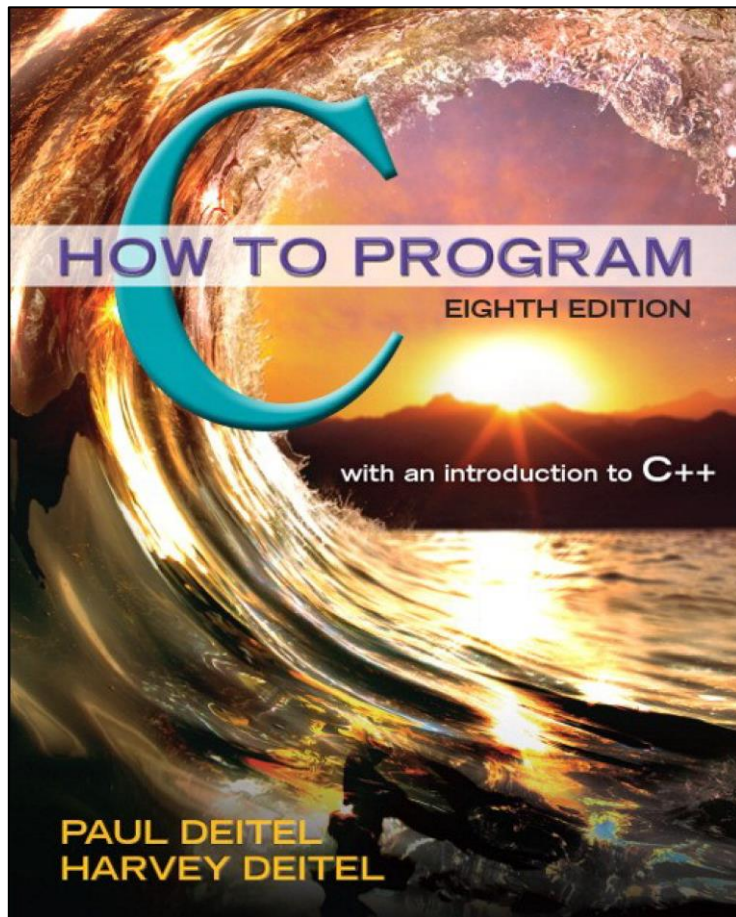


C How to Program

Eighth Edition



Chapter 1

Introduction to Computers,
the Internet and the Web

Learning Objectives

- Basic computer concepts.
- The different types of programming languages.
- The history of the C programming language.
- The purpose of the C Standard Library.
- The basics of object technology.
- A typical C program development environment.
- To test-drive a C application in Windows, Linux and Mac OS X.
- Some basics of the Internet and the World Wide Web.

Outline (1 of 5)

1.1 Introduction

1.2 Hardware and Software

1.2.1 Moore's Law

1.2.2 Computer Organization

1.3 Data Hierarchy

1.4 Machine Languages, Assembly Languages and High-Level Languages

1.5 The C Programming Language

1.6 C Standard Library

1.7 C++ and Other C-Based Languages

Outline (2 of 5)

1.8 Object Technology

- 1.8.1 The Automobile as an Object

- 1.8.2 Methods and Classes

- 1.8.3 Instantiation

- 1.8.4 Reuse

- 1.8.5 Messages and Method Calls

- 1.8.6 Attributes and Instance Variables

- 1.8.7 Encapsulation and Information Hiding

- 1.8.8 Inheritance

Outline (3 of 5)

1.9 Typical C Program-Development Environment

1.9.1 Phase 1: Creating a Program

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

1.9.3 Phase 4: Linking

1.9.4 Phase 5: Loading

1.9.5 Phase 6: Execution

1.9.6 Problems That May Occur at Execution Time

1.9.7 Standard Input, Standard Output and Standard Error Streams

Outline (4 of 5)

1.10 Test-Driving a C Application in Windows, Linux and Mac OS X

1.10.1 Running a C Application from the Windows Command Prompt

1.10.2 Running a C Application Using GNU C with Linux

1.10.3 Running a C Application Using the Terminal on Mac OS X

1.11 Operating Systems

1.11.1 Windows—A Proprietary Operating System

1.11.2 Linux—An Open-Source Operating System

1.11.3 Apple's Mac OS X; Apple's iOS for iPhone®, iPad® and iPod Touch®
Devices

1.11.4 Google's Android

Outline (5 of 5)

1.12 The Internet and World Wide Web

1.12.1 The Internet: A Network of Networks

1.12.2 The World Wide Web: Making the Internet User-Friendly

1.12.3 Web Services

1.12.4 Ajax

1.12.5 The Internet of Things

1.13 Some Key Software Terminology

1.14 Keeping Up-to-Date with Information Technologies

1.1 Introduction

- The core of the book emphasizes effective software engineering through the proven methodologies of structured programming in C and object-oriented programming in C++.
- All of these example programs may be downloaded from our website www.deitel.com/books/chtp7/.
- You'll learn how to command computers to perform tasks.
 - **Software** (i.e., the instructions you write to command computers to perform **actions** and make **decisions**) controls computers (often referred to as **hardware**).

1.2 Hardware and Software (1 of 3)

- Computers can perform calculations and make logical decisions phenomenally faster than human beings can.
- Today's personal computers can perform billions of calculations in one second—more than a human can perform in a lifetime.
- **Supercomputers** are already performing **thousands of trillions (quadrillions)** of instructions per second!
- Computers process data under the control of sequences of instructions called **computer programs**.
- These programs guide the computer through ordered actions specified by people called computer **programmers**.

1.2 Hardware and Software (2 of 3)

- The programs that run on a computer are referred to as **software**.
- A computer consists of various devices referred to as **hardware**
 - (e.g., the keyboard, screen, mouse, hard disks, memory, DVD drives and processing units).
- Computing costs are **dropping dramatically**, owing to rapid developments in hardware and software technologies.

1.2 Hardware and Software (3 of 3)

- Computers that might have filled large rooms and cost millions of dollars decades ago are now inscribed on silicon chips smaller than a fingernail, costing perhaps a few dollars each.
- Silicon-chip technology has made computing so economical that computers have become commodity.

1.2.1 Moore's Law (1 of 3)

- For many decades, hardware costs have fallen rapidly.
- Every year or two, the capacities of computers have approximately **doubled** inexpensively.
- This trend often is called **Moore's Law**, named for the person who identified it in the 1960s, Gordon Moore, co-founder of Intel.

1.2.1 Moore's Law (2 of 3)

- Moore's Law and related observations apply especially 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 their programs (i.e., do their work).
- Similar growth has occurred in the communications field.

1.2.1 Moore's Law (3 of 3)

- Costs have plummeted as enormous demand for communications bandwidth (i.e., information-carrying capacity) has attracted intense competition.
- Such phenomenal improvement is fostering the **Information Revolution.**

1.2.2 Computer Organization

- Regardless of differences in physical appearance, computers can be envisioned as divided into various **logical units** or sections (Figure 1.1)

Figure 1.1 Logical Units of a Computer (1 of 5)

Logical unit	Description
Input unit	<p>This “receiving” section obtains information (data and computer programs) from input devices and places it at the disposal of the other units for processing. Most user input is entered into computers through keyboards, touch screens and mouse devices. Other forms of input include receiving voice commands, scanning images and barcodes, reading from secondary storage devices (such as hard drives, DVD drives, Blu-ray Disc™ drives and USB flash drives—also called “thumb drives” or “memory sticks”), receiving video from a webcam and having your computer receive information from the Internet (such as when you stream videos from YouTube® or download e-books from Amazon). Newer forms of input include position data from a GPS device, and motion and orientation information from an accelerometer (a device that responds to up/down, left/right and forward/backward acceleration) in a smartphone or game controller (such as Microsoft® Kinect® for Xbox®, Wii™ Remote and Sony® PlayStation® Move).</p>

Figure 1.1 Logical Units of a Computer (2 of 5)

Logical unit	Description
Output unit	<p>This “shipping” section takes information the computer has processed and places it on various output devices to make it available for use outside the computer. Most information that’s output from computers today is displayed on screens (including touch screens), printed on paper (“going green” discourages this), played as audio or video on PCs and media players (such as Apple’s iPods) and giant screens in sports stadiums, transmitted over the Internet or used to control other devices, such as robots and “intelligent” appliances. Information is also commonly output to secondary storage devices, such as hard drives, DVD drives and USB flash drives. Popular recent forms of output are smartphone and game controller vibration, and virtual reality devices like Oculus Rift.</p>

Figure 1.1 Logical Units of a Computer (3 of 5)

Logical unit	Description
Memory unit	<p>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, primary memory or RAM (Random Access Memory). Main memories on desktop and notebook computers contain as much as 128 GB of RAM, though 2 to 16 GB is most common. GB stands for gigabytes; a gigabyte is approximately one billion bytes. A byte is eight bits. A bit is either a 0 or a 1.</p>

Figure 1.1 Logical Units of a Computer (4 of 5)

Logical unit	Description
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 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 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. A multi-core processor implements multiple processors on a single integrated-circuit chip—a dual-core processor has two CPUs and a quadcore processor has four CPUs. Today’s desktop computers have processors that can execute billions of instructions per second.

