Chapter 1 Introduction to Computers, the Internet and the World Wide Web

C++ How to Program, Late Objects Version, 7/e

1.1 Introduction

- Download the example programs from `www.deitel.com/books/cpphtp7LOV/`
- Computers (often referred to as hardware) are controlled by software (i.e., the instructions you write to command the computer to perform actions and make decisions).
- C++ is standardized in the United States through the American National Standards Institute (ANSI) and worldwide through the efforts of the International Organization for Standardization (ISO).

1.1 Introduction (cont.)

- To keep up to date with C++ developments at Deitel & Associates, please register for our free e-mail newsletter, the *Deitel® Buzz Online*, at
  - `www.deitel.com/newsletter/subscribe.html`
- C++ and related Resource Centers at
  - `www.deitel.com/ResourceCenters.html`
- Errata and updates for this book are posted at
  - `www.deitel.com/books/cpphtp7LOV/`
- You are embarking on a challenging and rewarding path.
- As you proceed, if you have any questions, please send e-mail to
  - `deitel@deitel.com`

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.
- Today’s fastest supercomputers can perform thousands of trillions (quadrillions) of instructions per second!
- Computers process data under the control of sets of instructions called computer programs.
- 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.
- The programs that run on a computer are referred to as software.

1.3 Computer Organization

- Virtually every computer may be envisioned as divided into six logical units or sections:
  - Input unit. This “receiving” section obtains from input devices and places it at the disposal of the other units so that it can be processed.
  - 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.
  - 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. It also retains processed information until it can be placed on output devices by the output unit. Information in the memory unit is volatile. 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. It also contains the computer's decision mechanisms. 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.
  - 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
  - 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.

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.

1.7 Machine Languages, Assembly Languages and High-Level Languages

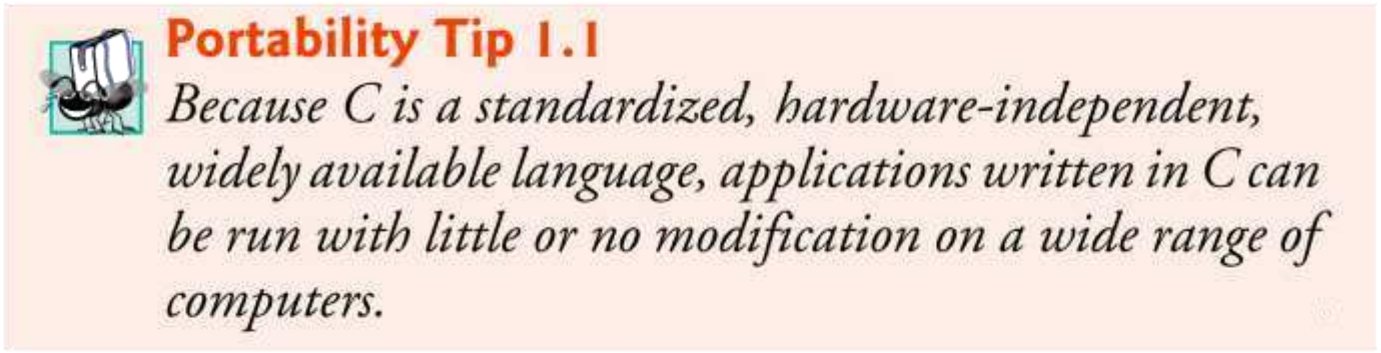
- Programmers write instructions in various programming languages, some directly understandable by computers and others requiring intermediate translation steps.
- Computer languages may be divided into three general types:
  - Machine languages
  - Assembly languages
  - High-level languages
- Any computer can directly understand only its own machine language.
  - The "natural language" of a computer and as such is defined by its hard-ware de-sign.
- Machine languages are machine dependent.
  - Too slow (for development), tedious and error prone for most programmers.

1.7 Machine Languages, Assembly Languages and High-Level Languages (cont.)

- English-like abbreviations that represent elementary operations formed the basis of assembly languages.
- Translator programs called assemblers convert assembly-language programs to machine language.
- 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.
- Interpreter programs execute high-level language programs directly (without the delay of compilation), although slower than compiled programs run.

