneously. Rather, it runs a small portion of one user’s job, then moves on to service the next user, perhaps providing service to each user several times per second. Thus, the users’ programs **appear** to be running simultaneously. An advantage of timesharing is that user requests receive almost immediate responses.

### 1.5 Personal, Distributed and Client/Server Computing

In 1977, Apple Computer popularized **personal computing**. Computers became so economical that people could buy them for their own personal or business use. In 1981, IBM, the world’s largest computer vendor, introduced the IBM Personal Computer. This quickly legitimized personal computing in business, industry and government organizations, as IBM mainframes were heavily used.

These computers were “stand-alone” units—people transported disks back and forth between them to share information (often called “sneakernet”). Although early personal computers were not powerful enough to timeshare several users, these machines could be linked together in computer networks, sometimes over telephone lines and sometimes in **local area networks (LANs)** within an organization. This led to the phenomenon of dis-
distributed computing, in which an organization’s computing, instead of being performed only at some central computer installation, is distributed over networks to the sites where the organization’s work is performed. Personal computers were powerful enough to handle the computing requirements of individual users as well as the basic communications tasks of passing information between computers electronically.

Today’s personal computers are as powerful as the million-dollar machines of just a few decades ago. The most powerful desktop machines—called workstations—provide individual users with enormous capabilities. Information is shared easily across computer networks, where computers called file servers offer a common data store that may be used by client computers distributed throughout the network, hence the term client/server computing. C++ has become widely used for writing software for operating systems, for computer networking and for distributed client/server applications. Today’s popular operating systems such as UNIX, Linux, Mac OS X and Microsoft’s Windows-based systems provide the kinds of capabilities discussed in this section.

1.6 The Internet and the World Wide Web
The Internet—a global network of computers—was initiated almost four decades ago with funding supplied by the U.S. Department of Defense. Originally designed to connect the main computer systems of about a dozen universities and research organizations, the Internet today is accessible by computers worldwide.

With the introduction of the World Wide Web—which allows computer users to locate and view multimedia-based documents on almost any subject over the Internet—the Internet has exploded into one of the world’s premier communication mechanisms.

The Internet and the World Wide Web are surely among humankind’s most important and profound creations. In the past, most computer applications ran on computers that were not connected to one another. Today’s applications can be written to communicate among the world’s computers. The Internet mixes computing and communications technologies. It makes our work easier. It makes information instantly and conveniently accessible worldwide. It enables individuals and local small businesses to get worldwide exposure. It is changing the way business is done. People can search for the best prices on virtually any product or service. Special-interest communities can stay in touch with one another. Researchers can be made instantly aware of the latest breakthroughs. After you master Chapter 19, Web Programming, you will be able to develop Internet-based computer applications.

1.7 Machine Languages, Assembly Languages and High-Level Languages
Programmers write instructions in various programming languages, some directly understandable by computers and others requiring intermediate translation steps. Hundreds of computer languages are in use today. These may be divided into three general types:

1. Machine languages
2. Assembly languages
3. High-level languages

Any computer can directly understand only its own machine language. Machine language is the “natural language” of a computer and as such is defined by its hardware design.
1.7 Machine Languages, Assembly Languages and High-Level Languages

Note: Machine language is often referred to as object code. This term predates “object-oriented programming.” These two uses of “object” are unrelated.] Machine languages generally consist of strings of numbers (ultimately reduced to 1s and 0s) that instruct computers to perform their most elementary operations one at a time. Machine languages are machine dependent (i.e., a particular machine language can be used on only one type of computer). Such languages are cumbersome for humans, as illustrated by the following section of an early machine-language program that adds overtime pay to base pay and stores the result in gross pay:

```
+1300042774
+1400593419
+1200274027
```

Machine-language programming was simply too slow, tedious and error-prone for most programmers. Instead of using the strings of numbers that computers could directly understand, programmers began using English-like abbreviations to represent elementary operations. These abbreviations formed the basis of assembly languages. Translator programs called assemblers were developed to convert early assembly-language programs to machine language at computer speeds. The following section of an assembly-language program also adds overtime pay to base pay and stores the result in gross pay:

```
load    basepay
add     overpay
store   grosspay
```

Although such code is clearer to humans, it is incomprehensible to computers until translated to machine language.

Computer usage increased rapidly with the advent of assembly languages, but programmers still had to use many instructions to accomplish even the simplest tasks. To speed the programming process, high-level languages were developed in which single statements could be written to accomplish substantial tasks. Translator programs called compilers convert high-level language programs into machine language. High-level languages allow programmers to write instructions that look almost like everyday English and contain commonly used mathematical notations. A payroll program written in a high-level language might contain a statement such as

```
grossPay = basePay + overTimePay;
```

From the programmer’s standpoint, obviously, high-level languages are preferable to machine and assembly language. C, C++, Microsoft’s .NET languages (e.g., Visual Basic .NET, Visual C++ .NET and C#) and Java are among the most widely used high-level programming languages.

The process of compiling a high-level language program into machine language can take a considerable amount of computer time. Interpreter programs were developed to execute high-level language programs directly, although much more slowly. Interpreters are popular in program development environments in which new features are being added and errors corrected. Once a program is fully developed, a compiled version can be produced to run most efficiently.
1.8 History of C and C++

C++ evolved from C, which evolved from two previous programming languages, BCPL and B. BCPL was developed in 1967 by Martin Richards as a language for writing operating-systems software and compilers for operating systems. Ken Thompson modeled many features in his language B after their counterparts in BCPL and used B to create early versions of the UNIX operating system at Bell Laboratories in 1970.

The C language was evolved from B by Dennis Ritchie at Bell Laboratories. C uses many important concepts of BCPL and B. C initially became widely known as the development language of the UNIX operating system. Today, most operating systems are written in C and/or C++. C is now available for most computers and is hardware independent. With careful design, it is possible to write C programs that are portable to most computers.

The widespread use of C with various kinds of computers (sometimes called hardware platforms) unfortunately led to many variations. This was a serious problem for program developers, who needed to write portable programs that would run on several platforms. A standard version of C was needed. The American National Standards Institute (ANSI) cooperated with the International Organization for Standardization (ISO) to standardize C worldwide; the joint standard document was published in 1990 and is referred to as ANSI/ISO 9899: 1990.

Portability Tip 1.1

Because C is a standardized, hardware-independent, widely available language, applications written in C often can be run with little or no modification on a wide range of computer systems.

C++, an extension of C, was developed by Bjarne Stroustrup in the early 1980s at Bell Laboratories. C++ provides a number of features that “spruce up” the C language, but more importantly, it provides capabilities for object-oriented programming.

A revolution is brewing in the software community. Building software quickly, correctly and economically remains an elusive goal, and this at a time when the demand for new and more powerful software is soaring. Objects are essentially reusable software components that model items in the real world. Software developers are discovering that using a modular, object-oriented design and implementation approach can make them much more productive than they can be with previous popular programming techniques. Object-oriented programs are easier to understand, correct and modify.