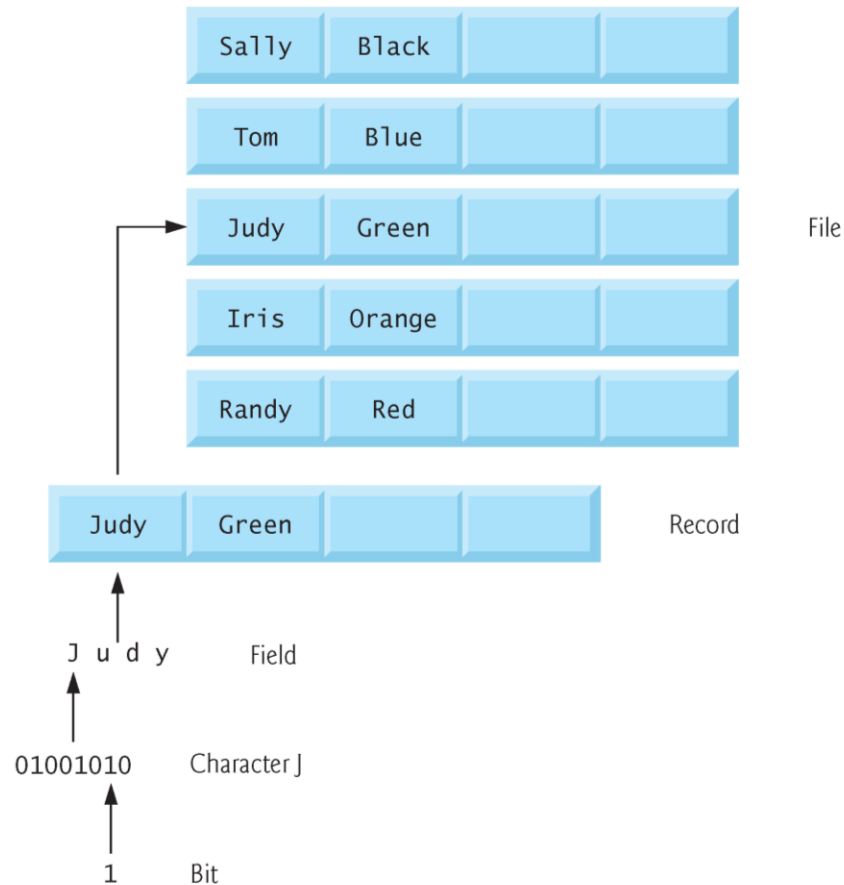
Figure 1.1 Logical Units of a Computer (5 of 5)

Logical unit	Description
Secondary storage unit	<p>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. Information on secondary storage devices is persistent—it’s preserved even when the computer’s power is turned off. Secondary storage information takes much longer to access than information in primary memory, but its cost per unit is much less. Examples of secondary storage devices include hard drives, DVD drives and USB flash drives, some of which can hold over 2 TB (TB stands for terabytes; a terabyte is approximately one trillion bytes). Typical hard drives on desktop and notebook computers hold up to 2 TB, and some desktop hard drives can hold up to 6 TB.</p>

1.3 Data Hierarchy (1 of 6)

- Data items processed by computers form a **data hierarchy** that becomes larger and more complex in structure as we progress from bits to characters to fields, and so on.

Figure 1.2 Data Hierarchy



1.3 Data Hierarchy (2 of 6)

- Bits
 - The smallest data item in a computer can assume the value 0 or the value 1.
 - Short for “binary digit”—a digit that can assume one of **two** values.
- Characters
 - It’s tedious for people to work with data in the low-level form of bits

1.3 Data Hierarchy (3 of 6)

- Instead, they prefer to work with **decimal digits** (0–9), **letters** (A–Z and a–z), and **special symbols** (e.g., \$, @, %, &, *, (,), –, +, ", :, ? and /)
- The computer's **character set** is the set of all the characters used to write programs and represent- data items.
- Computers process only 1s and 0s, so a computer's character set represents every character as a pattern of 1s and 0s.
- C supports various character sets (including **Unicode**®) that are composed of characters containing one, two or four bytes (8, 16 or 32 bits).
- Unicode contains characters for many of the world's languages.

1.3 Data Hierarchy (4 of 6)

- Fields
 - Composed of characters or bytes.
 - A field is a group of characters or bytes that conveys meaning.
- Records
 - Several related fields can be used to compose a **record**—a group of related fields.

1.3 Data Hierarchy (5 of 6)

- File
 - A group of related records.
 - More generally, a file contains arbitrary data in arbitrary formats.
 - In some operating systems, a file is viewed simply as a **sequence of bytes**—any organization of the bytes in a file, such as organizing the data into records, is a view created by the application programmer.
- Database
 - A **database** is a collection of data organized for easy access and manipulation.

1.3 Data Hierarchy (6 of 6)

- The most popular model is the **relational database**, in which data is stored in simple **tables**.
- A table includes **records** and **fields**.
- You can **search**, **sort** and otherwise manipulate the data based on its relationship to multiple tables or databases.
- Big Data
 - According to IBM, approximately 2.5 quintillion bytes (2.5 **exabytes**) of data are created daily and 90% of the world's data was created in just the past two years
 - According to an IDC study, the global data supply will reach 40 **zettabytes** (equal to 40 trillion gigabytes) annually by 2020.
 - **Big data** applications deal with massive amounts of data

Figure 1.3 Byte Measurements

Unit	Bytes	Which is approximately
1 kilobyte (KB)	1024 bytes	10^3 (1024 bytes exactly)
1 megabyte (MB)	1024 kilobytes	10^6 (1,000,000 bytes)
1 gigabyte (GB)	1024 megabytes	10^9 (1,000,000,000 bytes)
1 terabyte (TB)	1024 gigabytes	10^{12} (1,000,000,000,000 bytes)
1 petabyte (PB)	1024 terabytes	10^{15} (1,000,000,000,000,000 bytes)
1 exabyte (EB)	1024 petabytes	10^{18} (1,000,000,000,000,000,000 bytes)
1 zettabyte (ZB)	1024 exabytes	10^{21} (1,000,000,000,000,000,000,000 bytes)

1.4 Machine Languages, Assembly Languages and High-Level Languages (1 of 3)

- Programmers write instructions in various programming languages, some directly understandable by computers and others requiring intermediate **translation** steps.
- Any computer can directly understand only its own **machine language**, defined by its hardware design.
- Machine languages generally consist of numbers (ultimately reduced to 1s and 0s). Such languages are cumbersome for humans.
- Programming in machine language—the numbers that computers could directly understand—was simply too slow and tedious for most programmers.
- Instead, they began using English like abbreviations to represent elementary operations.

1.4 Machine Languages, Assembly Languages and High-Level Languages (2 of 3)

- These abbreviations formed the basis of **assembly languages**.
- **Translator programs** called **assemblers** were developed to convert assembly-language programs to machine language.
- Although assembly-language code is clearer to humans, it's incomprehensible to computers until translated to machine language.
- To speed the programming process even further, **high-level languages** were developed in which single statements could be written to accomplish substantial tasks.
- High-level languages allow you to write instructions that look almost like everyday English and contain commonly used mathematical expressions.

1.4 Machine Languages, Assembly Languages and High-Level Languages (3 of 3)

- Translator programs called **compilers** convert high-level language programs into machine language.
- **Interpreter** programs were developed to execute high-level language programs directly, although more slowly than compiled programs.
- **Scripting languages** such as JavaScript and PHP are processed by interpreters.

1.5 The C Programming Language (1 of 3)

- 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.

1.5 The C Programming Language (2 of 3)

- The C language was evolved from B by Dennis Ritchie at Bell Laboratories and was originally implemented in 1972.
- C initially became widely known as the development language of the UNIX operating system.
- Many of today's leading operating systems are written in C and/or C++.
- C is mostly hardware independent.
- With careful design, it's possible to write C programs that are portable to most computers.

1.5 The C Programming Language (3 of 3)

Built for Performance

- C is widely used to develop systems that demand performance, such as operating systems, embedded systems, real-time systems and communications systems (Figure 1.4).

