OBJECTIVES

In this chapter, you’ll learn:

- Basic computer concepts.
- The different types of programming languages.
- The history of the C programming language.
- The purpose of the C Standard Library.
- The elements of a typical C program development environment.
- How C provides a foundation for further study of programming languages in general and of C++, Java and C# in particular.
- The history of the Internet and the World Wide Web.
1.1 Introduction
1.2 Computers: Hardware and Software
1.3 Computer Organization
1.4 Personal, Distributed and Client/Server Computing
1.5 The Internet and the World Wide Web
1.6 Machine Languages, Assembly Languages and High-Level Languages
1.7 History of C
1.8 C Standard Library
1.9 C++
1.10 Java
1.11 Fortran, COBOL, Pascal and Ada
1.12 BASIC, Visual Basic, Visual C++, C# and .NET
1.13 Key Software Trend: Object Technology
1.15 Typical C Program Development Environment
1.15 Hardware Trends
1.16 Notes About C and This Book
1.17 Web Resources
1.1 Introduction

- The core of the book emphasizes achieving program clarity through the proven techniques of structured programming.
- You’ll learn programming the right way from the beginning.
- We present hundreds of complete working programs and shows the outputs produced when those programs are run on a computer.
- We call this the “live-code approach.” All of these example programs may be downloaded from our website www.deitel.com/books/chtp6/.
- It’s software (i.e., the instructions you write to command computers to perform actions and make decisions) that controls computers (often referred to as hardware).
This text introduces programming in C, which was standardized in 1989 as ANSI X3.159-1989 in the United States through the American National Standards Institute (ANSI), then worldwide through the efforts of the International Standards Organization (ISO).

We call this Standard C.

We also introduce C99 (ISO/IEC 9899:1999)—the latest version of the C standard.

A new C standard, which has been informally named C1X, is under development and likely to be published around 2012.

Optional Appendix E presents the Allegro game programming C library.
1.1 Introduction (Cont.)

- The appendix shows how to use Allegro to create a simple game.
- We show how to display graphics and smoothly animate objects, and we explain additional features such as sound, keyboard input and text output.
- The appendix includes web links and resources that point you to over 1000 Allegro games and to tutorials on advanced Allegro techniques.
- Computing costs have decreased dramatically due to rapid developments in both hardware and software technologies.
The large computers introduced decades ago were called **mainframes** and current versions are widely used today in business, government and industry.

Silicon chip technology has made computing so economical that more than a billion general-purpose computers are in use worldwide.

Billions more special purpose computers are used in intelligent electronic devices like car navigation systems, energy-saving appliances and game controllers.
1.1 Introduction (Cont.)

- C++, an object-oriented programming language based on C, is of such interest today that we’ve included a detailed introduction to C++ and object-oriented programming in the later chapters.
- To keep up to date with C and C++ developments at Deitel & Associates, register for the Deitel® Buzz Online, at
  - www.deitel.com/newsletter/subscribe.html
- Check out the growing list of C and related Resource Centers at
  - www.deitel.com/ResourceCenters.html
Some Resource Centers that will be valuable to you as you read the C portion of this book are C, Code Search Engines and Code Sites, Computer Game Programming and Programming Projects.

Errata and updates for this book are posted at

- [www.deitel.com/books/cht-p6/](http://www.deitel.com/books/cht-p6/)
1.2 Computers: Hardware and Software

- A **computer** is a device that can perform computations and make logical decisions billions of times faster than human beings can.
- Many of today’s personal computers can perform several billion additions per second.
- Today’s fastest **supercomputers** can perform thousands of trillions (quadrillions) of instructions per second!
- To put that in perspective, a quadrillion-instruction-per-second computer can perform more than 100,000 calculations per second for every person on the planet!
Computers process data under the control of sets of instructions called computer programs.

These programs guide the computer through orderly sets of actions specified by people called computer programmers.