1.9 C++ Standard Library

C++ programs consist of pieces called classes and functions. You can program each piece that you may need to form a C++ program. However, most C++ programmers take advantage of the rich collections of existing classes and functions in the C++ Standard Library. Thus, there are really two parts to learning the C++ “world.” The first is learning the C++ language itself; the second is learning how to use the classes and functions in the C++ Standard Library. Throughout the book, we discuss many of these classes and functions. P J. Plauger’s book, The Standard C Library (Upper Saddle River, NJ: Prentice Hall PTR, 1992), is a must read for programmers who need a deep understanding of the ANSI C library functions that are included in C++, how to implement them and how to use them to write portable code. The standard class libraries generally are provided by independent software vendors. Many special-purpose class libraries are supplied by independent software vendors.
1.10 History of Java

Microprocessors are having a profound impact in intelligent consumer electronic devices. Recognizing this, Sun Microsystems in 1991 funded an internal corporate research project code-named Green. The project resulted in the development of a C++-based language that its creator, James Gosling, called Oak after an oak tree outside his window at Sun. It was later discovered that there already was a computer language called Oak. When a group of Sun people visited a local coffee shop, the name Java was suggested and it stuck.

The Green project ran into some difficulties. The marketplace for intelligent consumer electronic devices did not develop in the early 1990s as quickly as Sun had antici-
Chapter 1 Introduction to Computers, the Internet and World Wide Web

The project was in danger of being canceled. By sheer good fortune, the World Wide Web exploded in popularity in 1993, and Sun saw the immediate potential of using Java to add dynamic content (e.g., interactivity, animations and the like) to Web pages. This breathed new life into the project.

Sun formally announced Java in 1995. Java generated immediate interest in the business community because of the phenomenal success of the World Wide Web. Java is now used to develop large-scale enterprise applications, to enhance the functionality of Web servers (the computers that provide the content we see in our Web browsers), to provide applications for consumer devices (such as cell phones, pagers and personal digital assistants) and for many other purposes. Current versions of C++, such as Microsoft’s Visual C++ .NET and Borland’s C++Builder™, have similar capabilities.

1.11 FORTRAN, COBOL, Pascal and Ada

Hundreds of high-level languages have been developed, but only a few have achieved broad acceptance. FORTRAN (FORmula TRANslator) was developed by IBM Corporation in the mid-1950s to be used for scientific and engineering applications that require complex mathematical computations. FORTRAN is still widely used, especially in engineering applications.

COBOL (COmmon Business Oriented Language) was developed in the late 1950s by computer manufacturers, the U.S. government and industrial computer users. COBOL is used for commercial applications that require precise and efficient manipulation of large amounts of data. Much business software is still programmed in COBOL.

During the 1960s, many large software development efforts encountered severe difficulties. Software deliveries were typically late, costs greatly exceeded budgets and the finished products were unreliable. People began to realize that software development was a far more complex activity than they had imagined. Research in the 1960s resulted in the evolution of structured programming—a disciplined approach to writing programs that are clearer, easier to test and debug and easier to modify than large programs produced with previous techniques.

One of the more tangible results of this research was the development of the Pascal programming language by Professor Niklaus Wirth in 1971. Named after the seventeenth-century mathematician and philosopher Blaise Pascal, it was designed for teaching structured programming and rapidly became the preferred programming language in most colleges. Pascal lacks many features needed in commercial, industrial and government applications, so it has not been widely accepted in these environments.

The Ada programming language was developed under the sponsorship of the U.S. Department of Defense (DOD) during the 1970s and early 1980s. Hundreds of separate languages were being used to produce the DOD’s massive command-and-control software systems. The DOD wanted a single language that would fill most of its needs. The Ada language was named after Lady Ada Lovelace, daughter of the poet Lord Byron. Lady Lovelace is credited with writing the world’s first computer program in the early 1800s (for the Analytical Engine mechanical computing device designed by Charles Babbage). One important capability of Ada, called multitasking, allows programmers to specify that many activities are to occur in parallel. Java, through a technique called multithreading, also enables programmers to write programs with parallel activities. Although multithreading is not part of standard C++, it is available through various add-on class libraries.
1.12 Basic, Visual Basic, Visual C++, C# and .NET

The BASIC (Beginner’s All-purpose Symbolic Instruction Code) programming language was developed in the mid-1960s at Dartmouth College as a means of writing simple programs. BASIC’s primary purpose was to familiarize novices with programming techniques. Microsoft’s Visual Basic language, introduced in the early 1990s to simplify the development of Microsoft Windows applications, has become one of the most popular programming languages in the world.

Microsoft’s latest development tools are part of its corporate-wide strategy for integrating the Internet and the Web into computer applications. This strategy is implemented in Microsoft’s .NET platform, which provides developers with the capabilities they need to create and run computer applications that can execute on computers distributed across the Internet. Microsoft’s three primary programming languages are Visual Basic .NET (based on the original BASIC), Visual C++ .NET (based on C++) and C# (a new language based on C++ and Java that was developed expressly for the .NET platform). Developers using .NET can write software components in the language they are most familiar with and then form applications by combining those components with components written in any .NET language.

1.13 Key Software Trend: Object Technology

One of the authors, Harvey Deitel, remembers the great frustration that was felt in the 1960s by software development organizations, especially those working on large-scale projects. During his undergraduate years, he had the privilege of working summers at a leading computer vendor on the teams developing timesharing, virtual memory operating systems. This was a great experience for a college student. But, in the summer of 1967, reality set in when the company “decommitted” from producing as a commercial product the particular system on which hundreds of people had been working for many years. It was difficult to get this software right. Software is “complex stuff.”

Improvements to software technology did emerge with the benefits of structured programming (and the related disciplines of structured systems analysis and design) being realized in the 1970s. Not until the technology of object-oriented programming became widely used in the 1990s, though, did software developers finally feel they had the necessary tools for making major strides in the software development process.

Actually, object technology dates back to the mid 1960s. The C++ programming language, developed at AT&T by Bjarne Stroustrup in the early 1980s, is based on two languages—C, which initially was developed at AT&T to implement the UNIX operating system in the early 1970s, and Simula 67, a simulation programming language developed in Europe and released in 1967. C++ absorbed the features of C and added Simula’s capabilities for creating and manipulating objects. Neither C nor C++ was originally intended for wide use beyond the AT&T research laboratories. But grass roots support rapidly developed for each.

What are objects and why are they special? Actually, object technology is a packaging scheme that helps us create meaningful software units. These can be large and are highly focussed on particular applications areas. There are date objects, time objects, paycheck objects, invoice objects, audio objects, video objects, file objects, record objects and so on. In fact, almost any noun can be reasonably represented as an object.
We live in a world of objects. Just look around you. There are cars, planes, people, animals, buildings, traffic lights, elevators and the like. Before object-oriented languages appeared, programming languages (such as FORTRAN, COBOL, Pascal, Basic and C) were focussed on actions (verbs) rather than on things or objects (nouns). Programmers living in a world of objects programmed primarily using verbs. This made it awkward to write programs. Now, with the availability of popular object-oriented languages such as C++ and Java, programmers continue to live in an object-oriented world and can program in an object-oriented manner. This is a more natural process than procedural programming and has resulted in significant productivity enhancements.

A key problem with procedural programming is that the program units do not easily mirror real-world entities effectively, so these units are not particularly reusable. It is not unusual for programmers to “start fresh” on each new project and have to write similar software “from scratch.” This wastes time and money, as people repeatedly “reinvent the wheel.” With object technology, the software entities created (called classes), if properly designed, tend to be much more reusable on future projects. Using libraries of reusable componentry, such as MFC (Microsoft Foundation Classes), Microsoft’s .NET Framework Class Library and those produced by Rogue Wave and many other software development organizations, can greatly reduce the amount of effort required to implement certain kinds of systems (compared to the effort that would be required to reinvent these capabilities on new projects).

Software Engineering Observation 1.3

Extensive class libraries of reusable software components are available over the Internet and the World Wide Web. Many of these libraries are available at no charge.