Figure 1.4 Some Popular Performance-Oriented C Applications (1 of 2)

Application	Description
Operating systems	C's portability and performance make it desirable for implementing operating systems, such as Linux and portions of Microsoft's Windows and Google's Android. Apple's OS X is built in Objective-C, which was derived from C. We discuss some key popular desktop/notebook operating systems and mobile operating systems in Section 1.11.
Embedded systems	The vast majority of the microprocessors produced each year are embedded in devices other than general-purpose computers. These embedded systems include navigation systems, smart home appliances, home security systems, smartphones, tablets, robots, intelligent traffic intersections and more. C is one of the most popular programming languages for developing embedded systems, which typically need to run as fast as possible and conserve memory. For example, a car's antilock brakes must respond immediately to slow or stop the car without skidding; game controllers used for video games should respond instantaneously to prevent any lag between the controller and the action in the game, and to ensure smooth animations.

Figure 1.4 Some Popular Performance-Oriented C Applications (2 of 2)

Application	Description
Real-time systems	Real-time systems are often used for “mission-critical” applications that require nearly instantaneous and predictable response times. Real-time systems need to work continuously for example, an air-traffic-control system must constantly monitor the positions and velocities of the planes and report that information to air-traffic controllers without delay so that they can alert the planes to change course if there’s a possibility of a collision.
Communications systems	Communications systems need to route massive amounts of data to their destinations quickly to ensure that things such as audio and video are delivered smoothly and without delay.

Portability Tip 1.1

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

1.6 C Standard Library (1 of 3)

- As you'll learn in Chapter 5, C programs consist of pieces called **functions**.
- You can program all the functions you need to form a C program, but most C programmers take advantage of the rich collection of existing functions called the **C Standard Library**.
- Visit the following website for the C Standard Library documentation:
www.dinkumware.com/manuals/#Standard%20C%20Library
- This textbook encourages a **building-block approach** to creating programs.

1.6 C Standard Library (2 of 3)

- Avoid reinventing the wheel.
- Instead, use existing pieces—this is called **software reuse**.
- When programming in C you'll typically use the following building blocks:
 - C Standard Library functions
 - Functions you create yourself
 - Functions other people (whom you trust) have created and made available to you

1.6 C Standard Library (3 of 3)

- The advantage of creating your own functions is that you'll know exactly how they work. You'll be able to examine the C code.
- The disadvantage is the time-consuming effort that goes into designing, developing and debugging new functions.

Performance Tip 1.1

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

Portability Tip 1.2

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

1.7 C++ and Other C-Based Languages (1 of 2)

- 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 more productive.

1.7 C++ and Other C-Based Languages (2 of 2)

- Chapters 15-23 present a condensed treatment of C++ selected from our book **C++ How to Program, 9/e**.
- Figure 1.5 introduces several other popular C-based programming languages.

Figure 1.5 Popular C-Based Programming Languages (1 of 3)

Programming language	Description
Objective-C	Objective-C is an object-oriented language based on C. It was developed in the early 1980s and later acquired by NeXT, which in turn was acquired by Apple. It has become the key programming language for the OS X operating system and all iOS-powered devices (such as iPods, iPhones and iPads).
Java	Sun Microsystems in 1991 funded an internal corporate research project which resulted in the C++-based object-oriented programming language called Java. A key goal of Java is to enable the writing of programs that will run on a broad variety of computer systems and computer-controlled devices. This is sometimes called “write once, run anywhere.” Java is 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 (smartphones, television set-top boxes and more) and for many other purposes. Java is also the language of Android app development.

Figure 1.5 Popular C-Based Programming Languages (2 of 3)

Programming language	Description
C#	Microsoft's three primary object-oriented programming languages are Visual Basic (based on the original Basic), Visual C++ (based on C++) and Visual C# (based on C++ and Java, and developed for integrating the Internet and the web into computer applications). Non-Microsoft versions of C# are also available.
PHP	PHP, an object-oriented, open-source scripting language supported by a community of users and developers, is used by millions of websites. PHP is platform independent—implementations exist for all major UNIX, Linux, Mac and Windows operating systems. PHP also supports many databases, including the popular open-source MySQL.
Python	Python, another object-oriented scripting language, was released publicly in 1991. Developed by Guido van Rossum of the National Research Institute for Mathematics and Computer Science in Amsterdam (CWI), Python draws heavily from Modula-3—a systems programming language. Python is “extensible”—it can be extended through classes and programming interfaces.

Figure 1.5 Popular C-Based Programming Languages (3 of 3)

Programming language	Description
JavaScript	JavaScript is the most widely used scripting language. It's primarily used to add dynamic behavior to web pages—for example, animations and improved interactivity with the user. It's provided with all major web browsers.
Swift	Swift, Apple's new programming language for developing iOS and Mac apps, was announced at the Apple World Wide Developer Conference (WWDC) in June 2014. Although apps can still be developed and maintained with Objective-C, Swift is Apple's app-development language of the future. It's a modern language that eliminates some of the complexity of Objective-C, making it easier for beginners and those transitioning from other high-level languages such as Java, C# , C++ and C. Swift emphasizes performance and security, and has full access to the iOS and Mac programming capabilities.

1.8 Object Technology (1 of 13)

- **Objects**, or more precisely the **classes** objects come from, are essentially **reusable** software components.
- Almost any **noun** can be reasonably represented as a software object in terms of **attributes** (e.g., name, color and size) and **behaviors** (e.g., calculating, moving and communicating).
- Software developers are discovering that using a modular, object-oriented design and implementation approach can make software-development groups much more productive than was possible with earlier techniques—object-oriented programs are often easier to understand, correct and modify.

1.8 Object Technology (2 of 13)

The Automobile as an Object

- Suppose you want to **drive a car and make it go faster by pressing its accelerator pedal**.
- Before you can drive a car, someone has to **design** it.
- A car typically begins as engineering drawings, similar to the **blueprints** that describe the design of a house.
- These drawings include the design for an accelerator pedal.

1.8 Object Technology (3 of 13)

- The pedal **hides** from the driver the complex mechanisms that actually make the car go faster, just as the brake pedal hides the mechanisms that slow the car, and the steering wheel **hides** the mechanisms that turn the car.
- This enables people with little or no knowledge of how engines, braking and steering mechanisms work to drive a car easily.
- Before you can drive a car, it must be **built** from the engineering drawings that describe it.
- A completed car has an **actual** accelerator pedal to make the car go faster, but even that's not enough—the car won't accelerate on its own (hopefully!), so the driver must **press** the pedal to accelerate the car.

1.8 Object Technology (4 of 13)

Methods and Classes

- Performing a task in a program requires a **method**.
- The method houses the program statements that actually perform its tasks.
- It hides these statements from its user, just as a car's accelerator pedal hides from the driver the mechanisms of making the car go faster.
- In object-oriented programming languages, we create a program unit called a **class** to house the set of methods that perform the class's tasks.

1.8 Object Technology (5 of 13)

- For example, a class that represents a bank account might contain one method to **deposit** money to an account, another to **withdraw** money from an account and a third to **inquire** what the account's current balance is.
- A class is similar in concept to a car's engineering drawings, which house the design of an accelerator pedal, steering wheel, and so on.

1.8 Object Technology (6 of 13)

Instantiation

- Just as someone has to **build a car** from its engineering drawings before you can actually drive a car, you must **build an object** from a class before a program can perform the tasks that the class's methods define.
- The process of doing this is called **instantiation**. An object is then referred to as an **instance** of its class.

1.8 Object Technology (7 of 13)

Reuse

- Just as a car's engineering drawings can be **reused** many times to build many cars, you can **reuse** a class many times to build many objects.
- Reuse of existing classes when building new classes and programs saves time and effort.
- Reuse also helps you build more reliable and effective systems, because existing classes and components often have gone through extensive **testing**, **debugging** and **performance tuning**.

1.8 Object Technology (8 of 13)

Messages and Method Calls

- When you drive a car, pressing its gas pedal sends a **message** to the car to perform a task—that is, to go faster. Similarly, you **send messages to an object**.
- Each message is implemented as a **method call** that tells a method of the object to perform its task.
- For example, a program might call a particular bank-account object's **deposit** method to increase the account's balance.

1.8 Object Technology (9 of 13)

Attributes and Instance Variables

- A car, besides having capabilities to accomplish tasks, also has **attributes**, such as its color, its number of doors, the amount of gas in its tank, its current speed and its record of total miles driven (i.e., its odometer reading).
- Like its capabilities, the car's attributes are represented as part of its design in its engineering diagrams (which, for example, include an odometer and a fuel gauge).
- As you drive an actual car, these attributes are carried along with the car.

1.8 Object Technology (10 of 13)

- Every car maintains its **own** attributes.
- For example, each car knows how much gas is in its own gas tank, but **not** how much is in the tanks of **other** cars.
- An object, similarly, has attributes that it carries along as it's used in a program.