1.8 History of C and C++

- C++ evolved from C, which evolved from two previous programming languages, BCPL and B.
- BCPL was developed in 1967 by Martin Richards as a language for writing operating systems software and compilers for operating systems.
- Ken Thompson modeled many features in B after their counterparts in BCPL and used B to create early versions of the UNIX operating system at Bell Laboratories in 1970.
- The C language was evolved from B by Dennis Ritchie at Bell Laboratories.
  - C initially became widely known as the development language of the UNIX operating system.
  - Today, most operating systems are written in C and/or C++.
  - C is available for most computers and is hardware independent.
  - It's possible to write C programs that are portable to most computers.



Portability Tip 1.1

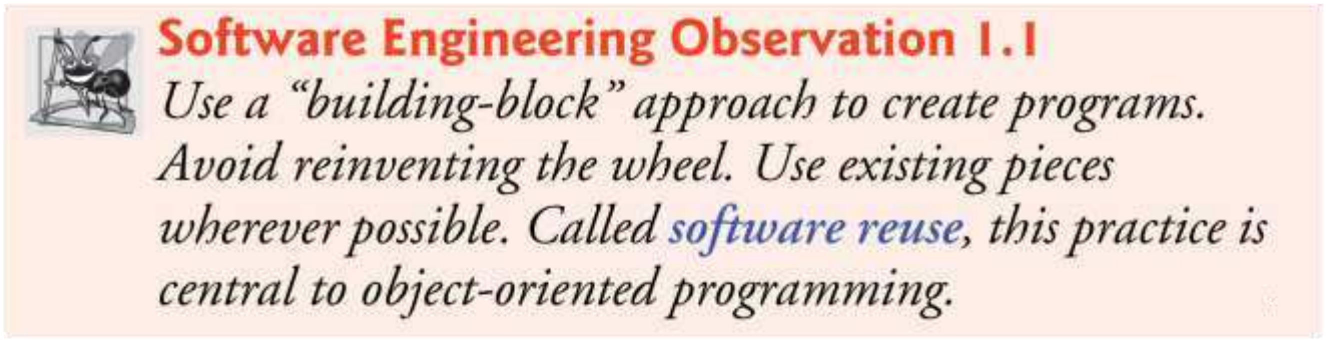
Because C is a standardized, hardware-independent, widely available language, applications written in C can be run with little or no modification on a wide range of computers.

1.8 History of C and C++ (cont.)

- C++, an extension of C, was developed by Bjarne Stroustrup in the early 1980s at Bell Laboratories.
- C++ provides capabilities for object-oriented programming.
- Objects are essentially reusable software components that model items in the real world.
- Modular, object-oriented design and implementation makes programmers much more productive than can previous popular programming techniques.
- Object-oriented programs are easier to understand, correct and modify.

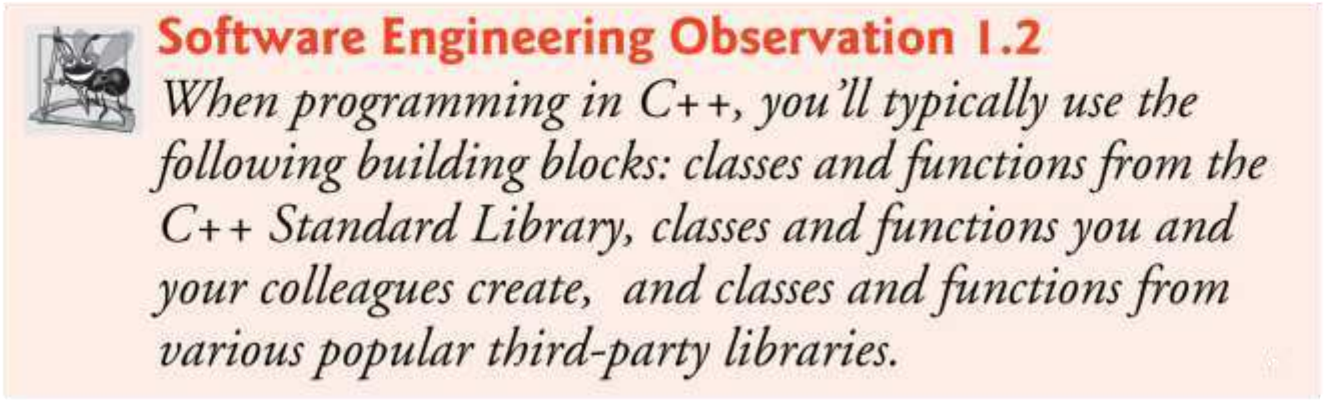
1.9 C++ Standard Library

- C++ programs consist of pieces called classes and functions.
- Most C++ programmers take advantage of the rich collections of classes and functions in the C++ Standard Library.
- Two parts to learning the C++ "world."
  - The C++ language itself, and
  - How to use the classes and functions in the C++ Standard Library.
- Many special-purpose class libraries are supplied by independent software vendors.



