



Introduction to Computers, the Internet and the Web

C How to Program, 7/e



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.
- To test-drive a C application in Windows, Linux and Mac OS X.
- Some basics of the Internet and the World Wide Web.



- 1.1 Introduction
- 1.2 Computers and the Internet in Industry and Research
- 1.3 Hardware and Software
 - 1.3.1 Moore's Law
 - 1.3.2 Computer Organization
- 1.4 Data Hierarchy
- 1.5 Programming Languages
- 1.6 The C Programming Language
- 1.7 C Standard Library
- 1.8 C++ and Other C-Based Languages
- 1.9 Object Technology
- 1.10 Typical C Program Development Environment
 - 1.10.1 Phase 1: Creating a Program
 - 1.10.2 Phases 2 and 3: Preprocessing and Compiling a C Program
 - 1.10.3 Phase 4: Linking
 - 1.10.4 Phase 5: Loading
 - 1.10.5 Phase 6: Execution
 - 1.10.6 Problems That May Occur at Execution Time
 - 1.10.7 Standard Input, Standard Output and Standard Error Streams
- 1.11 Test-Driving a C Application in Windows, Linux and Mac OS X
 - 1.11.1 Running a C Application from the Windows **Command Prompt**
 - 1.11.2 Running a C Application Using GNU C with Linux
 - 1.11.3 Running a C Application Using GNU C with Mac OS X
- 1.12 Operating Systems
 - 1.12.1 Windows—A Proprietary Operating System
 - 1.12.2 Linux—An Open-Source Operating System
 - 1.12.3 Apple's Mac OS X; Apple's iOS for iPhone®, iPad® and iPod Touch® Devices
 - 1.12.4 Google's Android
- 1.13 The Internet and World Wide Web
- 1.14 Some Key Software Development Terminology
- 1.15 Keeping Up-to-Date with Information Technologies
- 1.16 Web Resources



1.1 Introduction

- ▶ C is a concise yet powerful computer programming language that's appropriate for technically oriented people with little or no programming experience and for experienced programmers to use in building substantial software systems.
- ▶ The core of the book emphasizes effective software engineering through the proven methodologies of structured programming in C and object-oriented programming in C++.



1.2 Computers and the Internet in Industry and Research

- ▶ These are exciting times in the **computer field**. Many of the most influential and successful businesses of the last two decades are technology companies, including Apple, IBM, Dell, Intel, Microsoft, Google, Amazon, Facebook, Twitter, eBay and many more.
- ▶ These companies are major employers of people who study computer science, computer engineering, information systems or related disciplines.
- ▶ Few examples of the ways in which computers are used in research and industry : Electronic health records, Human Genome Project, Cloud Computing, Robots etc



1.3 Hardware and Software

- ▶ A **computer** is a device that can perform computations and make logical decisions billions of times faster than human beings can.
- ▶ 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 etc)
- ▶ The programs that run on a computer are referred to as software.



1.3.1 Moore's Law

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



1.3.2 Computer Organization

- ▶ 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).



1.3.2 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.



1.3.3 Computer Organization (cont.)

- **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**.



1.3.3 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.



1.3.4 Computer Organization (cont.)

- **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.