1.8 Object Technology (11 of 13)

- These attributes are specified as part of the object's class.
- For example, a bank-account object has a **balance attribute** that represents the amount of money in the account.
- Each bank-account object knows the balance in the account it represents, but **not** the balances of the **other** accounts in the bank.
- Attributes are specified by the class's **instance variables**.

1.8 Object Technology (12 of 13)

Encapsulation

- Classes **encapsulate** (i.e., wrap) attributes and methods into objects—an object's attributes and methods are intimately related.
- Objects may communicate with one another, but normally they're not allowed to know how other objects are implemented—implementation details are **hidden** within the objects themselves.
- This **information hiding** is crucial to good software engineering.

1.8 Object Technology (13 of 13)

Inheritance

- A new class of objects can be created quickly and conveniently by **inheritance**—the new class absorbs the characteristics of an existing class, possibly customizing them and adding unique characteristics of its own.
- In our car analogy, an object of class “convertible” certainly **is an** object of the more **general** class “automobile,” but more **specifically**, the roof can be raised or lowered.

Software Engineering Observation 1.1

Use a building-block approach to creating your programs. Avoid reinventing the wheel—use existing high-quality pieces wherever possible. Such **software reuse** is a key benefit of object-oriented programming.

1.9 Typical C Program Development Environment (1 of 2)

- 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
- These are: **edit, preprocess, compile, link, load and execute.**
- Although **C How to Program, 7/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.

1.9 Typical C Program Development Environment (2 of 2)

- [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/C to locate “getting started” tutorials for popular C compilers and development environments.

1.9 Phase 1: Creating a Program

- 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.

1.9 Phases 2 and 3: Preprocessing and Compiling a C Program (1 of 2)

- In Phase 2, the 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.

1.9 Phases 2 and 3: Preprocessing and Compiling a C Program (2 of 2)

- 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.
- A **syntax error** occurs when the compiler cannot recognize a statement because it violates the rules of the language.
- Syntax errors are also called **compile errors**, or **compile-time errors**.

Figure 1.6 Typical C Development Environment (1 of 2)

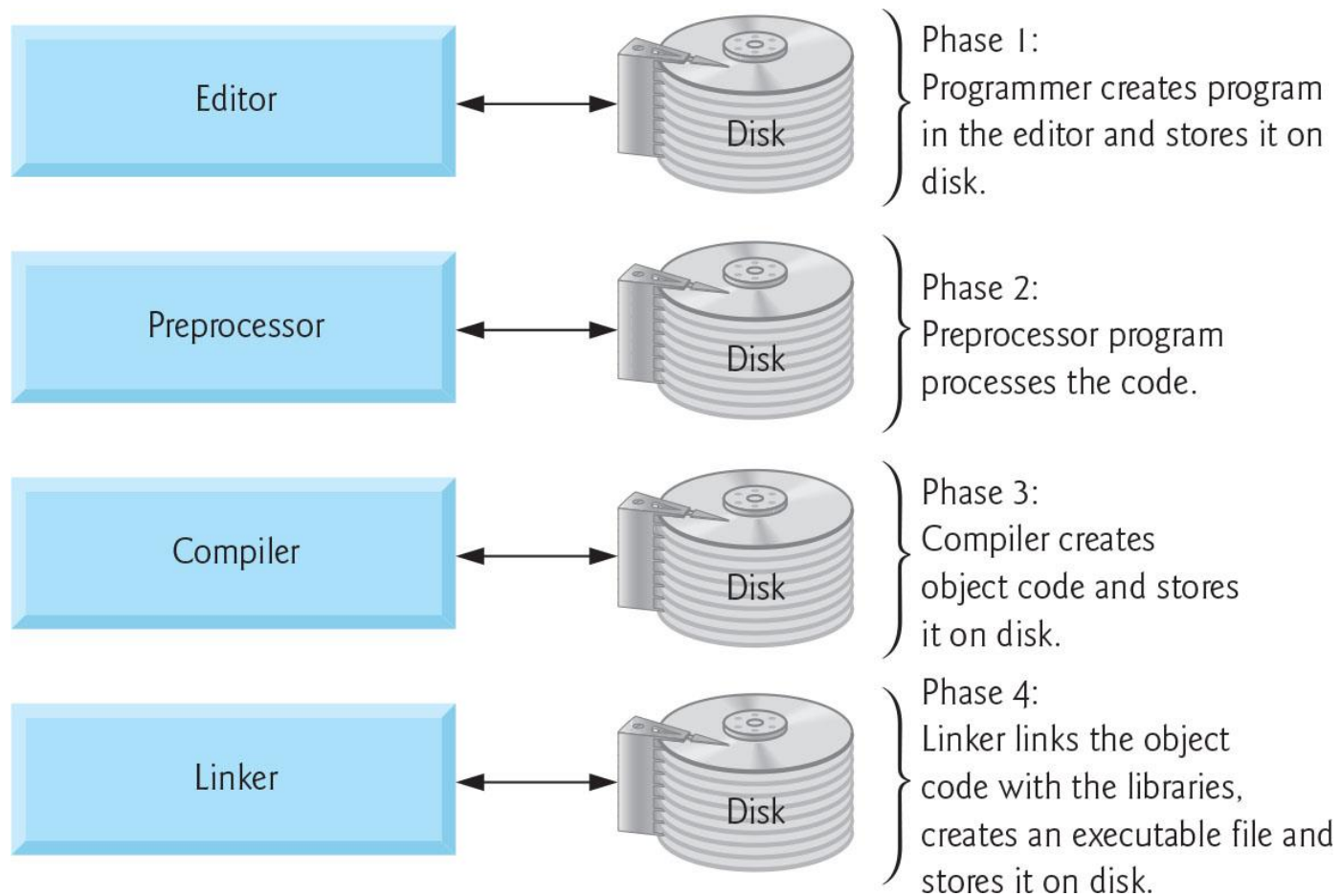
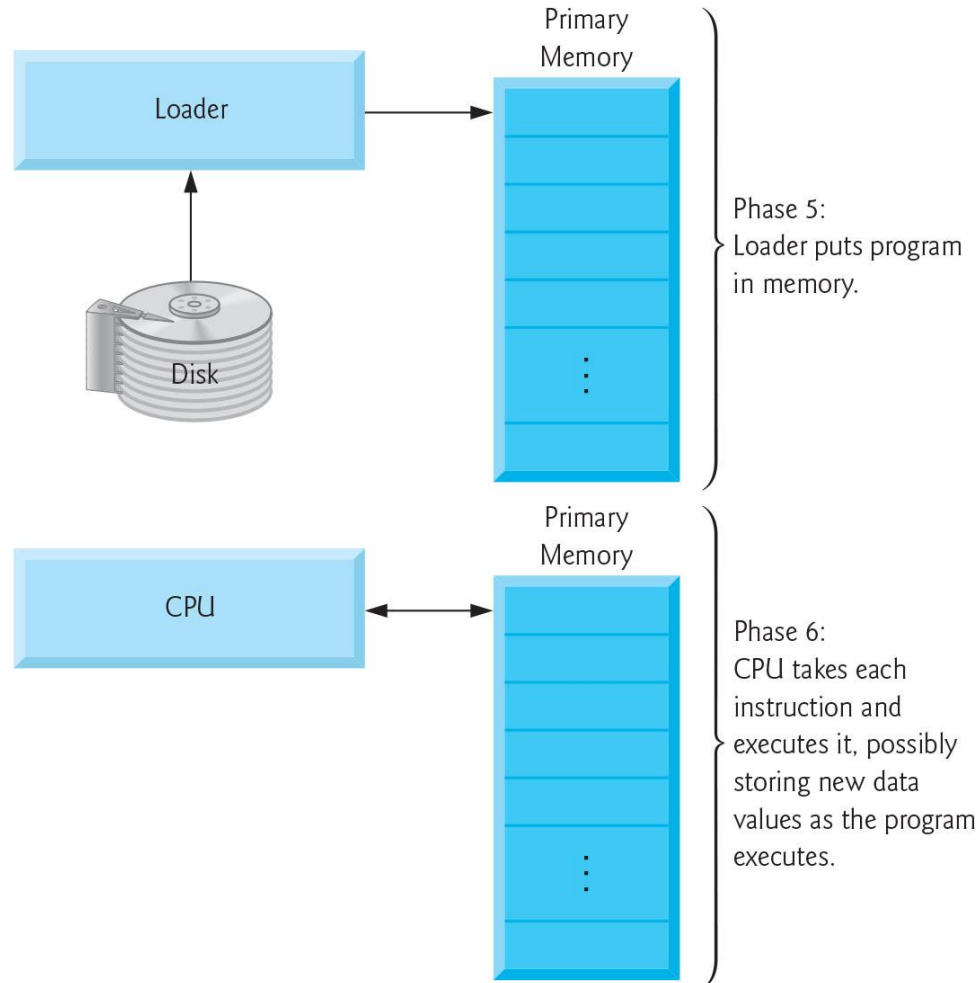


Figure 1.6 Typical C Development Environment (2 of 2)



1.9 Phase 4: Linking (1 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 **gcc** (the GNU compiler).

1.9 Phase 4: Linking (2 of 2)

- To compile and link a program named `welcome.c` type
 - `gcc welcome.c`
- at the Linux prompt and press the **Enter** key (or **Return** key).
- [Note: Linux commands are case sensitive; make sure that each `c` is lowercase 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.