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’ll learn proven methods that are reducing software development costs—structured programming (in the C chapters) and object-oriented programming (in the C++ chapters).
Every computer may be envisioned as divided into six logical units or sections:

- **Input unit.** This “receiving” section obtains information (data and computer programs) from input devices and places it at the disposal of the other units so that it can be processed. Humans typically enter information 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 barcodes, reading from secondary storage devices (like hard drives, CD drives, DVD drives and USB drives—also called “thumb drives”) and having your computer receive information from the Internet (such as when you download videos from YouTube™, e-books from Amazon and the like).
1.3 Computer Organization (cont.)

- **Output unit.** This “shipping” section takes information that the computer has processed and places it on various output devices to make it available for use outside the computer. Most information that is output from computers today is displayed on screens, printed on paper, played on audio players (such as Apple’s popular iPods), or used to control other devices. Computers also can output their information to networks, such as the Internet.
Memory unit. This rapid-access, relatively low-capacity “warehouse” section retains information that has been entered through the input unit, making it 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 volatile—it’s typically lost when the computer’s power is turned off. The memory unit is often called either memory or primary memory.
1.3 Computer Organization (cont.)

- **Arithmetic and logic unit (ALU).** This “manufacturing” section performs calculations, such as addition, subtraction, multiplication and division. It also contains the decision mechanisms that allow the computer, for example, to compare two items from the memory unit to determine whether they’re equal. In today’s systems, the ALU is usually implemented as part of the next logical unit, the CPU.
Central processing unit (CPU). This "administrative" section coordinates and supervises the operation of the other sections. The CPU tells the input unit when to read information 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. A multi-core processor implements multiprocessing on a single integrated circuit chip—for example a dual-core processor has two CPUs and a quad-core processor has four CPUs.
1.3 Computer Organization (cont.)

- **Secondary storage unit.** This is the long-term, high-capacity “warehousing” section. Programs or data not actively being used by the other units normally are placed on secondary storage devices (e.g., your hard drive) until they’re again needed, possibly hours, days, months or even years later. Therefore, information on secondary storage devices is said to be **persistent**—it is preserved even when the computer’s power is turned off. Secondary storage information 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. Examples of secondary storage devices include CDs, DVDs and flash drives (sometimes called memory sticks), which can hold hundreds of millions to billions of characters.
1.4 Personal, Distributed and Client/Server Computing

- In 1977, Apple Computer popularized personal computing.
- In 1981, IBM, the world’s largest computer vendor, introduced the IBM Personal Computer (PC).
- This quickly legitimized personal computing in business, industry and government organizations, where IBM mainframes were heavily used.
- These computers were “stand-alone” units—people transported disks back and forth between them to share information (this was often called “sneakernet”).
- 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 **distributed computing**.

Information is shared easily across computer networks, where computers called **servers** (file servers, database servers, web servers, etc.) offer capabilities that may be used by **client** computers distributed throughout the network, hence the term **client/server computing**.

**C** is widely used for writing software for operating systems, for computer networking and for distributed client/server applications.
1.5 The Internet and the World Wide Web

- 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 the world’s premier communication mechanism.

- Today’s applications can be written to communicate among the world’s computers.
Programmers write instructions in various programming languages, some directly understandable by computers and others requiring intermediate translation steps.

Computer languages may be divided into three general types:
- Machine languages
- Assembly languages
- 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.
Machine language is often referred to as object code.

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).
1.6 Machine Languages, Assembly Languages and High-Level Languages

- 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

- 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.
1.6 Machine Languages, Assembly Languages and High-Level 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’s incomprehensible to computers until translated to machine language.
1.6 Machine Languages, Assembly Languages and High-Level Languages

- 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

```markdown
grossPay = basePay + overtimePay;
```

C, C++, Microsoft’s .NET languages (e.g., Visual Basic, Visual C++ and Visual C#) and Java are among the most widely used high-level programming languages.