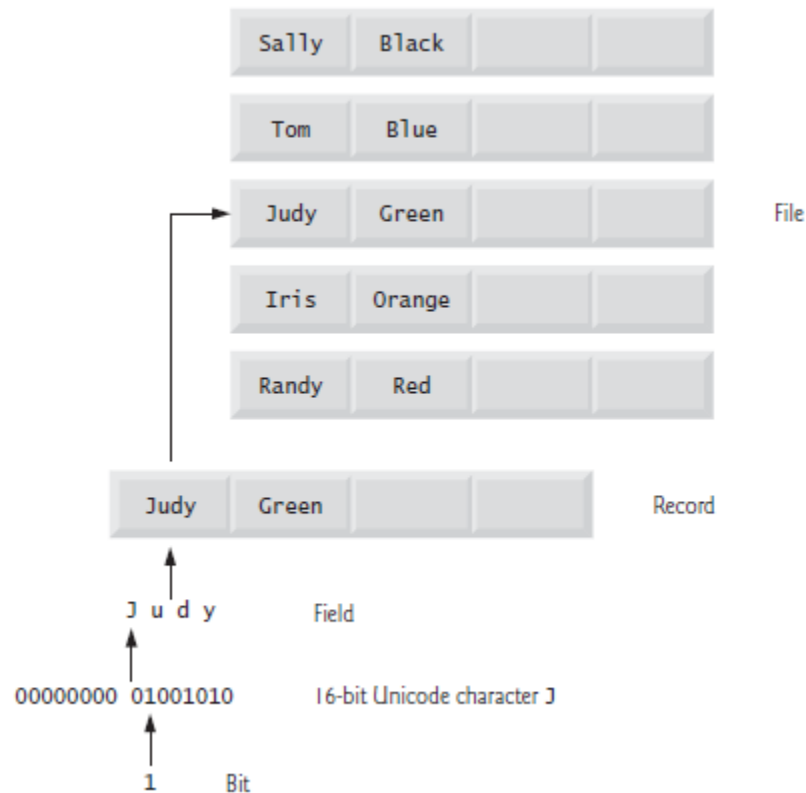


1.3.5 Computer Organization (cont.)

- **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 Data Hierarchy

- ▶ 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
- ▶ The illustrates of a portion of the data hierarchy





1.4 Data Hierarchy

- ▶ The data hierarchy's levels
 - **Bits** : The smallest data item in a computer can assume the value 0 or the value 1.
 - **Characters** : It's tedious for people to work with data in the low-level form of bits. Instead, they prefer to work with decimal digits (0–9), letters (A–Z and a–z), and special symbols (e.g., \$, @, %, &, *, (,), –, +, ", :, ? and /). Digits, letters and special symbols are known as characters.
 - **Fields** : fields are composed of characters or bytes. A field is a group of characters or bytes that conveys meaning. For example, a field consisting of uppercase and lowercase letters could be used to represent a person's name.
 - **Records** : Several related fields can be used to compose a record. In a payroll system, for example, the record for an employee might consist Name, address etc



1.4 Data Hierachy

- ▶ The data hierarchy's levels
 - **File** : A file is a group of related records. In some operating systems, a file is viewed simply as a sequence of bytes—any organization of the bytes in a file
 - **Database** : A database is an electronic collection of data that's organized for easy access and manipulation. The most popular database model is the relational database in which data is stored in simple tables.



1.5 Programming Language

- ▶ 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.



1.5 Programming Language

1. **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).



1.5 Programming Language

- ▶ 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**.



1.5 Programming Language

2. **Assembly Language** 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.



1.5 Programming Language

3. 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.



1.5 Programming Language

- ▶ A payroll program written in a high-level language might contain a statement such as
 - `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.



1.6 The C Programming Language

- ▶ 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.
- ▶ 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.



1.6 The C Programming Language

- ▶ 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.



1.7 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**.
- ▶ This textbook encourages a **building-block approach** to creating programs. Use existing pieces—this is called **software reusability**.



1.7 C Standard Library (Cont.)

- ▶ 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
- ▶ 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.



1.8 C++ and Other C-Based Languages

- ▶ 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.



1.8 C++ and Other C-Based Languages

- ▶ Several other popular C-based programming languages
 - **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 Mac OS X operating system and all iOS-based devices (such as iPods, iPhones and iPads).
 - **Visual C#** : Microsoft's three primary object-oriented programming languages are Visual Basic, Visual C++ (based on C++) and C# (based on C++ and Java, and developed for integrating the Internet and the web into computer applications).
 - **Java** : Founded by Sun Microsystems in 1991. 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



1.8 C++ and Other C-Based Languages

- ▶ Several other popular C-based programming languages
 - **PHP**: an object-oriented, open-source scripting language based on C and supported by a community of users and developers—is used by many websites including Wikipedia and Facebook
 - **JavaScript**: developed by Netscape—is the most widely used scripting language. It's primarily used to add programmability to web pages—for example, animations and interactivity with the user. It's provided with all major web browsers.