Some organizations report that the key benefit object-oriented programming gives them is not software reuse. Rather, they indicate that it tends to produce software that is more understandable, better organized and easier to maintain, modify and debug. This can be significant, because it has been estimated that as much as 80 percent of software costs are associated not with the original efforts to develop the software, but with the continued evolution and maintenance of that software throughout its lifetime.

Whatever the perceived benefits of object orientation are, it is clear that object-oriented programming will be the key programming methodology for the next several decades.

1.14 Typical C++ Development Environment

Let’s consider the steps in creating and executing a C++ application using a C++ development environment (illustrated in Fig. 1.1). C++ systems generally consist of three parts: a program development environment, the language and the C++ Standard Library. C++ programs typically go through six phases: edit, preprocess, compile, link, load and execute. The following discussion explains a typical C++ program development environment. [Note: On our Web site at www.deitel.com/books/downloads.html, we provide DEITEL® DIVE INTO™ Series publications to help you begin using several popular C++ development tools, including Borland® C++Builder™, Microsoft® Visual C++® 6, Microsoft® Visual C++® .NET, GNU C++ on Linux and GNU C++ on the Cygwin™ UNIX® environment for Windows®. We will make other DIVE INTO™ Series publications available as instructors request them.]
Phase 1: Creating a Program
Phase 1 consists of editing a file with an editor program (normally known simply as an editor). You type a C++ program (typically referred to as source code) using the editor, make any necessary corrections and save the program on a secondary storage device, such as your...
hard drive. C++ source code file names often end with the .cpp, .cxx, .cc or .C extensions (note that C is in uppercase) which indicate that a file contains C++ source code. See the documentation for your C++ environment for more information on file-name extensions.

Two editors widely used on UNIX systems are vi and emacs. C++ software packages for Microsoft Windows such as Borland C++ (www.borland.com), Metrowerks CodeWarrior (www.metrowerks.com) and Microsoft Visual C++ (www.msdn.microsoft.com/visualc/) have editors integrated into the programming environment. You can also use a simple text editor, such as Notepad in Windows, to write your C++ code. We assume the reader knows how to edit a program.

Phases 2 and 3: Preprocessing and Compiling a C++ Program

In phase 2, the programmer gives the command to compile the program. In a C++ system, a preprocessor program executes automatically before the compiler’s translation phase begins (so we call preprocessing phase 2 and compiling phase 3). The C++ preprocessor obeys commands called preprocessor directives, which indicate that certain manipulations are to be performed on the program before compilation. These manipulations usually include other text files to be compiled and perform various text replacements. The most common preprocessor directives are discussed in the early chapters; a detailed discussion of all the preprocessor features appears in Appendix F, Preprocessor. In phase 3, the compiler translates the C++ program into machine-language code (also referred to as object code).

Phase 4: Linking

Phase 4 is called linking. C++ programs typically contain references to functions and data defined elsewhere, such as in the standard libraries or in the private libraries of groups of programmers working on a particular project. The object code produced by the C++ compiler typically contains “holes” due to these missing parts. A linker links the object code with the code for the missing functions to produce an executable image (with no missing pieces). If the program compiles and links correctly, an executable image is produced.

Phase 5: Loading

Phase 5 is called loading. Before a program can be executed, it must first be placed in memory. This is done by the loader, which takes the executable image from disk and transfers it to memory. Additional components from shared libraries that support the program are also loaded.

Phase 6: Execution

Finally, the computer, under the control of its CPU, executes the program one instruction at a time.

Problems That May Occur at Execution Time

Programs do not always work on the first try. Each of the preceding phases can fail because of various errors that we discuss throughout the book. For example, an executing program might attempt to divide by zero (an illegal operation for whole-number arithmetic in C++). This would cause the C++ program to display an error message. If this occurs, you would have to return to the edit phase, make the necessary corrections and proceed through the remaining phases again to determine that the corrections fix the problem(s).

Most programs in C++ input and/or output data. Certain C++ functions take their input from cin (the standard input stream; pronounced “see-in”), which is normally the
keyboard, but cin can be redirected to another device. Data is often output to cout (the standard output stream; pronounced “see-out”), which is normally the computer screen, but cout can be redirected to another device. When we say that a program prints a result, we normally mean that the result is displayed on a screen. Data may be output to other devices, such as disks and hardcopy printers. There is also a standard error stream referred to as cerr. The cerr stream (normally connected to the screen) is used for displaying error messages. It is common for users to assign cout to a device other than the screen while keeping cerr assigned to the screen, so that normal outputs are separated from errors.

Common Programming Error 1.1

Errors like division by zero occur as a program runs, so they are called runtime errors or execution-time errors. Fatal runtime errors cause programs to terminate immediately without having successfully performed their jobs. Nonfatal runtime errors allow programs to run to completion, often producing incorrect results. [Note: On some systems, divide-by-zero is not a fatal error. Please see your system documentation.]

1.15 Notes About C++ and C++ How to Program, 5/e

Experienced C++ programmers sometimes take pride in being able to create some weird, contorted, convoluted usage of the language. This is a poor programming practice. It makes programs more difficult to read, more likely to behave strangely, more difficult to test and debug, and more difficult to adapt to changing requirements. This book is geared for novice programmers, so we stress program clarity. The following is our first “good programming practice.”

Good Programming Practice 1.1

Write your C++ programs in a simple and straightforward manner. This is sometimes referred to as KIS (“keep it simple”). Do not “stretch” the language by trying bizarre usages.

You have heard that C and C++ are portable languages, and that programs written in C and C++ can run on many different computers. Portability is an elusive goal. The ANSI C standard document contains a lengthy list of portability issues, and complete books have been written that discuss portability.

Portability Tip 1.3

Although it is possible to write portable programs, there are many problems among different C and C++ compilers and different computers that can make portability difficult to achieve. Writing programs in C and C++ does not guarantee portability. The programmer often will need to deal directly with compiler and computer variations. As a group, these are sometimes called platform variations.

We have audited our presentation against the ANSI/ISO C++ standard document for completeness and accuracy. However, C++ is a rich language, and there are some features we have not covered. If you need additional technical details on C++, you may want to read the C++ standard document, which can be ordered from the ANSI Web site at webstore.ansi.org/ansidocstore/default.asp

The title of the document is “Information Technology – Programming Languages – C++” and its document number is INCITS/ISO/IEC 14882-2003.
Chapter 1 Introduction to Computers, the Internet and World Wide Web

We have included an extensive bibliography of books and papers on C++ and object-oriented programming. We also have included a C++ Resources appendix containing many Internet and Web sites relating to C++ and object-oriented programming. We have listed several Web sites in Section 1.19 including links to free C++ compilers, resource sites and some fun C++ games and game programming tutorials.

Good Programming Practice 1.2
Read the manuals for the version of C++ you are using. Refer to these manuals frequently to be sure you are aware of the rich collection of C++ features and that you are using them correctly.

Good Programming Practice 1.3
Your computer and compiler are good teachers. If after reading your C++ language manual, you still are not sure how a feature of C++ works, experiment using a small "test program" and see what happens. Set your compiler options for "maximum warnings." Study each message that the compiler generates and correct the programs to eliminate the messages.