1.9 Phase 5: Loading

- 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.

1.9 Phase 6: Execution

- 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**.

1.9 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'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.
- 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.

Common Programming Error 1.1

Errors such as division-by-zero occur as a program runs, so they are called runtime errors or execution-time errors. Divide-by-zero is generally a fatal error, i.e., one that causes the program to terminate immediately without successfully performing its job. Nonfatal errors allow programs to run to completion, often producing incorrect results.

1.9 Standard Input, Standard Output and Standard Error Streams (1 of 2)

- 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.

1.9 Standard Input, Standard Output and Standard Error Streams (2 of 2)

- 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.

1.10 Test-Driving a C Application in Windows, Linux and Mac OS X (1 of 2)


- In this section, you'll run and interact with your first C application.
- You'll begin by running an entertaining guess-the-number game, which picks a number from 1 to 1000 and prompts you to guess it.
- 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.

1.10 Test-Driving a C Application in Windows, Linux and Mac OS X (2 of 2)

- For this test-drive only, we've modified this application from the exercise you'll be asked to create in Chapter 5.
- Normally this application randomly selects the correct answers.
- The modified application uses the same sequence of correct answers every time you execute the program (though this may vary by compiler), so you can use the same guesses we use in this section and see the same results.

1.10.1 Running a C Application from the Windows Command Prompt (1 of 7)

Figure 1.7 Opening a **Command Prompt** window and changing the directory.

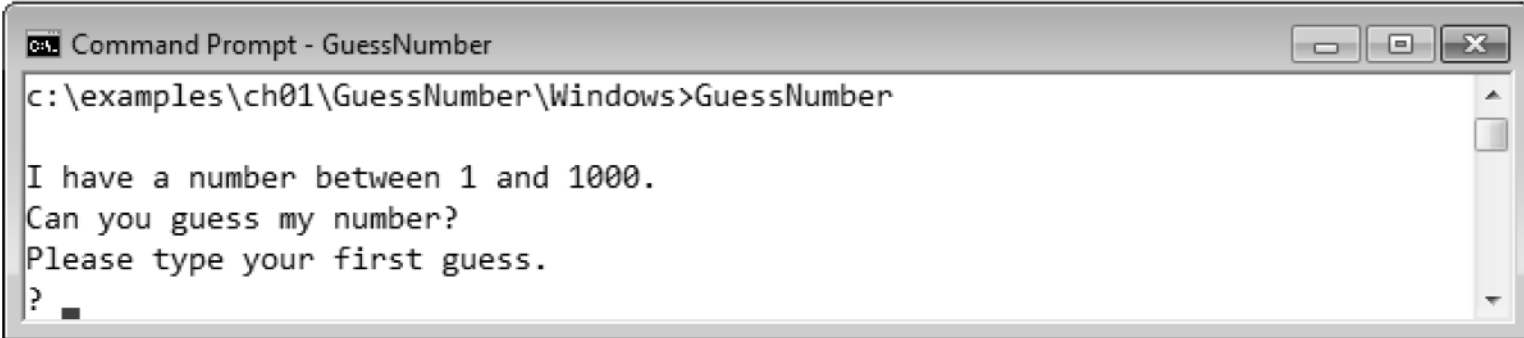


```
Command Prompt
C:\>cd c:\examples\ch01\GuessNumber\Windows
c:\examples\ch01\GuessNumber\Windows>
```

The image shows a screenshot of a Windows Command Prompt window. The title bar reads "Command Prompt". The command prompt shows the current directory as "C:\>". The user has entered the command "cd c:\examples\ch01\GuessNumber\Windows", and the prompt has moved to the new directory, showing "c:\examples\ch01\GuessNumber\Windows>".

1.10.1 Running a C Application from the Windows Command Prompt (2 of 7)

Figure 1.8 Running the **GuessNumber** application.

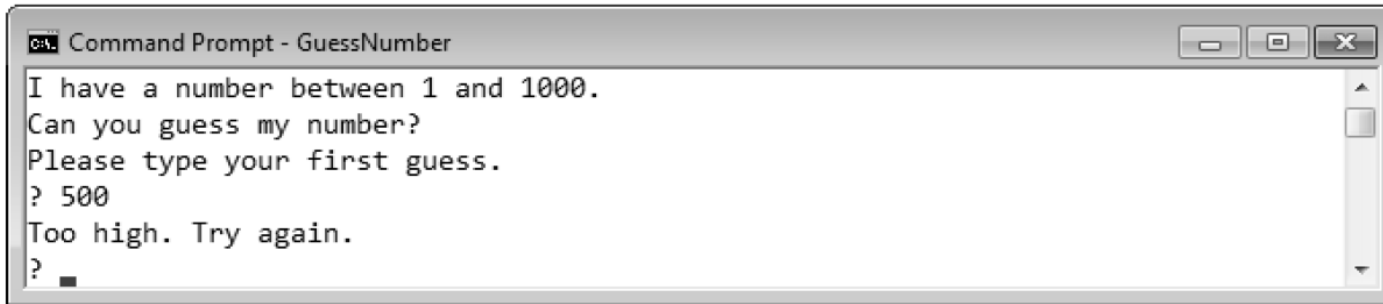


```
Command Prompt - GuessNumber
c:\examples\ch01\GuessNumber\Windows>GuessNumber

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


1.10.1 Running a C Application from the Windows Command Prompt (3 of 7)

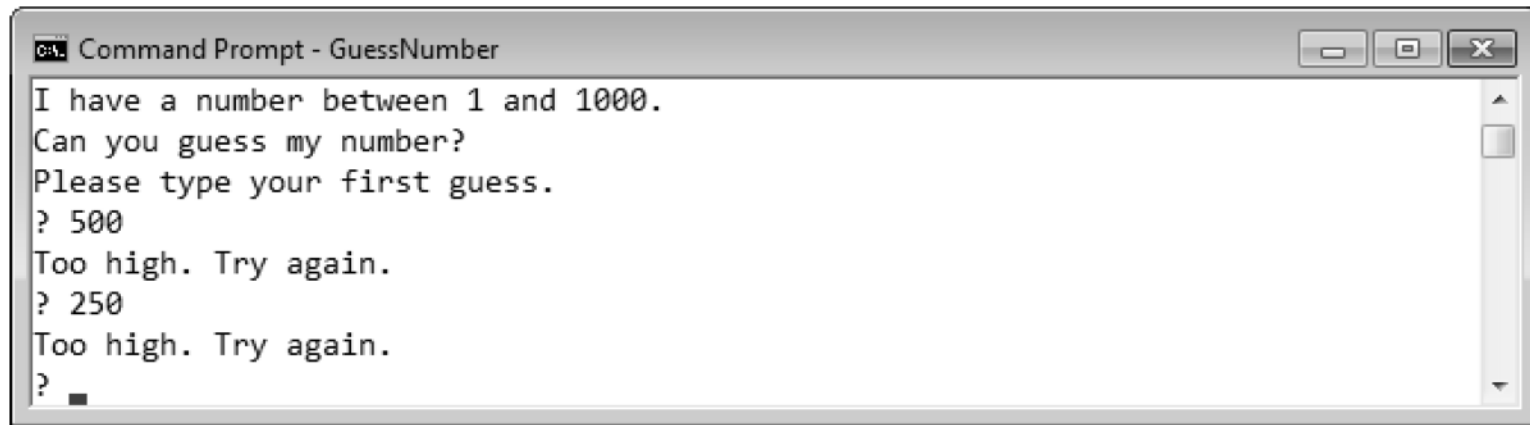
Figure 1.9 Entering your first guess.