1.9 Object Technology

- ▶ Building software quickly, correctly and economically remains an elusive goal at a time when demands for new and more powerful software are soaring.
- ▶ **Objects**, or more precisely the classes objects come from, **are essentially reusable software components**.
- ▶ There are date objects, time objects, audio objects, video objects, automobile objects, people objects, etc.
- ▶ 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)..



1.9 Object Technology

- ▶ 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.
- ▶ The Automobile as an Object
 - Simple analogy → car. This enables people with little or no knowledge of how engines, braking and steering mechanisms work to drive a car easily.
- ▶ 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.



1.9 Object Technology

- ▶ **Methods and Classes**
 - we create a program unit called a **class** to house the set of methods that perform the class's tasks. 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.
- ▶ **Instantiation**
- ▶ **Reuse**
- ▶ **Messages and Method Calls**
- ▶ **Attributes and Instance Variables**
- ▶ **Encapsulation and inheritance**



1.10 Typical C Program Development Environment

- ▶ 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**.



1.10.1 Phase I : Creating a Program

- ▶ Phase 1 consists of editing a file. This is accomplished with an **editor program**.
- ▶ 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.



1.10.2 Phase 2 & 3 : Preprocessing and Compiling a C Program

- ▶ 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.



1.10.2 Phase 2 & 3 : Preprocessing and Compiling a C Program

- ▶ These manipulations usually consist of including other files in the file to be compiled and performing various text replacements.
- ▶ 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. The compiler issues an error message to help you locate and fix the incorrect statement.

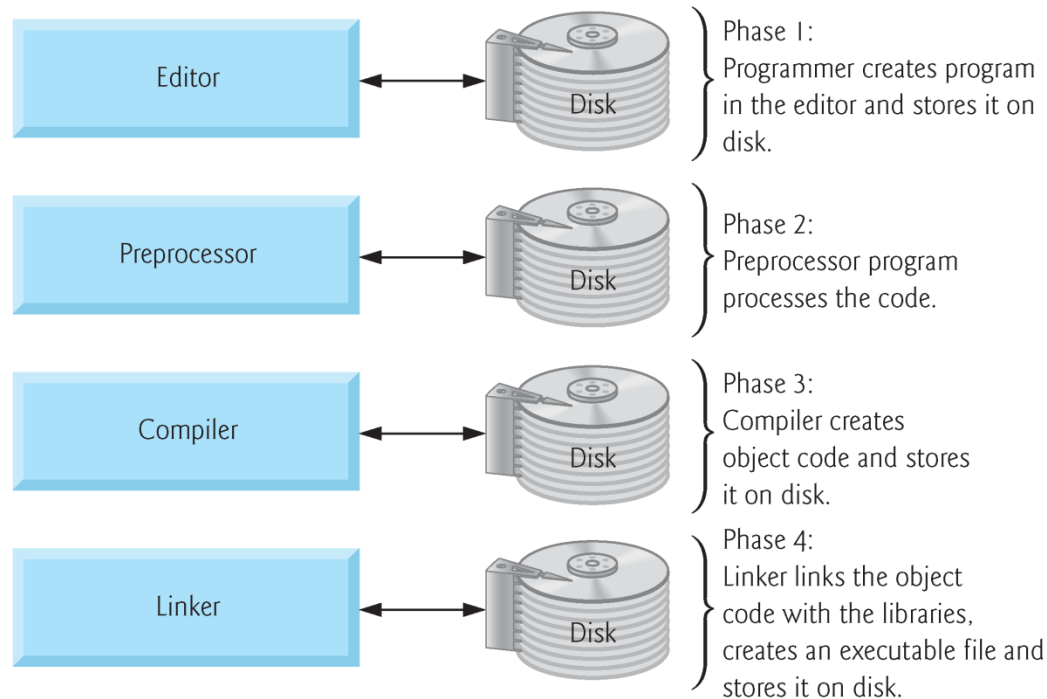


Fig. 1.1 | Typical C development environment. (Part I of 2.)