1.16 Test-Driving a C++ Application
In this section, you will run and interact with your first C++ application. You will begin by running an entertaining guess-the-number game, which picks a number from 1 to 1000 and prompts you to guess the number. If your guess is correct, the game ends. If your guess is not correct, the application indicates whether your guess is higher or lower than the correct number. There is no limit on the number of guesses you can make. [Note: For this test drive only, we have modified this application from the exercise you will be asked to create in Chapter 6, Functions and an Introduction to Recursion. Typically this application selects different numbers for you to guess each time you run it, because it chooses the numbers to guess at random. Our modified application chooses the same "correct" guesses every time you execute the program. This allows you to use the same guesses and see the same results that we show as we walk you through interacting with your first C++ application.]

We will demonstrate running a C++ application in two ways—using the Windows XP Command Prompt and using a shell on Linux (similar to a Windows Command Prompt). The application runs similarly on both platforms. Many development environments are available in which readers can compile, build and run C++ applications, such as Borland’s C++Builder, Metrowerks, GNU C++, Microsoft Visual C++.NET, etc. While we don’t test-drive each of these environments, we do provide information in Section 1.19 regarding free C++ compilers available for download on the Internet. Please see your instructor for information on your specific development environment. Also, we provide several Dive-Into™ Series publications to help you get started with various C++ compilers. These are available free for download at www.deitel.com/books/cpphtp5/index.html.

In the following steps, you will run the application and enter various numbers to guess the correct number. The elements and functionality that you see in this application are typical of those you will learn to program in this book. Throughout the book, we use fonts to distinguish between features you see on the screen (e.g., the Command Prompt) and elements that are not directly related to the screen. Our convention is to emphasize screen features like titles and menus (e.g., the File menu) in a semibold sans-serif Helvetica font and to emphasize file names, text displayed by an application and values you should enter into an application (e.g., GuessNumber or 500), in a sans-serif Lucida font. As you have
noticed, the defining occurrence of each term is set in blue, heavy bold. For the figures in this section, we highlight the user input required by each step and point out significant parts of the application. To make these features more visible, we have modified the background color of the Command Prompt window (for the Windows test-drive only). To modify the colors of the Command Prompt on your system, open a Command Prompt, then right click the title bar and select Properties. In the “Command Prompt” Properties dialog box that appears, click the Colors tab, and select your preferred text and background colors.

Running a C++ application from the Windows XP Command Prompt

1. Checking your setup. Read the Before You Begin section at the beginning of this textbook to make sure that you have copied the book’s examples to your hard drive correctly.

2. Locating the completed application. Open a Command Prompt window. For readers using Windows 95, 98 or 2000, select Start > Programs > Accessories > Command Prompt. For Windows XP users, select Start > All Programs > Accessories > Command Prompt. To change to your completed GuessNumber application directory, type cd C:\examples\ch01\GuessNumber\Windows, then press Enter (Fig. 1.2). The command cd is used to change directories.

3. Running the GuessNumber application. Now that you are in the directory that contains the GuessNumber application, type the command GuessNumber (Fig. 1.3) and press Enter. [Note: GuessNumber.exe is the actual name of the application; however, Windows assumes the .exe extension by default.]

4. Entering your first guess. The application displays “Please type your first guess.”, then displays a question mark (?) as a prompt on the next line (Fig. 1.3). At the prompt, enter 500 (Fig. 1.4).

5. Entering another guess. The application displays “Too high. Try again.”, meaning that the value you entered is greater than the number the application chose as the correct guess. So, you should enter a lower number for your next guess. At the prompt, enter 250 (Fig. 1.5). The application again displays “Too high. Try again.”, because the value you entered is still greater than the number that the correct guess.

Fig. 1.2 | Opening a Command Prompt window and changing the directory.

Fig. 1.3 | Running the GuessNumber application.
6. **Entering additional guesses.** Continue to play the game by entering values until you guess the correct number. Once you guess the answer, the application will display "Excellent! You guessed the number!" (Fig. 1.6).

7. **Playing the game again or exiting the application.** After guessing the correct number, the application asks if you would like to play another game (Fig. 1.6). At the "Would you like to play again (y or n)?" prompt, entering the one character y causes the application to choose a new number and displays the message "Please enter your first guess." followed by a question mark prompt (Fig. 1.7) so you can make your first guess in the new game. Entering the character n ends the application and returns you to the application’s directory at the Command Prompt (Fig. 1.8). Each time you execute this application from the beginning (i.e., Step 3), it will choose the same numbers for you to guess.

8. **Close the Command Prompt window.**
1.16 Test-Driving a C++ Application

Running a C++ Application Using GNU C++ with Linux

For this test drive, we assume that you know how to copy the examples into your home directory. Please see your instructor if you have any questions regarding copying the files to your Linux system. Also, for the figures in this section, we use a bold highlight to point out the user input required by each step. The prompt in the shell on our system uses the tilde (~) character to represent the home directory and each prompt ends with the dollar sign ($) character. The prompt will vary among Linux systems.

1. Locating the completed application. From a Linux shell, change to the completed GuessNumber application directory (Fig. 1.9) by typing

   cd Examples\ch01\GuessNumber\GNU_Linux

   then pressing Enter. The command cd is used to change directories.

2. Compiling the GuessNumber application. To run an application on the GNU C++ compiler, it must first be compiled by typing

   g++ GuessNumber.cpp -o GuessNumber

   as in Fig. 1.10. The preceding command compiles the application and produces an executable file called GuessNumber.

3. Running the GuessNumber application. To run the executable file GuessNumber, type ./GuessNumber at the next prompt, then press Enter (Fig. 1.11).

~$ cd examples/ch01/GuessNumber/GNU_Linux
~/examples/ch01/GuessNumber/GNU_Linux$
Chapter 1 Introduction to Computers, the Internet and World Wide Web

4. **Entering your first guess.** The application displays "Please type your first guess.", then displays a question mark (?) as a prompt on the next line (Fig. 1.11). At the prompt, enter **500** (Fig. 1.12). [Note: This is the same application that we modified and test-drove for Windows, but the outputs could vary, based on the compiler being used.]

5. **Entering another guess.** The application displays "Too high. Try again.", meaning that the value you entered is greater than the number the application chose as the correct guess (Fig. 1.12). At the next prompt, enter **250** (Fig. 1.13). This time the application displays "Too low. Try again.", because the value you entered is less than the correct guess.

6. **Entering additional guesses.** Continue to play the game (Fig. 1.14) by entering values until you guess the correct number. When you guess the answer, the application displays "Excellent! You guessed the number!" (Fig. 1.14).
After guessing the correct number, the application asks if you would like to play another game. At the "Would you like to play again (y or n)?" prompt, entering the one character y causes the application to choose a new number and displays the message "Please enter your first guess." followed by a question mark prompt (Fig. 1.15) so you can make your first guess in the new game. Entering the character n ends the application and returns you to the application’s directory in the shell (Fig. 1.16). Each time you execute this application from the beginning (i.e., Step 3), it will choose the same numbers for you to guess.

Excellent! You guessed the number.
Would you like to play again (y or n)?

I have a number between 1 and 1000.
Can you guess my number?
Please type your first guess.
?

Exiting the game.
1.17 Software Engineering Case Study: Introduction to Object Technology and the UML (Required)

Now we begin our early introduction to object orientation, a natural way of thinking about the world and writing computer programs. Chapters 1–7, 9 and 13 all end with a brief “Software Engineering Case Study” section in which we present a carefully paced introduction to object orientation. Our goal here is to help you develop an object-oriented way of thinking and to introduce you to the Unified Modeling Language™ (UML™)—a graphical language that allows people who design object-oriented software systems to use an industry-standard notation to represent them.