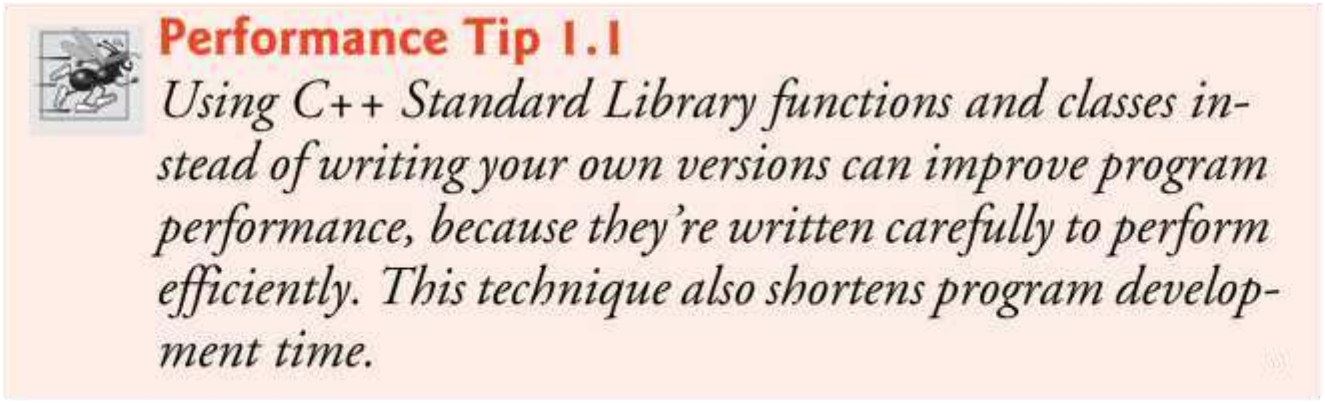
Software Engineering Observation 1.1

Use a "building-block" approach to create programs. Avoid reinventing the wheel. Use existing pieces wherever possible. Called software reuse, this practice is central to object-oriented programming.



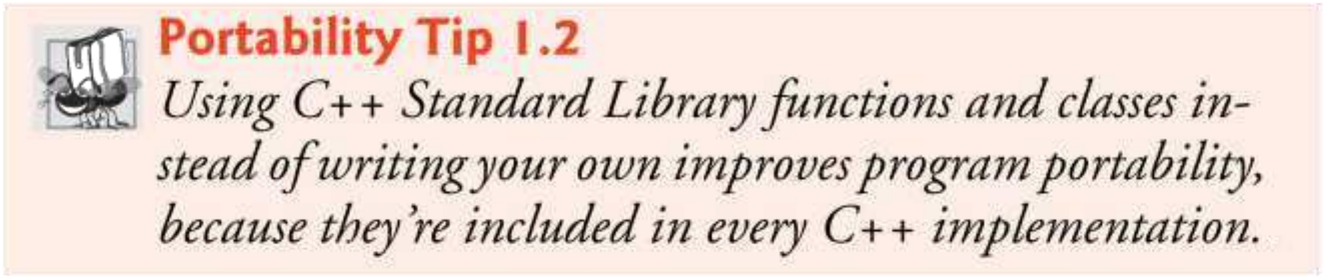
Software Engineering Observation 1.2

When programming in C++, you'll typically use the following building blocks: classes and functions from the C++ Standard Library, classes and functions you and your colleagues create, and classes and functions from various popular third-party libraries.



Performance Tip 1.1

Using C++ Standard Library functions and classes instead of writing your own versions can improve program performance, because they're written carefully to perform efficiently. This technique also shortens program development time.