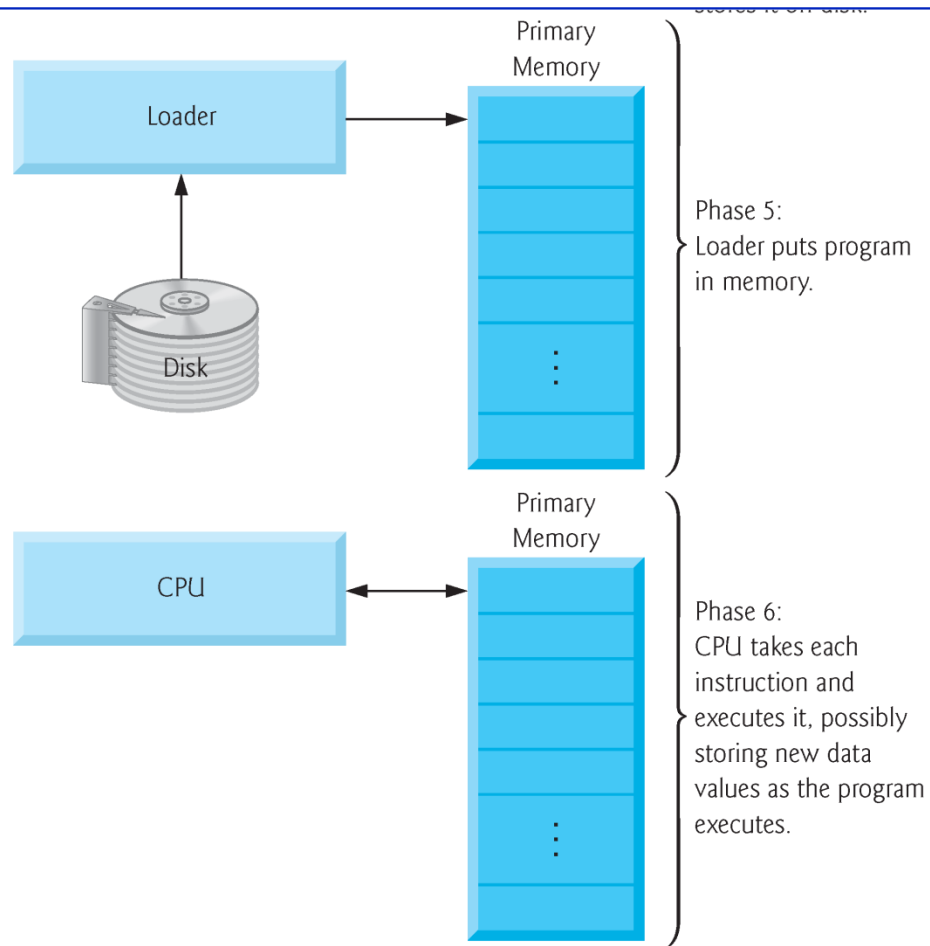


Fig. 1.1 | Typical C development environment. (Part 2 of 2.)



1.10.3 Phase 4 : Linking

- ▶ 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**).



1.10.3 Phase 4 : Linking

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



1.10.4 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.
- ▶ Finally, the computer, under the control of its CPU, **executes** the program one instruction at a time.



1.10.5 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.



1.10.6 Problem 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.



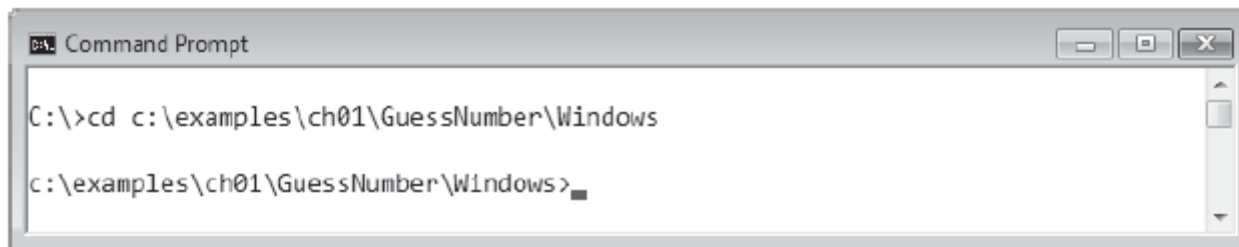
1.11 Test-Driving a C Application in Windows, Linux and Mac OS X

- ▶ We'll demonstrate running a C application using the Windows Command Prompt, a shell on Linux and a Terminal window in Mac OS X. The application runs similarly on all three platforms.
- ▶ Many development environments are available in which you can compile, build and run C applications, such as GNU C, Dev C++, Microsoft Visual C++, CodeLite, Net-Beans, Eclipse, Xcode, etc.