In this required section, we introduce basic object-oriented concepts and terminology. The optional sections in Chapters 2–7, 9 and 13 present an object-oriented design and implementation of the software for a simple automated teller machine (ATM) system. The “Software Engineering Case Study” sections at the ends of Chapters 2–7

- analyze a typical requirements document that describes a software system (the ATM) to be built
- determine the objects required to implement that system
- determine the attributes the objects will have
- determine the behaviors these objects will exhibit
- specify how the objects interact with one another to meet the system requirements

The “Software Engineering Case Study” sections at the ends of Chapters 9 and 13 modify and enhance the design presented in Chapters 2–7. Appendix G contains a complete, working C++ implementation of the object-oriented ATM system.

Although our case study is a scaled-down version of an industry-level problem, we nevertheless cover many common industry practices. You will experience a solid introduction to object-oriented design with the UML. Also, you will sharpen your code-reading skills by touring the complete, carefully written and well-documented C++ implementation of the ATM.

Basic Object Technology Concepts

We begin our introduction to object orientation with some key terminology. Everywhere you look in the real world you see objects—people, animals, plants, cars, planes, buildings, computers and so on. Humans think in terms of objects. Telephones, houses, traffic lights, microwave ovens and water coolers are just a few more objects we see around us every day.

We sometimes divide objects into two categories: animate and inanimate.Animate objects are “alive” in some sense—they move around and do things. Inanimate objects, on the other hand, do not move on their own. Objects of both types, however, have some things in common. They all have attributes (e.g., size, shape, color and weight), and they all exhibit behaviors (e.g., a ball rolls, bounces, inflates and deflates; a baby cries, sleeps, crawls, walks and blinks; a car accelerates, brakes and turns; a towel absorbs water). We will study the kinds of attributes and behaviors that software objects have.

Humans learn about existing objects by studying their attributes and observing their behaviors. Different objects can have similar attributes and can exhibit similar behaviors. Comparisons can be made, for example, between babies and adults and between humans and chimpanzees.

Copyright © 1992-2005 by Deitel & Associates, Inc. All Rights Reserved
Object-oriented design (OOD) models software in terms similar to those that people use to describe real-world objects. It takes advantage of class relationships, where objects of a certain class, such as a class of vehicles, have the same characteristics—cars, trucks, little red wagons and roller skates have much in common. OOD takes advantage of inheritance relationships, where new classes of objects are derived by absorbing characteristics of existing classes and adding unique characteristics of their own. An object of class “convertible” certainly has the characteristics of the more general class “automobile,” but more specifically, the roof goes up and down.

Object-oriented design provides a natural and intuitive way to view the software design process—namely, modeling objects by their attributes, behaviors and interrelationships just as we describe real-world objects. OOD also models communication between objects. Just as people send messages to one another (e.g., a sergeant commands a soldier to stand at attention), objects also communicate via messages. A bank account object may receive a message to decrease its balance by a certain amount because the customer has withdrawn that amount of money.

OOD encapsulates (i.e., wraps) attributes and operations (behaviors) into objects—an object’s attributes and operations are intimately tied together. Objects have the property of information hiding. This means that objects may know how to communicate with one another across well-defined interfaces, but normally they are not allowed to know how other objects are implemented—implementation details are hidden within the objects themselves. We can drive a car effectively, for instance, without knowing the details of how engines, transmissions, brakes and exhaust systems work internally—as long as we know how to use the accelerator pedal, the brake pedal, the steering wheel and so on. Information hiding, as we will see, is crucial to good software engineering.

Languages like C++ are object oriented. Programming in such a language is called object-oriented programming (OOP), and it allows computer programmers to implement an object-oriented design as a working software system. Languages like C, on the other hand, are procedural, so programming tends to be action oriented. In C, the unit of programming is the function. In C++, the unit of programming is the class from which objects are eventually instantiated (an OOP term for “created”). C++ classes contain functions that implement operations and data that implements attributes.

C programmers concentrate on writing functions. Programmers group actions that perform some common task into functions, and group functions to form programs. Data is certainly important in C, but the view is that data exists primarily in support of the actions that functions perform. The verbs in a system specification help the C programmer determine the set of functions that will work together to implement the system.

**Classes, Data Members and Member Functions**

C++ programmers concentrate on creating their own user-defined types called classes. Each class contains data as well as the set of functions that manipulate that data and provide services to clients (i.e., other classes or functions that use the class). The data components of a class are called data members. For example, a bank account class might include an account number and a balance. The function components of a class are called member functions (typically called methods in other object-oriented programming languages such as Java). For example, a bank account class might include member functions to make a deposit (increasing the balance), make a withdrawal (decreasing the balance) and inquire what the current balance is. The programmer uses built-in types (and other user-defined types) to define the data components of the class and member functions to implement the behavior of the class.
types) as the “building blocks” for constructing new user-defined types (classes). The nouns in a system specification help the C++ programmer determine the set of classes from which objects are created that work together to implement the system.

Classes are to objects as blueprints are to houses—a class is a “plan” for building an object of the class. Just as we can build many houses from one blueprint, we can instantiate (create) many objects from one class. You cannot cook meals in the kitchen of a blueprint; you can cook meals in the kitchen of a house. You cannot sleep in the bedroom of a blueprint; you can sleep in the bedroom of a house.

Classes can have relationships with other classes. For example, in an object-oriented design of a bank, the “bank teller” class needs to relate to other classes, such as the “customer” class, the “cash drawer” class, the “safe” class, and so on. These relationships are called associations.

Packaging software as classes makes it possible for future software systems to reuse the classes. Groups of related classes are often packaged as reusable components. Just as realtors often say that the three most important factors affecting the price of real estate are “location, location and location,” people in the software development community often say that the three most important factors affecting the future of software development are “reuse, reuse and reuse.”

Software Engineering Observation 1.4

Reuse of existing classes when building new classes and programs saves time, money and effort. Reuse also helps programmers build more reliable and effective systems, because existing classes and components often have gone through extensive testing, debugging and performance tuning.

Indeed, with object technology, you can build much of the new software you will need by combining existing classes, just as automobile manufacturers combine interchangeable parts. Each new class you create will have the potential to become a valuable software asset that you and other programmers can reuse to speed and enhance the quality of future software development efforts.

Introduction to Object-Oriented Analysis and Design (OOAD)

Soon you will be writing programs in C++. How will you create the code for your programs? Perhaps, like many beginning programmers, you will simply turn on your computer and start typing. This approach may work for small programs (like the ones we present in the early chapters of the book), but what if you were asked to create a software system to control thousands of automated teller machines for a major bank? Or what if you were asked to work on a team of 1,000 software developers building the next generation of the U.S. air traffic control system? For projects so large and complex, you could not simply sit down and start writing programs.

To create the best solutions, you should follow a detailed process for analyzing your project’s requirements (i.e., determining what the system is supposed to do) and developing a design that satisfies them (i.e., deciding how the system should do it). Ideally, you would go through this process and carefully review the design (or have your design reviewed by other software professionals) before writing any code. If this process involves analyzing and designing your system from an object-oriented point of view, it is called object-oriented analysis and design (OOAD). Experienced programmers know that analysis and design can save many hours by helping avoid an ill-planned system development...
Introduction to Object Technology and the UML (Required)

approach that has to be abandoned partway through its implementation, possibly wasting considerable time, money and effort.

OOAD is the generic term for the process of analyzing a problem and developing an approach for solving it. Small problems like the ones discussed in these first few chapters do not require an exhaustive OOAD process. It may be sufficient, before we begin writing C++ code, to write pseudocode—an informal text-based means of expressing program logic. It is not actually a programming language, but we can use it as a kind of outline to guide us as we write our code. We introduce pseudocode in Chapter 4.