```
Command Prompt - GuessNumber
I have a number between 1 and 1000.
Can you guess my number?
Please type your first guess.
? 500
Too high. Try again.
? 
```

1.10.1 Running a C Application from the Windows Command Prompt (4 of 7)

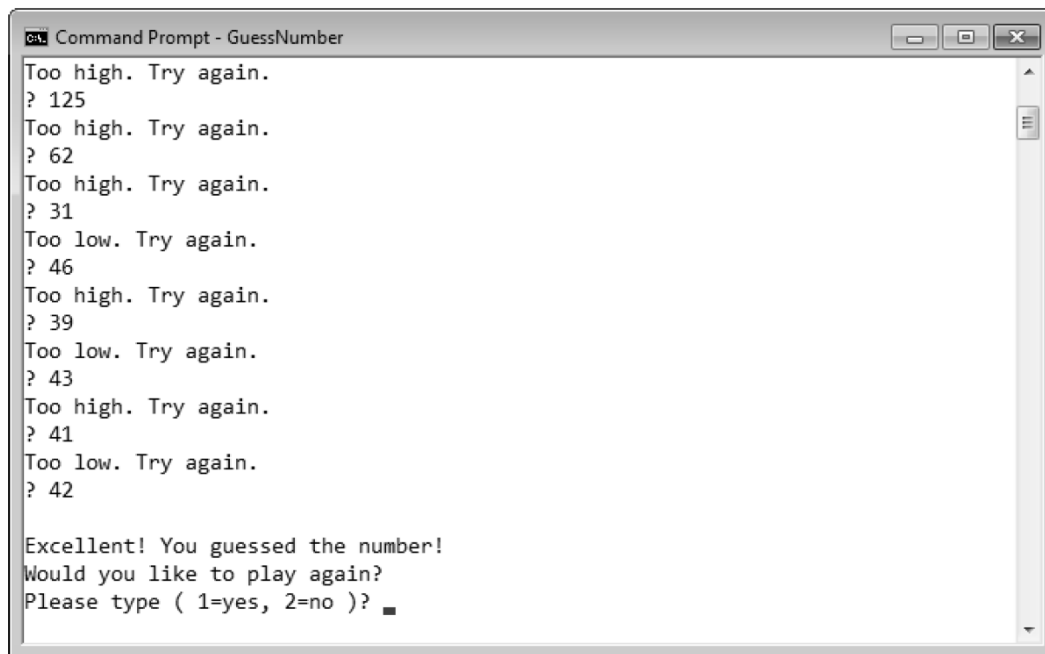
Figure 1.10 Entering a second guess and receiving feedback.



```
C:\> Command Prompt - GuessNumber
I have a number between 1 and 1000.
Can you guess my number?
Please type your first guess.
? 500
Too high. Try again.
? 250
Too high. Try again.
? 
```

1.10.1 Running a C Application from the Windows Command Prompt (5 of 7)

Figure 1.11 Entering additional guesses and guessing the correct number.

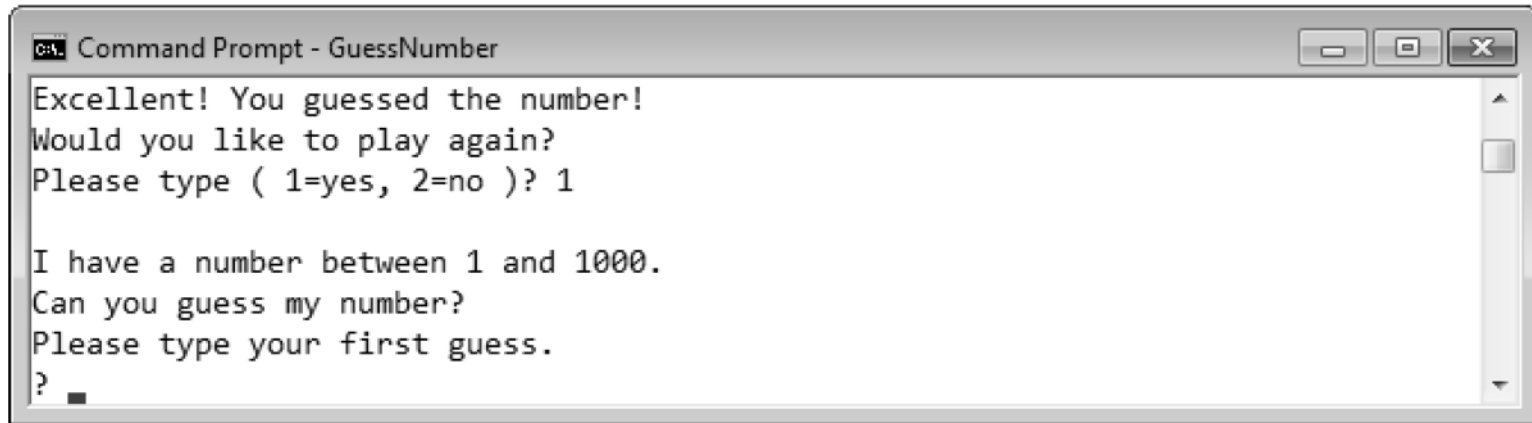


```
Command Prompt - GuessNumber
Too high. Try again.
? 125
Too high. Try again.
? 62
Too high. Try again.
? 31
Too low. Try again.
? 46
Too high. Try again.
? 39
Too low. Try again.
? 43
Too high. Try again.
? 41
Too low. Try again.
? 42

Excellent! You guessed the number!
Would you like to play again?
Please type ( 1=yes, 2=no )? █
```

1.10.1 Running a C Application from the Windows Command Prompt (6 of 7)

Figure 1.12 Playing the game again.

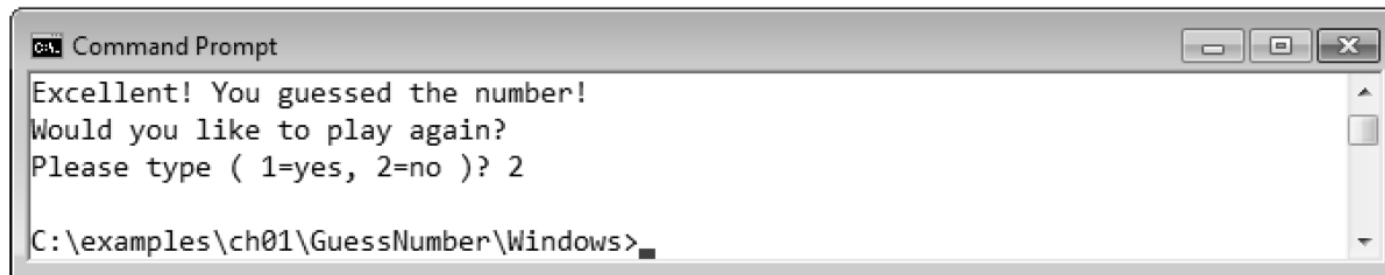


```
Command Prompt - GuessNumber
Excellent! You guessed the number!
Would you like to play again?
Please type ( 1=yes, 2=no )? 1

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

1.10.1 Running a C Application from the Windows Command Prompt (7 of 7)

Figure 1.13 Exiting the game.



```
Command Prompt
Excellent! You guessed the number!
Would you like to play again?
Please type ( 1=yes, 2=no )? 2
C:\examples\ch01\GuessNumber\Windows>
```

1.10.2 Running a C Application Using GNU C with Linux (1 of 8)

Figure 1.14 Changing to the **GuessNumber** application's directory.

```
~$ cd examples/ch01/GuessNumber/GNU  
~/examples/ch01/GuessNumber/GNU$
```

1.10.2 Running a C Application Using GNU C with Linux (2 of 8)

Figure 1.15 Compiling the **GuessNumber** application using the **gcc** command.

```
~/examples/ch01/GuessNumber/GNU$ gcc -std=c11 GuessNumber.c -o GuessNumber  
~/examples/ch01/GuessNumber/GNU$
```

1.10.2 Running a C Application Using GNU C with Linux (3 of 8)

Figure 1.16 Running the **GuessNumber** application.

```
~/examples/ch01/GuessNumber/GNU$ ./GuessNumber
```

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


1.10.2 Running a C Application Using GNU C with Linux (4 of 8)

Figure 1.17 Entering an initial guess.

```
~/examples/ch01/GuessNumber/GNU$ ./GuessNumber
```

```
I have a number between 1 and 1000.
```

```
Can you guess my number?
```

```
Please type your first guess.
```

```
? 500
```

```
Too high. Try again.
```

```
?
```

1.10.2 Running a C Application Using GNU C with Linux (5 of 8)

Figure 1.18 Entering a second guess and receiving feedback.

```
~/examples/ch01/GuessNumber/GNU$ ./GuessNumber
```

```
I have a number between 1 and 1000.
```

```
Can you guess my number?
```

```
Please type your first guess.
```

```
? 500
```

```
Too high. Try again.
```

```
? 250
```

```
Too low. Try again.
```

```
?
```