Interpreter programs were developed to execute high-level language programs directly (without the delay of compilation), although slower than compiled programs run.
1.7 History of C

- C evolved from two previous languages, BCPL and B.
- BCPL was developed in 1967 by Martin Richards as a language for writing operating-systems software and compilers.
- Ken Thompson modeled many features in his B language after their counterparts in BCPL, and in 1970 he used B to create early versions of the UNIX operating system at Bell Laboratories.
- Both BCPL and B were “typeless” languages—every data item occupied one “word” in memory, and the burden of typing variables fell on the shoulders of the programmer.
1.7 History of C (Cont.)

- The C language was evolved from B by Dennis Ritchie at Bell Laboratories and was originally implemented on a DEC PDP-11 computer in 1972.
- C initially became widely known as the development language of the UNIX operating system.
- Today, virtually all new major operating systems are written in C and/or C++.
- C is available for most computers.
- C is mostly hardware independent.
- With careful design, it’s possible to write C programs that are portable to most computers.
By the late 1970s, C had evolved into what is now referred to as “traditional C.” The publication in 1978 of Kernighan and Ritchie’s book, *The C Programming Language*, drew wide attention to the language.

The rapid expansion of C over various types of computers (sometimes called hardware platforms) led to many variations that were similar but often incompatible.

In 1989, the C standard was approved; this standard was updated in 1999.

C99 is a revised standard for the C programming language that refines and expands the capabilities of C.
Not all popular C compilers support C99.

Of those that do, most implement only a subset of the new features.

Chapters 1–14 of this book are based on the widely adopted international Standard (ANSI/ISO) C.

Appendix G introduces C99 and provides links to popular C99 compilers and IDEs. The appendix ties the C99 features back to the earlier sections of the book for classes that cover C99.
**Portability Tip 1.1**

Because C is a hardware-independent, widely available language, applications written in C can run with little or no modifications on a wide range of different computer systems.
1.8 C Standard Library

- As you’ll learn in Chapter 5, C programs consist of modules or pieces called **functions**.
- You can program all the functions you need to form a C program, but most C programmers take advantage of a rich collection of existing functions called the **C Standard Library**.
- Visit the following website for the complete C Standard Library documentation, including the C99 features:
- This textbook encourages a **building-block approach** to creating programs.
Avoid reinventing the wheel.

Instead, use existing pieces—this is called **software reusability**, and it’s a key to the field of object-oriented programming, as you’ll see when you study C++.

When programming in C you’ll typically use the following building blocks:
- C Standard Library functions
- Functions you create yourself
- Functions other people have created and made available to you
If you use existing functions, you can avoid reinventing the wheel.

In the case of the Standard C functions, you know that they’re carefully written, and you know that because you’re using functions that are available on all Standard C implementations, your programs will have a greater chance of being portable and error-free.
Performance Tip 1.1

Using Standard C library functions instead of writing your own comparable versions can improve program performance, because these functions are carefully written to perform efficiently.
Performance Tip 1.2

Using Standard C library functions instead of writing your own comparable versions can improve program portability, because these functions are used in virtually all Standard C implementations.
1.9 C++

- C++ was developed by Bjarne Stroustrup at Bell Laboratories.
- It has its roots in C, providing a number of features that “spruce up” the C language.
- More important, it provides capabilities for object-oriented programming.
- Objects are essentially reusable software components that model items in the real world.
- Using a modular, object-oriented design and implementation approach can make software development groups much more productive than is possible with previous programming techniques.
In the later chapters of *C How to Program, 6/e*, we present a condensed treatment of C++ selected from our book *C++ How to Program, 7/e*.