As problems and the groups of people solving them increase in size, the methods of OOAD quickly become more appropriate than pseudocode. Ideally, a group should agree on a strictly defined process for solving its problem and a uniform way of communicating the results of that process to one another. Although many different OOAD processes exist, a single graphical language for communicating the results of any OOAD process has come into wide use. This language, known as the Unified Modeling Language (UML), was developed in the mid-1990s under the initial direction of three software methodologists: Grady Booch, James Rumbaugh and Ivar Jacobson.

History of the UML

In the 1980s, increasing numbers of organizations began using OOP to build their applications, and a need developed for a standard OOAD process. Many methodologists—including Booch, Rumbaugh and Jacobson—individually produced and promoted separate processes to satisfy this need. Each process had its own notation, or “language” (in the form of graphical diagrams), to convey the results of analysis and design.

By the early 1990s, different organizations, and even divisions within the same organization, were using their own unique processes and notations. At the same time, these organizations also wanted to use software tools that would support their particular processes. Software vendors found it difficult to provide tools for so many processes. Clearly, a standard notation and standard processes were needed.

In 1994, James Rumbaugh joined Grady Booch at Rational Software Corporation (now a division of IBM), and the two began working to unify their popular processes. They soon were joined by Ivar Jacobson. In 1996, the group released early versions of the UML to the software engineering community and requested feedback. Around the same time, an organization known as the Object Management Group™ (OMG™) invited submissions for a common modeling language. The OMG (www.omg.org) is a nonprofit organization that promotes the standardization of object-oriented technologies by issuing guidelines and specifications, such as the UML. Several corporations—among them HP, IBM, Microsoft, Oracle and Rational Software—had already recognized the need for a common modeling language. In response to the OMG’s request for proposals, these companies formed UML Partners—the consortium that developed the UML version 1.1 and submitted it to the OMG. The OMG accepted the proposal and, in 1997, assumed responsibility for the continuing maintenance and revision of the UML. In March 2003, the OMG released UML version 1.5. The UML version 2—which had been adopted and was in the process of being finalized at the time of this publication—marks the first major revision since the 1997 version 1.1 standard. Many books, modeling tools and industry experts are already using the UML version 2, so we present UML version 2 terminology and notation throughout this book.
What Is the UML?

The Unified Modeling Language is now the most widely used graphical representation scheme for modeling object-oriented systems. It has indeed unified the various popular notational schemes. Those who design systems use the language (in the form of diagrams) to model their systems, as we do throughout this book.

An attractive feature of the UML is its flexibility. The UML is extensible (i.e., capable of being enhanced with new features) and is independent of any particular OOAD process. UML modelers are free to use various processes in designing systems, but all developers can now express their designs with one standard set of graphical notations.

The UML is a complex, feature-rich graphical language. In our “Software Engineering Case Study” sections on developing the software for an automated teller machine (ATM), we present a simple, concise subset of these features. We then use this subset to guide you through a first design experience with the UML, intended for novice object-oriented programmers in a first- or second-semester programming course.

This case study was carefully developed under the guidance of distinguished academic and professional reviewers. We sincerely hope you enjoy working through it. If you have the slightest question, please communicate with us at deitel@deitel.com. We will respond promptly.

Internet and Web UML Resources

For more information about the UML, refer to the following Web sites. For additional UML sites, please refer to the Internet and Web resources listed at the end of Section 2.8.

www.uml.org
This UML resource page from the Object Management Group (OMG) provides specification documents for the UML and other object-oriented technologies.

www.ibm.com/software/rational/uml
This is the UML resource page for IBM Rational—the successor to the Rational Software Corporation (the company that created the UML).

Recommended Readings

Many books on the UML have been published. The following recommended books provide information about object-oriented design with the UML.


For additional books on the UML, please refer to the recommended readings listed at the end of Section 2.8, or visit www.amazon.com or www.bn.com. IBM Rational, formerly Rational Software Corporation, also provides a recommended-reading list for UML books at www.ibm.com/software/rational/info/technical/books.jsp.

Section 1.17 Self-Review Exercises

1.1 List three examples of real-world objects that we did not mention. For each object, list several attributes and behaviors.
1.2 Pseudocode is ________.
   a) another term for OOAD
   b) a programming language used to display UML diagrams
   c) an informal means of expressing program logic
   d) a graphical representation scheme for modeling object-oriented systems

1.3 The UML is used primarily to ________.
   a) test object-oriented systems
   b) design object-oriented systems
   c) implement object-oriented systems
   d) Both a and b

Answers to Section 1.17 Self-Review Exercises

1.1 [Note: Answers may vary.] a) A television’s attributes include the size of the screen, the number of colors it can display, its current channel and its current volume. A television turns on and off, changes channels, displays video and plays sounds. b) A coffee maker’s attributes include the maximum volume of water it can hold, the time required to brew a pot of coffee and the temperature of the heating plate under the coffee pot. A coffee maker turns on and off, brews coffee and heats coffee. c) A turtle’s attributes include its age, the size of its shell and its weight. A turtle walks, retreats into its shell, emerges from its shell and eats vegetation.

1.2 c.

1.3 b.

1.18 Wrap-Up

This chapter introduced basic hardware and software concepts, and explored C++’s role in developing distributed client/server applications. You studied the history of the Internet and the World Wide Web. We discussed the different types of programming languages, their history and which programming languages are most widely used. We also discussed the C++ Standard Library which contains reusable classes and functions that help C++ programmers create portable C++ programs.

We presented basic object technology concepts, including classes, objects, attributes, behaviors, encapsulation and inheritance. You also learned about the history and purpose of the UML—the industry-standard graphical language for modeling software systems.

You learned the typical steps for creating and executing a C++ application. Finally, you “test-drove” a sample C++ application similar to the types of applications you will learn to program in this book.

In the next chapter, you will create your first C++ applications. You will see several examples that demonstrate how programs display messages on the screen and obtain information from the user at the keyboard for processing. We analyze and explain each example to help you ease your way into C++ programming.

1.19 Web Resources

This section provides many Web resources that will be useful to you as you learn C++. The sites include C++ resources, C++ development tools for students and professionals and some links to fun games built with C++. This section also lists our own Web sites where you can find downloads and resources associated with this book. You will find additional Web Resources in Appendix I.
Chapter 1 Introduction to Computers, the Internet and World Wide Web

Deitel & Associates Web Sites

www.deitel.com/books/cppHTP5/index.html
The Deitel & Associates C++ How to Program, Fifth Edition site. Here you will find links to the book's examples (also included on the CD that accompanies the book) and other resources, such as our free Dive Into™ guides that help you get started with several C++ integrated development environments (IDEs).

www.deitel.com
Please check the Deitel & Associates site for updates, corrections and additional resources for all Deitel publications.

www.deitel.com/newsletter/subscribe.html
Please visit this site to subscribe for the Deitel® Buzz Online e-mail newsletter to follow the Deitel & Associates publishing program.

www.prenhall.com/deitel
Prentice Hall's site for Deitel publications. Here you will find detailed product information, sample chapters and Companion Web Sites containing book- and chapter-specific resources for students and instructors.