1.11.1 Running a C Application from the Windows Command Prompt

► https://media.pearsoncmg.com/ph/esm/deitel/C_HTTP7e/code_examples.html

1. *Checking your setup.* It's important to read the Before You Begin section at www.deitel.com/books/chtp7/ to make sure that you've copied the book's examples to your hard drive correctly.
2. *Locating the completed application.* Open a Command Prompt window. To change to the directory for the completed **GuessNumber** application, type `cd C:\examples\ch01\GuessNumber\Windows`, then press *Enter* (Fig. 1.8). The command `cd` is used to change directories.

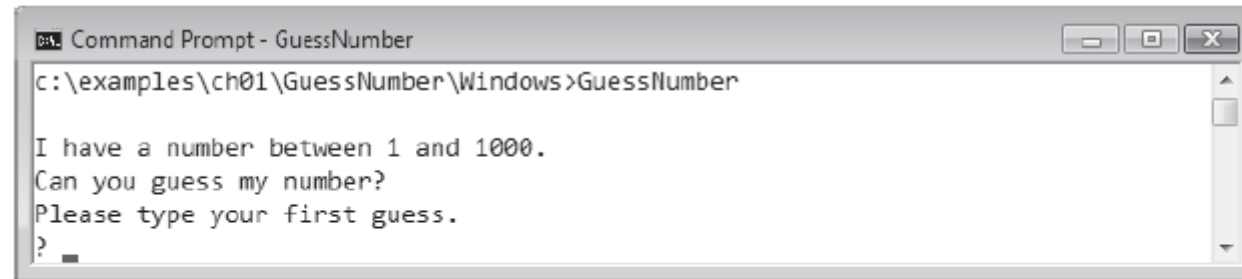


```
Command Prompt

C:\>cd c:\examples\ch01\GuessNumber\Windows

c:\examples\ch01\GuessNumber\Windows>
```

3. *Running the GuessNumber application.* Now that you are in the directory that contains the GuessNumber application, type the command GuessNumber (Fig. 1.9) and press *Enter*. [Note: GuessNumber.exe is the actual name of the application; however, Windows assumes the .exe extension by default.]




```
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.
? _
```

Fig. 1.9 | Running the GuessNumber application.

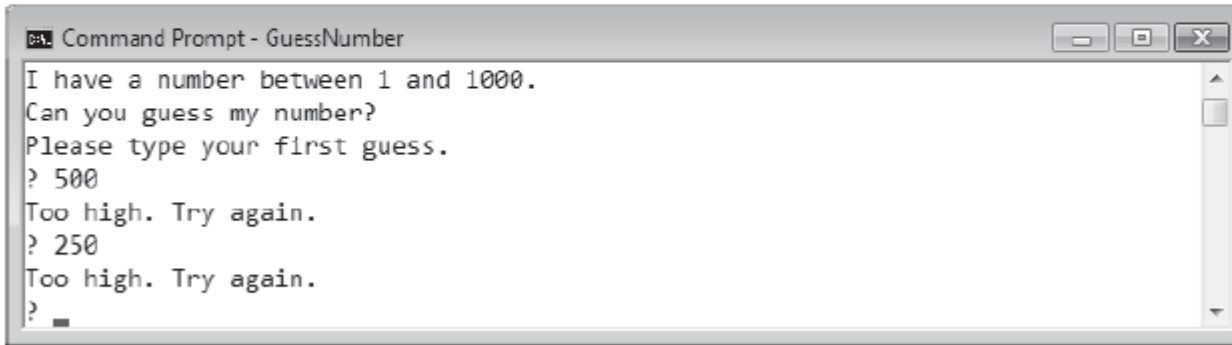
4. *Entering your first guess.* The application displays "Please type your first guess.", then displays a question mark (?) as a prompt on the next line (Fig. 1.9). At the prompt, enter 500 (Fig. 1.10).



```
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.
? _
```

Fig. 1.10 | Entering your first guess.