1.10 Java

- Microprocessors are having a profound impact in intelligent consumer electronic devices.
- Recognizing this, Sun Microsystems developed a C++-based language that it eventually called Java.
- 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.
- Java garnered the attention of the business community because of the phenomenal interest in 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 (e.g., cell phones, pagers and personal digital assistants) and for many other purposes.
Hundreds of high-level languages have been developed, but 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 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.
• People realized that software development was a 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 and easier to test, debug and 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 lacked many features needed in commercial, industrial and government applications, so it was not widely accepted outside academia.
The Ada 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 one language that would fill most of its needs.
1.11 Fortran, COBOL, Pascal and Ada (Cont.)

- 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.
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 the capabilities developers 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 (based on the original BASIC), Visual C++ (based on C++) and Visual C# (a new language based on C++ and Java that was developed expressly for the .NET platform).

Visual C++ can also be used to compile and run C programs.
The C++ programming language, developed at AT&T by Bjarne Stroustrup in the early 1980s, is based on two languages—C and Simula 67, a simulation programming language developed at the Norwegian Computing Center and released in 1967.

C++ absorbed the features of C and added Simula’s capabilities for creating and manipulating objects.

Object technology is a packaging scheme that helps us create meaningful software units.

There are date objects, time objects, paycheck objects, invoice objects, audio objects, video objects, file objects, record objects and so on.
1.14 Key Software Trend: Object Technology (Cont.)

- In fact, almost any noun can be reasonably represented as an object.
- We live in a world of objects.
- There are cars, planes, people, animals, buildings, traffic lights, elevators and the like.
- Before object-oriented languages appeared, procedural programming languages (such as Fortran, COBOL, Pascal, BASIC and C) were focused 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++, Java and C#, 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 gains.

A key problem with procedural programming is that the program units do not effectively mirror real-world entities, so these units are not particularly reusable.
It isn’t 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 reusable on future projects.

Using libraries of reusable componentry can greatly reduce effort required to implement certain kinds of systems.
Software Engineering Observation 1.1

Extensive class libraries of reusable software components are available on the Internet. Many of these libraries are free.
1.14 Key Software Trend: Object Technology (Cont.)

- Some organizations report that the key benefit of object-oriented programming is not software reuse but, rather, that the software they produce is more understandable, better organized and easier to maintain, modify and debug.

- This can be significant, because perhaps 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, it’s clear that object-oriented programming will be the key programming methodology for the next several decades.
C systems generally consist of several parts: a program development environment, the language and the C Standard Library.

C programs typically go through six phases to be executed (Fig. 1.1).

These are: **edit, preprocess, compile, link, load** and **execute**.

Although *C How to Program, 6/e* is a generic C textbook (written independently of the details of any particular operating system), we concentrate in this section on a typical Linux-based C system.
1.15 Typical C Program Development Environment (Cont.)

• [Note: The programs in this book will run with little or no modification on most current C systems, including Microsoft Windows-based systems.] If you’re not using a Linux system, refer to the manuals for your system or ask your instructor how to accomplish these tasks in your environment.

• Check out our C Resource Center at www.deitel.com to locate “getting started” tutorials for popular C compilers and development environments.

• Phase 1 consists of editing a file.

• This is accomplished with an editor program.
Two editors widely used on Linux systems are *vi* and *emacs*.

Software packages for the C/C++ integrated program development environments such as Eclipse and Microsoft Visual Studio have editors that are integrated into the programming environment.

You type a C program with the editor, make corrections if necessary, then store the program on a secondary storage device such as a hard disk.

C program file names should end with the `.c` extension.
In Phase 2, you give the command to compile the program.

The compiler translates the C program into machine language-code (also referred to as object code).

In a C system, a preprocessor program executes automatically before the compiler’s translation phase begins.

The C preprocessor obeys special commands called preprocessor directives, which indicate that certain manipulations are to be performed on the program before compilation.
These manipulations usually consist of including other files in the file to be compiled and performing various text replacements.

The most common preprocessor directives are discussed in the early chapters; a detailed discussion of preprocessor features appears in Chapter 13.