Compilers and Development Tools

www.thefreecountry.com/developer/c/compilers.shtml
This site lists free C and C++ compilers for a variety of operating systems.

msdn.microsoft.com/visualc
The Microsoft Visual C++ site provides product information, overviews, supplemental materials and ordering information for the Visual C++ compiler.

www.borland.com/bcppbuilder
This is a link to the Borland C++Builder. A free command-line version is available for download.

www.compilers.net
Compilers.net is designed to help users locate compilers.

developer.intel.com/software/products/compilers/cwin/index.htm
An evaluation download of the Intel C++ compiler is available at this site.

www.kai.com/C_plus_plus
This site offers the Kai C++ compiler for a 30-day free trial.

www.symbian.com/developer/development/cppdev.html
Symbian provides a C++ Developer's Pack and links to various resources, including code and development tools for C++ programmers implementing mobile applications for the Symbian operating system, which is popular on devices such as mobile phones.

Resources

www.hal9k.com/cug
The C/C++ Users Group (CUG) site contains C++ resources, journals, shareware and freeware.

www.devx.com
DevX is a comprehensive resource for programmers that provides the latest news, tools and techniques for various programming languages. The C++ Zone offers tips, discussion forums, technical help and online newsletters.

www.acm.org/crossroads/xrds3-2/ovp32.html
The Association for Computing Machinery (ACM) site offers a comprehensive listing of C++ resources, including recommended texts, journals and magazines, published standards, newsletters, FAQs and newsgroups.
1.19 Web Resources

The Association of C & C++ Users (ACCU) site contains links to C++ tutorials, articles, developer information, discussions and book reviews.

www.cuj.com
The C/C++ User's Journal is an online magazine that contains articles, tutorials and downloads. The site features news about C++, forums and links to information about development tools.

www.research.att.com/~bs/homepage.html
This is the site for Bjarne Stroustrup, designer of the C++ programming language. This site provides a list of C++ resources, FAQs and other useful C++ information.

Games

www.codearchive.com/list.php?go=0708
This site has several C++ games available for download.

www.mathtools.net/C_C__/Games/
This site includes links to numerous games built with C++. The source code for most of the games is available for download.

www.gametutorials.com/Tutorials/GT/GT_Pg1.htm
This site has tutorials on game programming in C++. Each tutorial includes a description of the game and a list of the methods and functions used in the tutorial.

www.forum.nokia.com/main/0,6566,050_20,00.html
Visit this Nokia site to learn how to use C++ to program games for some Nokia wireless devices.

Summary

- The various devices that comprise a computer system are referred to as hardware.
- The computer programs that run on a computer are referred to as software.
- A computer is capable of performing computations and making logical decisions at speeds millions (even billions) of times faster than human beings can.
- Computers process data under the control of sets of instructions called computer programs, which guide the computer through orderly sets of actions specified by computer programmers.
- The input unit is the “receiving” section of the computer. It obtains information from input devices and places it at the disposal of the other units for processing.
- The output unit is the “shipping” section of the computer. It takes information processed by the computer and places it on output devices to make it available for use outside the computer.
- The memory unit is the rapid-access, relatively low-capacity “warehouse” section of the computer. It retains information that has been entered through the input unit, making it immediately available for processing when needed, and retains information that has already been processed until it can be placed on output devices by the output unit.
- The arithmetic and logic unit (ALU) is the “manufacturing” section of the computer. It is responsible for performing calculations and making decisions.
- The central processing unit (CPU) is the “administrative” section of the computer. It coordinates and supervises the operation of the other sections.
Chapter 1 Introduction to Computers, the Internet and World Wide Web

• The secondary storage unit is the long-term, high-capacity “warehousing” section of the computer. Programs or data not being used by the other units are normally placed on secondary storage devices (e.g., disks) until they are needed, possibly hours, days, months or even years later.

• Operating systems were developed to help make it more convenient to use computers.

• Multiprogramming involves the sharing of a computer’s resources among the jobs competing for its attention, so that the jobs appear to run simultaneously.

• With distributed computing, an organization’s computing is distributed over networks to the sites where the work of the organization is performed.

• Any computer can directly understand only its own machine language, which generally consist of strings of numbers that instruct computers to perform their most elementary operations.

• English-like abbreviations form the basis of assembly languages. Translator programs called assemblers convert assembly-language programs to machine language.

• Compilers translate high-level language programs into machine-language programs. High-level languages (like C++) contain English words and conventional mathematical notations.

• Interpreter programs directly execute high-level language programs, eliminating the need to compile them into machine language.

• C++ evolved from C, which evolved from two previous programming languages, BCPL and B.

• C++ is an extension of C developed by Bjarne Stroustrup in the early 1980s at Bell Laboratories. C++ enhances the C language and provides capabilities for object-oriented programming.

• Objects are reusable software components that model items in the real world. Using a modular, object-oriented design and implementation approach can make software development groups more productive than with previous programming techniques.

• C++ programs consist of pieces called classes and functions. You can program each piece you may need to form a C++ program. However, most C++ programmers take advantage of the rich collections of existing classes and functions in the C++ Standard Library.

• Java is used to create dynamic and interactive content for Web pages, develop enterprise applications, enhance Web server functionality, provide applications for consumer devices and more.

• FORTRAN (FORmula TRANslator) was developed by IBM Corporation in the mid-1950s for scientific and engineering applications that require complex mathematical computations.

• COBOL (COmmon Business Oriented Language) was developed in the late 1950s by a group of computer manufacturers and government and industrial computer users. COBOL is used primarily for commercial applications that require precise and efficient data manipulation.

• Ada was developed under the sponsorship of the United States Department of Defense (DOD) during the 1970s and early 1980s. Ada provides multitasking, which allows programmers to specify that many activities are to occur in parallel.

• The BASIC (Beginner’s All-Purpose Symbolic Instruction Code) programming language was developed in the mid-1960s at Dartmouth College as a language for writing simple programs. BASIC’s primary purpose was to familiarize novices with programming techniques.

• Microsoft’s Visual Basic was introduced in the early 1990s to simplify the process of developing Microsoft Windows applications.

• Microsoft has a corporate-wide strategy for integrating the Internet and the Web into computer applications. This strategy is implemented in Microsoft’s .NET platform.

• The .NET platform’s three primary programming languages are Visual Basic .NET (based on the original BASIC), Visual C++ .NET (based on C++) and C# (a new language based on C++ and Java that was developed expressly for the .NET platform).

Copyright © 1992-2005 by Deitel & Associates, Inc. All Rights Reserved
.NET developers can write software components in their preferred language, then form applications by combining those components with components written in any .NET language.

C++ systems generally consist of three parts: a program development environment, the language and the C++ Standard Library.

C++ programs typically go through six phases: edit, preprocess, compile, link, load and execute.

C++ source code file names often end with the .cpp, .cxx, .cc or .C extensions.

A preprocessor program executes automatically before the compiler’s translation phase begins. The C++ preprocessor obeys commands called preprocessor directives, which indicate that certain manipulations are to be performed on the program before compilation.

The object code produced by the C++ compiler typically contains “holes” due to references to functions and data defined elsewhere. A linker links the object code with the code for the missing functions to produce an executable image (with no missing pieces).

The loader takes the executable image from disk and transfers it to memory for execution.

Most programs in C++ input and/or output data. Data is often input from cin (the standard input stream) which is normally the keyboard, but cin can be redirected from another device. Data is often output to cout (the standard output stream), which is normally the computer screen, but cout can be redirected to another device. The cerr stream is used to display error messages.

The Unified Modeling Language (UML) is a graphical language that allows people who build systems to represent their object-oriented designs in a common notation.