1.10.2 Running a C Application Using GNU C with Linux (6 of 8)

Figure 1.19 Entering additional guesses and guessing the correct number.

```
Too low. Try again.  
? 375  
Too low. Try again.  
? 437  
Too high. Try again.  
? 406  
Too high. Try again.  
? 391  
Too high. Try again.  
? 383  
Too low. Try again.  
? 387  
Too high. Try again.  
? 385  
Too high. Try again.  
? 384  
  
Excellent! You guessed the number!  
Would you like to play again?  
Please type ( 1=yes, 2=no )?
```

1.10.2 Running a C Application Using GNU C with Linux (7 of 8)

Figure 1.20 Playing the game again.

```
Excellent! You guessed the number!  
Would you like to play again?  
Please type ( 1=yes, 2=no )? 1
```

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

1.10.2 Running a C Application Using GNU C with Linux (8 of 8)

Figure 1.21 Exiting the game.

```
Excellent! You guessed the number!  
Would you like to play again?  
Please type ( 1=yes, 2=no )? 2  
  
~/examples/ch01/GuessNumber/GNU$
```

1.10.3 Running a C Application Using the Terminal on Mac OS X (1 of 8)

Figure 1.22 Changing to the **GuessNumber** application's directory.

```
hostName:~ userFolder$ cd Documents/examples/ch01/GuessNumber/GNU  
hostName:GNU$
```

1.10.3 Running a C Application Using the Terminal on Mac OS X (2 of 8)

Figure 1.23 Compiling the **GuessNumber** application using the **gcc** command.

```
hostName:GNU~ userFolder$ clang GuessNumber.c -o GuessNumber  
hostName:GNU~ userFolder$
```

1.10.3 Running a C Application Using the Terminal on Mac OS X (3 of 8)

Figure 1.24 Running the **GuessNumber** application.

```
hostName:GNU~ userFolder$ ./GuessNumber
```

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


1.10.3 Running a C Application Using the Terminal on Mac OS X (4 of 8)

Figure 1.25 Entering an initial guess.

```
hostName:GNU~ userFolder$ ./GuessNumber
```

```
I have a number between 1 and 1000.
```

```
Can you guess my number?
```

```
Please type your first guess.
```

```
? 500
```

```
Too low. Try again.
```

```
?
```

1.10.3 Running a C Application Using the Terminal on Mac OS X (5 of 8)

Figure 1.26 Entering a second guess and receiving feedback.

```
hostName:GNU~ userFolder$ ./GuessNumber
```

```
I have a number between 1 and 1000.
```

```
Can you guess my number?
```

```
Please type your first guess.
```

```
? 500
```

```
Too low. Try again.
```

```
? 750
```

```
Too low. Try again.
```

```
?
```

1.10.3 Running a C Application Using the Terminal on Mac OS X (6 of 8)

Figure 1.27 Entering additional guesses and guessing the correct number.

```
? 825
Too high. Try again.
? 788
Too low. Try again.
? 806
Too low. Try again.
? 815
Too high. Try again.
? 811
Too high. Try again.
? 808

Excellent! You guessed the number!
Would you like to play again?
Please type ( 1=yes, 2=no )?
```

1.10.3 Running a C Application Using the Terminal on Mac OS X (7 of 8)

Figure 1.28 Playing the game again.

```
Excellent! You guessed the number!  
Would you like to play again?  
Please type ( 1=yes, 2=no )? 1  
  
I have a number between 1 and 1000.  
Can you guess my number?  
Please type your first guess.  
?
```

1.10.3 Running a C Application Using the Terminal on Mac OS X (8 of 8)

Figure 1.29 Exiting the game.

```
Excellent! You guessed the number!  
Would you like to play again?  
Please type ( 1=yes, 2=no )? 2  
  
hostName:GNU~ userFolder$
```

1.11 Operating Systems

- **Operating systems** are software systems that make using computers more convenient for users, application developers and system administrators.
- Operating systems provide services that allow each application to execute safely, efficiently and **concurrently** (i.e., in parallel) with other applications.
- The software that contains the core components of the operating system is called the **kernel**.
- Popular desktop operating systems include Linux, Windows 7 and Mac OS X.
- Popular mobile operating systems used in smartphones and tablets include Google's Android, Apple's iOS (for iPhone, iPad and iPod Touch devices), BlackBerry OS and Windows Phone 7.

1.11.1 Windows—A Proprietary Operating System

- In the mid-1980s, Microsoft developed the **Windows operating system**, consisting of a graphical user interface built on top of DOS—an enormously popular personal-computer operating system of the time that users interacted with by **typing** commands.
- Windows borrowed from many concepts (such as icons, menus and windows) developed by Xerox PARC and popularized by early Apple Macintosh operating systems.
- Windows is a **proprietary** operating system—it's controlled by Microsoft exclusively.

1.11.2 Linux—An Open-Source Operating System (1 of 2)

- The Linux operating system is perhaps the greatest success of the **open-source** movement.
- **Open-source software** departs from the **proprietary** software development style that dominated software's early years.
- With open-source development, individuals and companies **contribute** their efforts in developing, maintaining and evolving software in exchange for the right to use that software for their own purposes, typically at no charge.
- Rapid improvements to computing and communications, decreasing costs and open-source software have made it much easier and more economical to create a software-based business now than just a decade ago.
- A great example is Facebook, which was launched from a college dorm room and built with open-source software.

1.11.2 Linux—An Open-Source Operating System (2 of 2)

- The **Linux** kernel is the core of the most popular open-source, freely distributed, full-featured operating system.
- It's developed by a loosely organized team of volunteers and is popular in servers, personal computers and embedded systems.
- Unlike that of proprietary operating systems like Microsoft's Windows and Apple's Mac OS X, Linux source code (the program code) is available to the public for examination and modification and is free to download and install.
- Linux has become extremely popular on servers and in embedded systems, such as Google's Android-based smartphones.

1.11.3 Apple's Mac OS X; Apple's iOS for iPhone®, iPad® and iPod Touch® Devices (1 of 2)

- In 1979, Steve Jobs and several Apple employees visited Xerox PARC (Palo Alto Research Center) to learn about Xerox's desktop computer that featured a graphical user interface (GUI).
- That GUI served as the inspiration for the Apple Macintosh, launched with much fanfare in a memorable Super Bowl ad in 1984.
- The Objective-C programming language, created by Brad Cox and Tom Love at Stepstone in the early 1980s, added capabilities for object-oriented programming (OOP) to the C programming language.

1.11.3 Apple's Mac OS X; Apple's iOS for iPhone®, iPad® and iPod Touch® Devices (2 of 2)

- Steve Jobs left Apple in 1985 and founded NeXT Inc. In 1988, NeXT licensed Objective-C from StepStone and developed an Objective-C compiler and libraries which were used as the platform for the NeXTSTEP operating system's user interface and Interface Builder—used to construct graphical user interfaces.
- Jobs returned to Apple in 1996 when Apple bought NeXT. Apple's Mac OS X operating system is a descendant of NeXTSTEP.
- Apple's proprietary iPhone operating system, **iOS**, is derived from Apple's Mac OS X and is used in the iPhone, iPad and iPod Touch devices.

1.11.4 Google's Android

- **Android**—the fastest growing mobile and smartphone operating system—is based on the Linux kernel and Java.
- One benefit of developing Android apps is the openness of the platform. The operating system is open source and free.
- The Android operating system is used in numerous smartphones, e-reader devices, tablet computers, in-store touch-screen kiosks, cars, robots, multimedia players and more.

1.12 The Internet and the World Wide Web (1 of 4)

- The Internet—a global network of computers—was made possible by the **convergence of computing and communications technologies**.
- In the late 1960s, ARPA (the Advanced Research Projects Agency) rolled out blueprints for networking the main computer systems of about a dozen ARPA-funded universities and research institutions.
- ARPA proceeded to implement the **ARPANET**, which eventually evolved into today's **Internet**.
- A primary goal for ARPANET was to allow multiple users to send and receive information simultaneously over the same communications paths (e.g., phone lines).

1.12 The Internet and the World Wide Web (2 of 4)

TCP/IP

- The protocol (i.e., set of rules) for communicating over the ARPANET became known as **TCP—the Transmission Control Protocol**.
- TCP ensured that messages were properly routed from sender to receiver and that they arrived intact.
- As the Internet evolved, organizations worldwide were implementing their own networks. One challenge was to get these different networks to communicate.
- ARPA accomplished this with the development of **IP—the Internet Protocol**, truly creating a network of networks, the current architecture of the Internet.
- The combined set of protocols is now commonly called **TCP/IP**.