Portability Tip 1.2

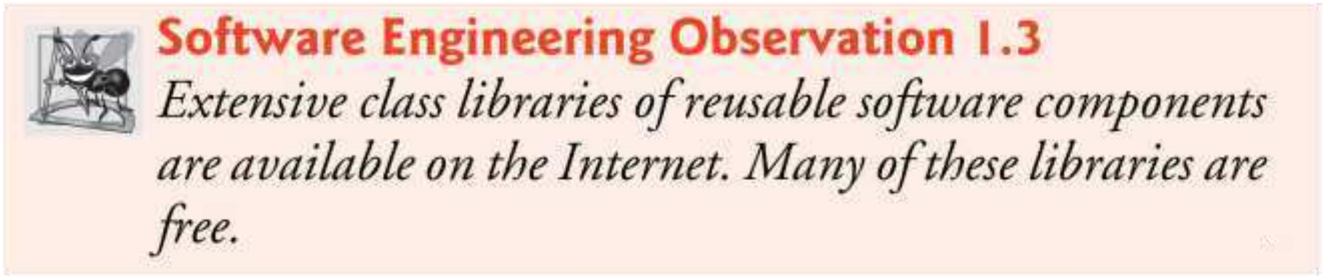
Using C++ Standard Library functions and classes instead of writing your own improves program portability, because they're included in every C++ implementation.

1.13 Key Software Trend: Object Technology

- Object technology dates back to the mid 1960s.
- The C++ programming language, developed at AT&T by Bjarne Stroustrup in the early 1980s, is based on two languages—C and Simula 67, a simulation programming language developed in Europe and released in 1967.
- C++ absorbed the features of C and added Simula's capabilities for creating and manipulating objects.
- Object technology is a packaging scheme that helps us create meaningful software units.
- 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).

1.13 Key Software Trend: Object Technology (cont.)

- 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.
- With object technology, 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 (compared to the effort that would be required to reinvent these capabilities on new projects).



Software Engineering Observation 1.3

Extensive class libraries of reusable software components are available on the Internet. Many of these libraries are free.

1.13 Key Software Trend: Object Technology (cont.)

- Key benefit of object-oriented programming is that the software is
  - more understandable
  - better organized and
  - easier to maintain, modify and debug
- 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.

1.14 Typical C++ Development Environment

- C++ systems generally consist of three parts: a program development environment, the language and the C++ Standard Library.
- C++ programs typically go through six phases: edit, preprocess, compile, link, load and execute.