5. *Entering another guess.* The application displays "Too high. Try again.", meaning that the value you entered is greater than the number the application chose as the correct guess. So, you should enter a lower number for your next guess. At the prompt, enter 250 (Fig. 1.11). The application again displays "Too high. Try again.", because the value you entered is still greater than the number that the application chose.



```
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.
? _
```

Fig. 1.11 | Entering a second guess and receiving feedback.

6. *Entering additional guesses.* Continue to play the game by entering values until you guess the correct number. The application will display "Excellent! You guessed the number!" (Fig. 1.12).



7. *Playing the game again or exiting the application.* After you guess correctly, the application asks if you'd like to play another game (Fig. 1.12). At the prompt, entering 1 causes the application to choose a new number and displays the message "Please type your first guess." followed by a question-mark prompt (Fig. 1.13), so you can make your first guess in the new game. Entering 2 ends the application and returns you to the application's directory at the Command Prompt

```
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 )? █
```

Fig. 1.12 | Entering additional guesses and guessing the correct number.

8. *Close the Command Prompt window.*

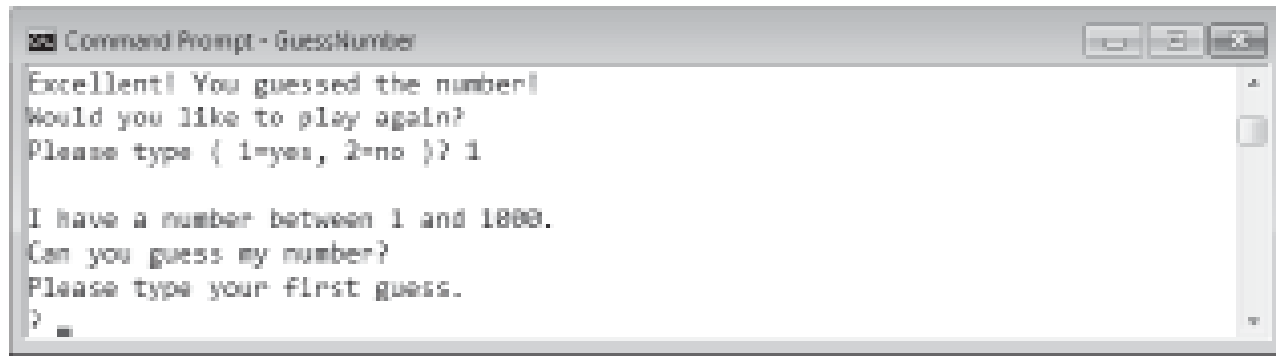


Fig. 1.13 | Playing the game again.

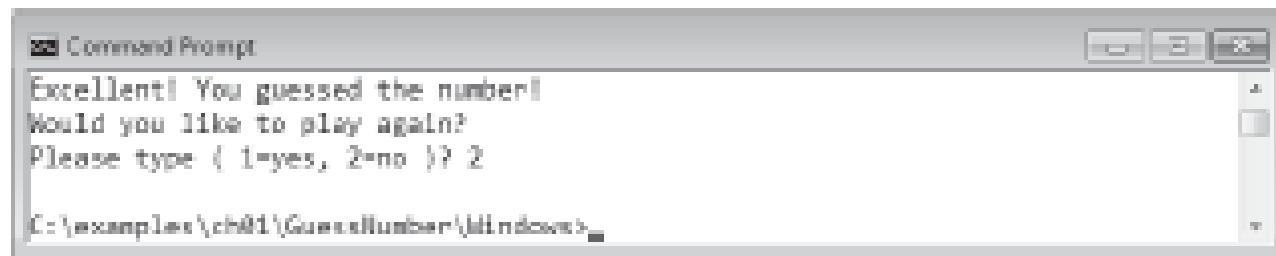


Fig. 1.14 | Exiting the game.