1.12 The Internet and the World Wide Web (3 of 4)

World Wide Web, HTML, HTTP

- The **World Wide Web** allows you to execute web-based applications and to locate and view multimedia-based documents on almost any subject over the Internet.
- In 1989, Tim Berners-Lee of CERN began to develop a technology for sharing information via hyperlinked text documents. Berners-Lee called his invention the **HyperText Markup Language (HTML)**.
- He also wrote communication protocols to form the backbone of his new information system, which he called the World Wide Web.
- In particular, he wrote the **Hypertext Transfer Protocol (HTTP)**—a communications protocol used to send information over the web.

1.12 The Internet and the World Wide Web (4 of 4)

Web Services

- **Web services** are software components stored on one computer that can be accessed by an app (or other software component) on another computer over the Internet.
- With web services, you can create **mashups**, which enable you to rapidly develop apps by combining complementary web services, often from multiple organizations and possibly other forms of information feeds.
- Programmableweb (<http://www.programmableweb.com/>) provides a directory of over 11,150 APIs and 7,300 mashups, plus how-to guides and sample code for creating your own mashups. lists some popular web services.
- According to Programmableweb, the three most widely used APIs for mashups are Google Maps, Twitter and YouTube.

Figure 1.30 Some Popular Web Services (1 of 2)

<http://www.programmableweb.com/category/all/apis>

Web services source	How it's used
Google Maps	Mapping services
Twitter	Microblogging
YouTube	Video search
Facebook	Social networking
Instagram	Photo sharing
Foursquare	Mobile check-in
LinkedIn	Social networking for business
Groupon	Social commerce
Netflix	Movie rentals
eBay	Internet auctions
Wikipedia	Collaborative encyclopedia

Figure 1.30 Some Popular Web Services (2 of 2)

Web services source	How it's used
PayPal	Payments
Last.fm	Internet radio
Amazon eCommerce	Shopping for books and lots of other products
Salesforce.com	Customer Relationship Management (CRM)
Skype	Internet telephony
Microsoft Bing	Search
Flickr	Photo sharing
Zillow	Real-estate pricing
Yahoo Search	Search
WeatherBug	Weather

Figure 1.31 Web Services Directories

Directory	URL
ProgrammableWeb	www.programmableweb.com
Google Code API Directory	code.google.com/apis/gdata/docs/directory.html

Figure 1.32 A Few Popular Web Mashups

URL	Description
http://twikle.com/	Twikle uses Twitter web services to aggregate popular news stories being shared online.
http://trendsmap.com/	TrendsMap uses Twitter and Google Maps. It allows you to track tweets by location and view them on a map in real time.
http://www.coindesk.com/price/bitcoin-price-ticker-widget/	The Bitcoin Price Ticker Widget uses CoinDesk's APIs to display the real-time Bitcoin price, the day's high and low prices and a graph of the price fluctuations over the last sixty minutes.
http://www.dutranslation.com/	The Double Translation mashup allows you to use Bing and Google translation services simultaneously to translate text to and from over 50 languages. You can then compare the results between the two.
http://musicupdated.com/	Music Updated uses Last.fm and YouTube web services. Use it to track album releases, concert information and more for your favorite artists.

1.13 Some Key Software Development Terminology

Figure 1.33 lists a number of buzzwords that you'll hear in the software development community.

Figure 1.33 Software Technologies (1 of 4)

Technology	Description
Agile software development	Agile software development is a set of methodologies that try to get software implemented faster and using fewer resources. Check out the Agile Alliance (www.agilealliance.org) and the Agile Manifesto (www.agilemanifesto.org).
Refactoring	Refactoring involves reworking programs to make them clearer and easier to maintain while preserving their correctness and functionality. It's widely employed with agile development methodologies. Many IDEs contain built-in refactoring tools to do major portions of the reworking automatically.
Design patterns	Design patterns are proven architectures for constructing flexible and maintainable object-oriented software. The field of design patterns tries to enumerate those recurring patterns, encouraging software designers to reuse them to develop better-quality software using less time, money and effort.

Figure 1.33 Software Technologies (2 of 4)

Technology	Description
Lamp	Lamp is an acronym for the open-source technologies that many developers use to build web applications inexpensively—it stands for Linux , Apache , MySQL and PHP (or Perl or Python —two other popular scripting languages). MySQL is an open-source database-management system. PHP is the most popular open-source server-side “scripting” language for developing web applications. Apache is the most popular web server software. The equivalent for Windows development is WAMP— Windows , Apache , MySQL and PHP .

Figure 1.33 Software Technologies (3 of 4)

Technology	Description
Software as a Service (SaaS)	<p>Software has generally been viewed as a product; most software still is offered this way. If you want to run an application, you buy a software package from a software vendor—often a CD, DVD or web download. You then install that software on your computer and run it as needed. As new versions appear, you upgrade your software, often at considerable cost in time and money. This process can become cumbersome for organizations that must maintain tens of thousands of systems on a diverse array of computer equipment.</p> <p>With Software as a Service (SaaS), the software runs on servers elsewhere on the Internet. When that server is updated, all clients worldwide see the new capabilities—no local installation is needed. You access the service through a browser. Browsers are quite portable, so you can run the same applications on a wide variety of computers from anywhere in the world. Salesforce.com, Google, and Microsoft's Office Live and Windows Live all offer SaaS.</p>

Figure 1.33 Software Technologies (4 of 4)

Technology	Description
Platform as a Service (PaaS)	Platform as a Service (PaaS) provides a computing platform for developing and running applications as a service over the web, rather than installing the tools on your computer. Some PaaS providers are Google App Engine, Amazon EC2 and Windows Azure™.
Cloud computing	SaaS and PaaS are examples of cloud computing . You can use software and data stored in the “cloud”—i.e., accessed on remote computers (or servers) via the Internet and available on demand—rather than having it stored locally on your desktop, notebook computer or mobile device. This allows you to increase or decrease computing resources to meet your needs at any given time, which is more cost effective than purchasing hardware to provide enough storage and processing power to meet occasional peak demands. Cloud computing also saves money by shifting to the service provider the burden of managing these apps (such as installing and upgrading the software, security, backups and disaster recovery).
Software Development Kit (SDK)	Software Development Kits (SDKs) include the tools and documentation developers use to program applications.

Figure 1.34 Software Product-Release Terminology (1 of 2)

Version	Description
Alpha	Alpha software is the earliest release of a software product that's still under active development. Alpha versions are often buggy, incomplete and unstable and are released to a relatively small number of developers for testing new features, getting early feedback, etc.
Beta	Beta versions are released to a larger number of developers later in the development process after most major bugs have been fixed and new features are nearly complete. Beta software is more stable, but still subject to change.
Release candidates	Release candidates are generally feature complete , (mostly) bug free and ready for use by the community, which provides a diverse testing environment—the software is used on different systems, with varying constraints and for a variety of purposes.

Figure 1.34 Software Product-Release Terminology (2 of 2)

Version	Description
Final release	Any bugs that appear in the release candidate are corrected, and eventually the final product is released to the general public. Software companies often distribute incremental updates over the Internet.
Continuous beta	Software that's developed using this approach (for example, Google search or Gmail) generally does not have version numbers. It's hosted in the cloud (not installed on your computer) and is constantly evolving so that users always have the latest version.

1.14 Keeping Up-To-Date with Information Technologies

Figure 1.35 lists key technical and business publications that will help you stay up-to-date with the latest news and trends and technology. You can also find a growing list of Internet-and web-related Resource Centers at www.deitel.com/ResourceCenters.html.

Figure 1.35 Technical and Business Publications (1 of 2)

Publication	URL
AllThingsD	allthingsd.com
Bloomberg BusinessWeek	www.businessweek.com
CNET	news.cnet.com
Communications of the ACM	cacm.acm.org
Computerworld	www.computerworld.com
Engadget	www.engadget.com
eWeek	www.eweek.com
Fast Company	www.fastcompany.com
Fortune	money.cnn.com/magazines/fortune
GigaOM	gigaom.com
Hacker News	news.ycombinator.com

Figure 1.35 Technical and Business Publications (2 of 2)

Publication	URL
IEEE Computer Magazine	www.computer.org/portal/web/computingnow/computer
InfoWorld	www.infoworld.com
Mashable	mashable.com
PCWorld	www.pcworld.com
SD Times	www.sdtimes.com
Slashdot	slashdot.org
Stack Overflow	stackoverflow.com
Technology Review	technologyreview.com
Techcrunch	techcrunch.com
The Next Web	thenextweb.com
The Verge	www.theverge.com
Wired	www.wired.com

Copyright



This work is protected by United States copyright laws and is provided solely for the use of instructors in teaching their courses and assessing student learning. Dissemination or sale of any part of this work (including on the World Wide Web) will destroy the integrity of the work and is not permitted. The work and materials from it should never be made available to students except by instructors using the accompanying text in their classes. All recipients of this work are expected to abide by these restrictions and to honor the intended pedagogical purposes and the needs of other instructors who rely on these materials.