In Phase 3, the compiler translates the C program into machine-language code.
Fig. 1.1 | Typical C development environment. (Part 1 of 2.)
Fig. 1.1 | Typical C development environment. (Part 2 of 2.)
The next phase is called **linking**.

C programs typically contain references to functions 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).

On a typical Linux system, the command to compile and link a program is called **cc** (or **gcc**).
To compile and link a program named `welcome.c` type
- `cc welcome.c`

at the Linux prompt and press the `Enter` key (or `Return` key).

[Note: Linux commands are case sensitive; make sure that you type lowercase C’s and that the letters in the filename are in the appropriate case.]

If the program compiles and links correctly, a file called `a.out` is produced.

This is the executable image of our `welcome.c` program.
1.15 Typical C Program Development Environment (Cont.)

- The next phase is called **loading**.
- Before a program can be executed, the program 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.
- Finally, the computer, under the control of its CPU, **executes** the program one instruction at a time.
To load and execute the program on a Linux system, type `. / a. out` at the Linux prompt and press `Enter`.

Programs do not always work on the first try.

Each of the preceding phases can fail because of various errors that we’ll discuss.

For example, an executing program might attempt to divide by zero (an illegal operation on computers just as in arithmetic).

This would cause the computer to display an error message.
1.15 Typical C Program Development Environment (Cont.)

- You would then return to the edit phase, make the necessary corrections and proceed through the remaining phases again to determine that the corrections work properly.
- Most C programs input and/or output data.
- Certain C functions take their input from stdin (the standard input stream), which is normally the keyboard, but stdin can be connected to another stream.
- Data is often output to stdout (the standard output stream), which is normally the computer screen, but stdout can be connected to another stream.
- 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 devices such as disks and printers.

There is also a **standard error stream** referred to as **stderr**.

The **stderr** stream (normally connected to the screen) is used for displaying error messages.

It’s common to route regular output data, i.e., **stdout**, to a device other than the screen while keeping **stderr** assigned to the screen so that the user can be immediately informed of errors.
Common Programming Error 1.1

Errors like division-by-zero occur as a program runs, so these errors are called runtime errors or execution-time errors. Divide-by-zero is generally a fatal error, i.e., an error that causes the program to terminate immediately without successfully performing its job. Nonfatal errors allow programs to run to completion, often producing incorrect results.
Every year or two, the capacities of computers have approximately doubled without any increase in price. This often is called Moore’s Law, named after the person who first identified and explained the trend, Gordon Moore, cofounder of Intel—the company that manufactures the vast majority of the processors in today’s personal computers.
1.16 Hardware Trends (Cont.)

- Moore’s Law and similar trends are especially true in relation to the amount of memory that computers have for programs, the amount of secondary storage (such as disk storage) they have to hold programs and data over longer periods of time, and their processor speeds—the speeds at which computers execute programs (i.e., do their work).
- Similar growth has occurred in the communications field, in which costs have plummeted as soaring demand for communications bandwidth has attracted intense competition.
1.17 Notes About C and This Book

- 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 may have heard that C is a portable language and that programs written in C can run on many different computers.

*Portability is an elusive goal.*
Portability Tip 1.2

Although it’s possible to write portable C programs, there are many problems between different C compilers and different computers that make portability difficult to achieve. Simply writing programs in C does not guarantee portability. You’ll often need to deal directly with computer variations.
1.17 Notes About C and This Book (Cont.)

- C is a rich language, and there are some subtleties in the language and some advanced subjects we have not covered.

- For additional technical details on C, read the C Standard document itself or the book by Kernighan and Ritchie *(The C Programming Language, Second Edition).*
Software Engineering Observation 1.2

Read the manuals for the version of C you’re using. Reference these manuals frequently to be sure you’re aware of the rich collection of C features and that you’re using these features correctly.
Software Engineering Observation 1.3

Your computer and compiler are good teachers. If you’re not sure how a C feature works, write a program with that feature, compile and run the program and see what happens.