Object-oriented design (OOD) models software components in terms of real-world objects. It takes advantage of class relationships, where objects of a certain class have the same characteristics. It also takes advantage of inheritance relationships, where newly created classes of objects are derived by absorbing characteristics of existing classes and adding unique characteristics of their own. OOD encapsulates data (attributes) and functions (behavior) into objects—the data and functions of an object are intimately tied together.

Objects have the property of information hiding—objects normally are not allowed to know how other objects are implemented.

Object-oriented programming (OOP) allows programmers to implement object-oriented designs as working systems.

C++ programmers create their own user-defined types called classes. Each class contains data (known as data members) and the set of functions (known as member functions) that manipulate that data and provide services to clients.

Classes can have relationships with other classes. These relationships are called associations.

Packaging software as classes makes it possible for future software systems to reuse the classes. Groups of related classes are often packaged as reusable components.

An instance of a class is called an object.

With object technology, programmers can build much of the software they will need by combining standardized, interchangeable parts called classes.

The process of analyzing and designing a system from an object-oriented point of view is called object-oriented analysis and design (OOAD).
Chapter 1  Introduction to Computers, the Internet and World Wide Web

arithmetic and logic unit (ALU)
assembler
assembly language
association
attribute of an object
BASIC (Beginner’s All-Purpose Instruction Code)
batch processing
behavior of an object
Booch, Grady
C
C++
C++ Standard Library
C#
central processing unit (CPU)
class
client
client/server computing
COBOL (COmmon Business Oriented Language)
compile phase
compiler
component
computer
computer program
computer programmer
core memory
data
data member
debug
decision
design
distributed computing
dynamic content
deploy phase
deployer
deployable
executable image
execute phase
executable
extensible
file server
FORTRAN (FORmula TRANslator)
function
hardware
hardware platform
high-level language
information hiding
inheritance
input device
input unit
input/output (I/O)
instanitate
interface
International Organization for Standardization (ISO)
Internet
interpreter
Jacobson, Ivar
Java
link phase
linker
live-code approach
load phase
loader
local area networks (LANs)
logical unit
machine dependent
machine independent
machine language
member function
memory
memory unit
method
MFC (Microsoft Foundation Classes)
Microsoft’s .NET Framework Class Library
multiprocessor
multiprogramming
multitasking
multithreading
.NET platform
.object
.object code
Object Management Group (OMG)
object-oriented analysis and design (OOAD)
object-oriented design (OOD)
object-oriented programming (OOP)
operating system
.operation
.output device
.output unit
.personal computing
platform
.portable
.preprocess phase
.preprocessor directives
.primary memory
.procedural programming
.pseudocode
.Rational Software Corporation
.requirements document

Copyright © 1992-2005 by Deitel & Associates, Inc. All Rights Reserved
Self-Review Exercises

Rumbaugh, James
runtime errors or execution-time errors
secondary storage unit
software
software reuse
source code
structured programming
structured systems analysis and design
supercomputer
task
throughput
timesharing
translation
translator program
Unified Modeling Language (UML)
user-defined type
Visual Basic .NET
Visual C++ .NET
workstation
World Wide Web

Self-Review Exercises

1.1 Fill in the blanks in each of the following:

a) The company that popularized personal computing was ________

b) The computer that made personal computing legitimate in business and industry was ________

c) Computers process data under the control of sets of instructions called computer ________

d) The six key logical units of the computer are the ________, ________, ________, ________, ________, and the ________

e) The three classes of languages discussed in the chapter are ________, ________, and ________

f) The programs that translate high-level language programs into machine language are called ________

g) C is widely known as the development language of the ________ operating system.

h) The ________ language was developed by Wirth for teaching structured programming.

i) The Department of Defense developed the Ada language with a capability called ________, which allows programmers to specify that many activities can proceed in parallel.

1.2 Fill in the blanks in each of the following sentences about the C++ environment.

a) C++ programs are normally typed into a computer using a(n) ________ program.

b) In a C++ system, a(n) ________ program executes before the compiler’s translation phase begins.

c) The ________ program combines the output of the compiler with various library functions to produce an executable image.

d) The ________ program transfers the executable image of a C++ program from disk to memory.

1.3 Fill in the blanks in each of the following statements (based on Section 1.17):

a) Objects have the property of ________—although objects may know how to communicate with one another across well-defined interfaces, they normally are not allowed to know how other objects are implemented.

b) C++ programmers concentrate on creating ________, which contain data members and the member functions that manipulate those data members and provide services to clients.

c) Classes can have relationships with other classes. These relationships are called ________.

d) The process of analyzing and designing a system from an object-oriented point of view is called ________.
Chapter 1 Introduction to Computers, the Internet and World Wide Web

c) OOD also takes advantage of _______ relationships, where new classes of objects are derived by absorbing characteristics of existing classes, then adding unique characteristics of their own.
f) _______ is a graphical language that allows people who design software systems to use an industry-standard notation to represent them.
g) The size, shape, color and weight of an object are considered _______ of the object.

Answers to Self-Review Exercises
1.1 a) Apple. b) IBM Personal Computer. c) programs. d) input unit, output unit, memory unit, arithmetic and logic unit, central processing unit, secondary storage unit. e) machine languages, assembly languages and high-level languages. f) compilers. g) UNIX. h) Pascal. i) multitasking.
1.2 a) editor. b) preprocessor. c) linker. d) loader.
1.3 a) information hiding. b) classes. c) associations. d) object-oriented analysis and design (OOAD). e) inheritance. f) The Unified Modeling Language (UML). g) attributes.

Exercises
1.4 Categorize each of the following items as either hardware or software:
   a) CPU
   b) C++ compiler
   c) ALU
   d) C++ preprocessor
   e) input unit
   f) an editor program

1.5 Why might you want to write a program in a machine-independent language instead of a machine-dependent language? Why might a machine-dependent language be more appropriate for writing certain types of programs?

1.6 Fill in the blanks in each of the following statements:
   a) Which logical unit of the computer receives information from outside the computer for use by the computer? __________.
   b) The process of instructing the computer to solve specific problems is called __________.
   c) What type of computer language uses English-like abbreviations for machine-language instructions? __________.
   d) Which logical unit of the computer sends information that has already been processed by the computer to various devices so that the information may be used outside the computer? __________.
   e) Which logical unit of the computer retains information? __________.
   f) Which logical unit of the computer performs calculations? __________.
   g) Which logical unit of the computer makes logical decisions? __________.
   h) The level of computer language most convenient to the programmer for writing programs quickly and easily is __________.
   i) The only language that a computer directly understands is called that computer’s __________.
   j) Which logical unit of the computer coordinates the activities of all the other logical units? __________.

1.7 Why is so much attention today focused on object-oriented programming in general and C++ in particular?
1.8 Distinguish between the terms fatal error and nonfatal error. Why might you prefer to experience a fatal error rather than a nonfatal error?

1.9 Give a brief answer to each of the following questions:
   a) Why does this text discuss structured programming in addition to object-oriented programming?
   b) What are the typical steps (mentioned in the text) of an object-oriented design process?
   c) What kinds of messages do people send to one another?
   d) Objects send messages to one another across well-defined interfaces. What interfaces does a car radio (object) present to its user (a person object)?

1.10 You are probably wearing on your wrist one of the world’s most common types of objects—a watch. Discuss how each of the following terms and concepts applies to the notion of a watch: object, attributes, behaviors, class, inheritance (consider, for example, an alarm clock), abstraction, modeling, messages, encapsulation, interface, information hiding, data members and member functions.