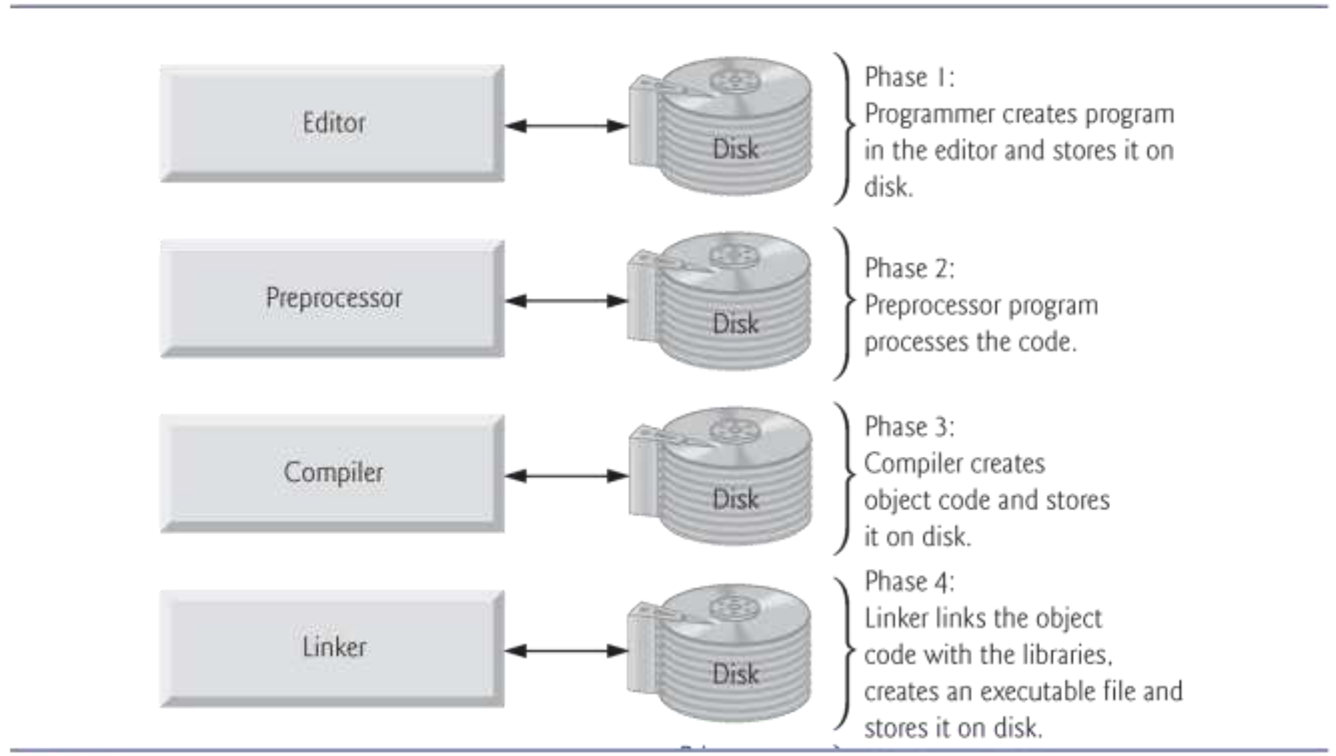

Editor
Disk
Phase 1:
Programmer creates program in the editor and stores it on disk.
Preprocessor
Disk
Phase 2:
Preprocessor program processes the code.
Compiler
Disk
Phase 3:
Compiler creates object code and stores it on disk.
Linker
Disk
Phase 4:
Linker links the object code with the libraries, creates an executable file and stores it on disk.

Fig. 1.1 | Typical C++ environment. (Part 1 of 2.)

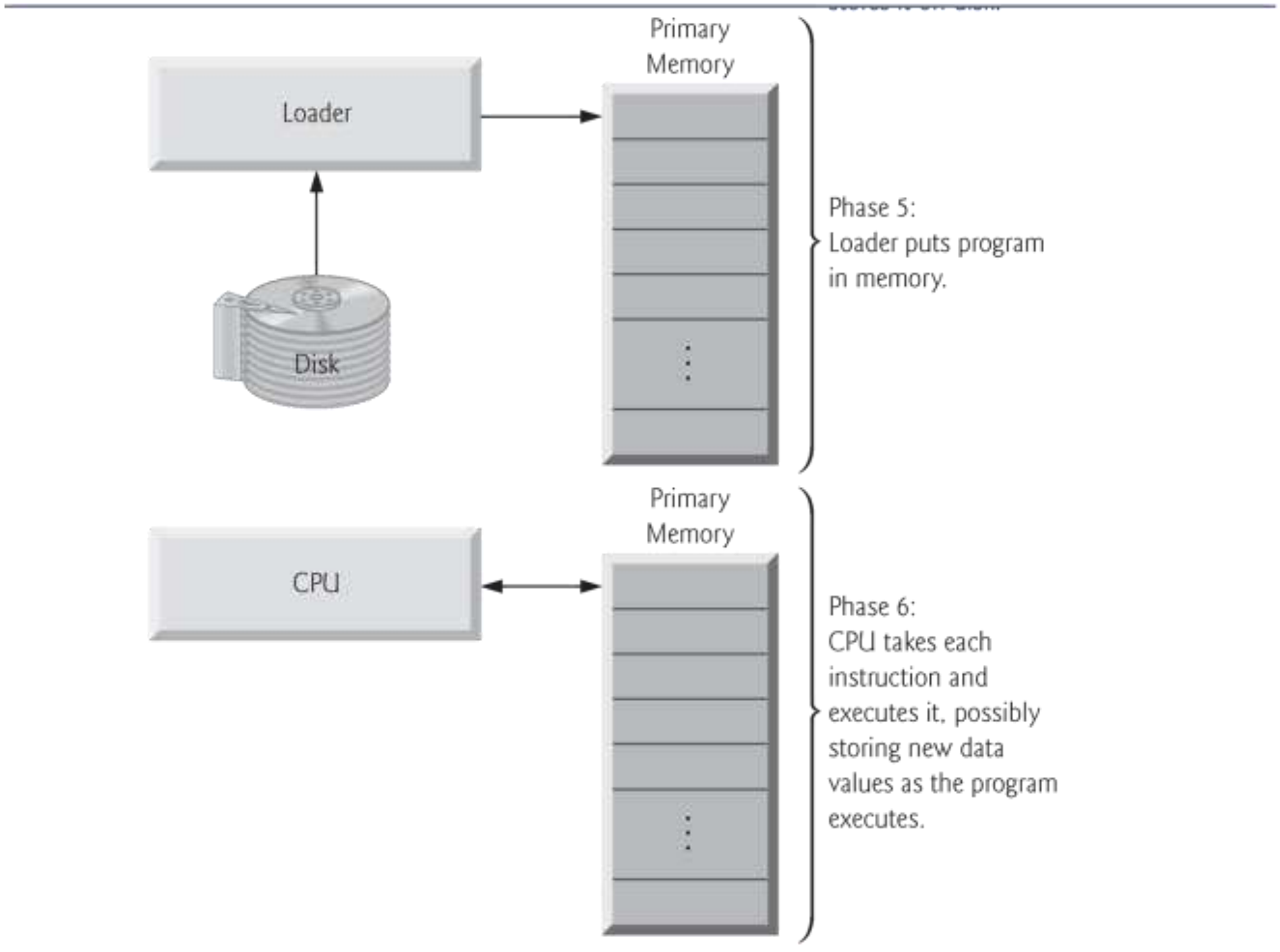

Primary Memory
Loader
Disk
Phase 5:
Loader puts program in memory.
Primary Memory
CPU
Phase 6:
CPU takes each instruction and executes it, possibly storing new data values as the program executes.

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

1.14 Typical C++ Development Environment (cont.)

- Phase 1 consists of editing a file with an editor program (normally known simply as an editor).
  - You type a C++ program (typically referred to as source code) using the editor, make corrections and save the program.
  - C++ source code filenames often end with the `.cpp`, `.cxx`, `.cc` or `.C` extensions (note that `C` is in uppercase) which indicate that a file contains C++ source code.
  - Two editors widely used on UNIX systems are `vi` and `emacs`.
  - C++ software packages such as Microsoft Visual C++ (`msdn.microsoft.com/vstudio/express/visualc/default.aspx`) have editors integrated into the programming environment.

1.14 Typical C++ Development Environment (cont.)

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

1.14 Typical C++ Development Environment (cont.)

- Phase 4 is called linking.
- The object code produced by the C++ compiler typically contains “holes” due to missing parts, such as references to functions from standard libraries.
- A linker links the object code with the code for the missing functions to produce an executable program.

1.14 Typical C++ Development Environment (cont.)

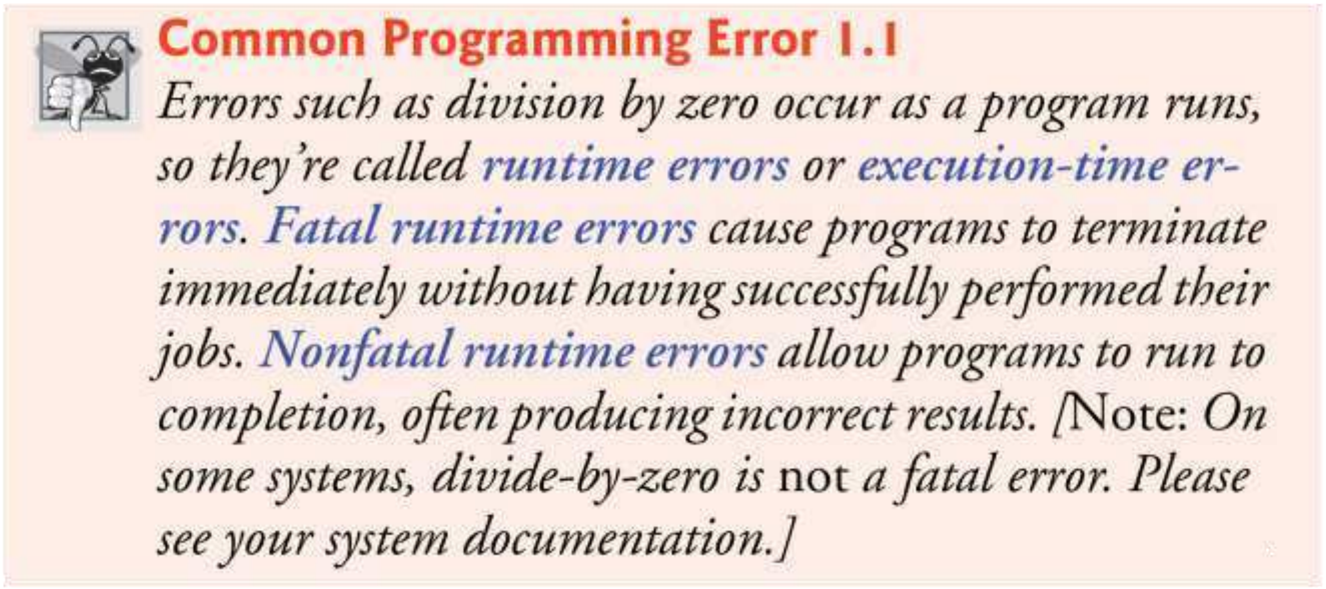
- Phase 5 is called loading.
- Before a program can be executed, it must first be placed in memory.
- This is done by the loader, which takes the executable image from disk and transfers it to memory.
- Additional components from shared libraries that support the program are also loaded.
- Finally, in Phase 6, the computer, under the control of its CPU, executes the program one instruction at a time.

1.14 Typical C++ Development Environment (cont.)

- Programs do not always work on the first try.
- Each of the preceding phases can fail because of various errors that we discuss throughout the book.
- If this occurs, you'd have to return to the edit phase, make the necessary corrections and proceed through the remaining phases again to determine that the corrections fix the problem(s).

1.14 Typical C++ Development Environment (cont.)

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

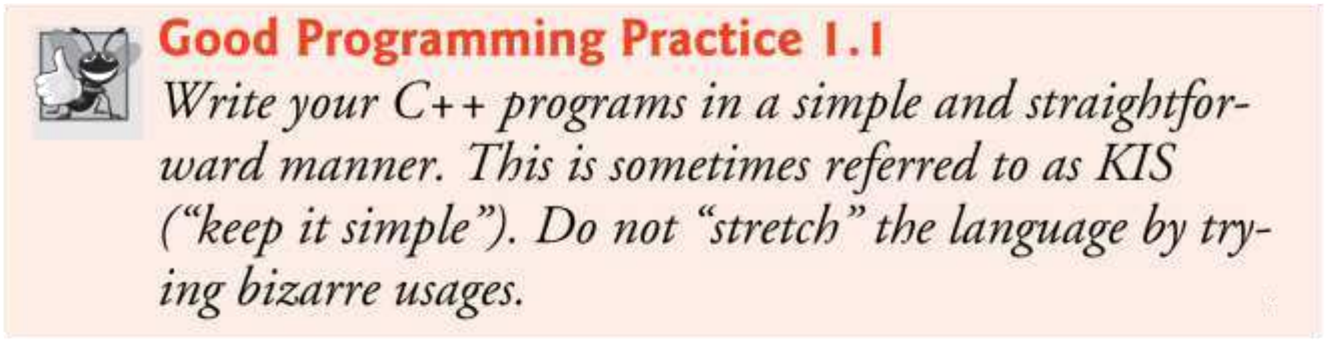


Common Programming Error 1.1

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

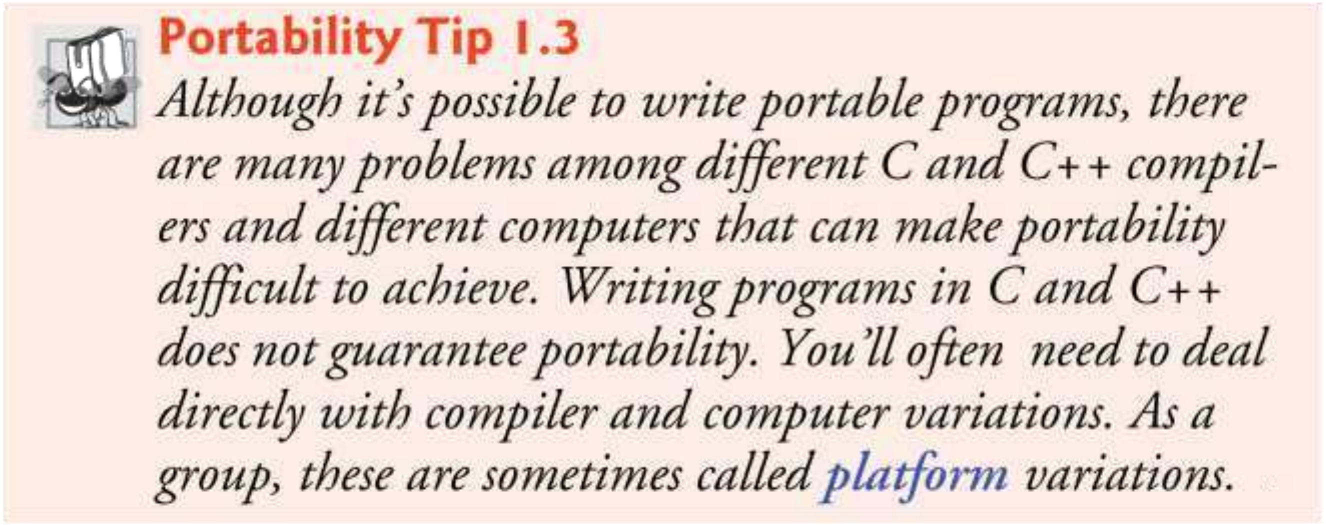
1.15 Notes About C++ and C++ *How to Program, Late objects Version 7/e*

- This book is geared for novice programmers, so we stress program *clarity*.
- If you need additional technical details on C++, you may want to read the C++ standard document, which can be ordered from ANSI at
  - `webstore.ansi.org`
- The title of the document is "Information Technology – Programming Languages – C++" and its document number is INCITS/ISO/IEC 14882-2003.
- We list many websites relating to C++ and object-oriented programming in our C++ Resource Center at `www.deitel.com/cplusplus/`



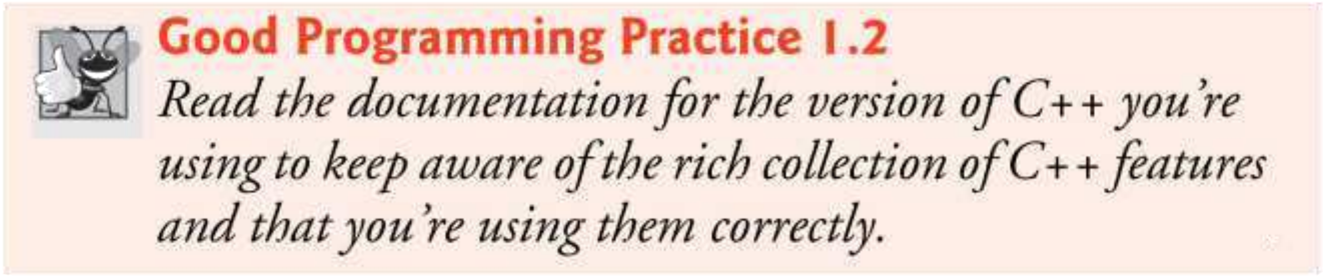
Good Programming Practice 1.1

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.



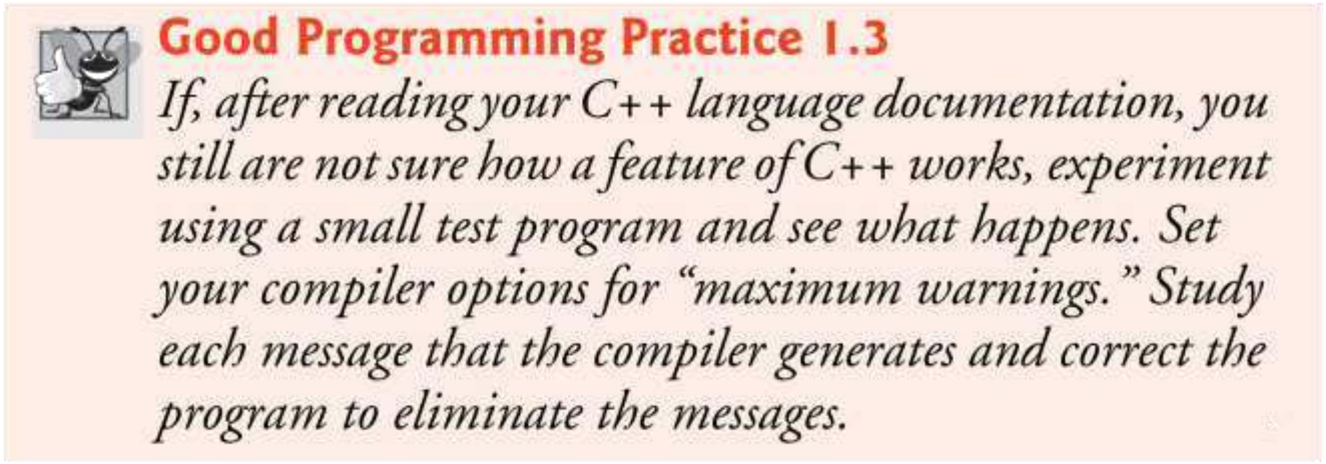
Portability Tip 1.3

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



Good Programming Practice 1.2

Read the documentation for the version of C++ you're using to keep aware of the rich collection of C++ features and that you're using them correctly.



Good Programming Practice 1.3

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