ONLINE ACCESS

Thank you for purchasing a new copy of *C++ How to Program*, Ninth Edition. Your textbook includes twelve months of prepaid access to the book's Companion Website. This prepaid subscription provides you with full access to the following student support areas:

- VideoNotes are step-by-step video tutorials specifically designed to enhance the programming concepts presented in this textbook
- Premium Web Chapters and Appendices
- Source Code

Use a coin to scratch off the coating and reveal your student access code.
Do not use a knife or other sharp object as it may damage the code.

To access the *C++ How to Program*, Ninth Edition, Companion Website for the first time, you will need to register online using a computer with an Internet connection and a web browser. The process takes just a couple of minutes and only needs to be completed once.

1. Go to http://www.pearsonhighered.com/deitel/
2. Click on Companion Website.
3. Click on the Register button.
4. On the registration page, enter your student access code* found beneath the scratch-off panel. Do not type the dashes. You can use lower- or uppercase.
5. Follow the on-screen instructions. If you need help at any time during the online registration process, simply click the Need Help? icon.
6. Once your personal Login Name and Password are confirmed, you can begin using the *C++ How to Program* Companion Website!

To log in after you have registered:

You only need to register for this Companion Website once. After that, you can log in any time at http://www.pearsonhighered.com/deitel/ by providing your Login Name and Password when prompted.

*Important: The access code can only be used once. This subscription is valid for twelve months upon activation and is not transferable. If this access code has already been revealed, it may no longer be valid. If this is the case, you can purchase a subscription by going to http://www.pearsonhighered.com/deitel/ and following the on-screen instructions.

Deitel® Series Page

How To Program Series

Android How to Program
C How to Program, 7/E
C++ How to Program, 9/E
C++ How to Program, Late Objects Version,
7/E Java™ How to Program, 9/E
Java™ How to Program, Late Objects Version,
8/E Internet & World Wide Web How to
Program, 5/E Visual Basic® 2012 How to
Program
Visual C#® 2012 How to Program, 3/E
Visual C++® 2008 How to Program,
2/E Small Java™ How to Program, 6/E
Small C++ How to Program, 5/E

Simply Series

Simply C++: An App-Driven Tutorial
Approach Simply Java™ Programming: An
App-Driven Tutorial Approach
Simply Visual Basic® 2010, 4/E: An
App-Driven Tutorial Approach

CourseSmart Web Books

www.deitel.com/books/CourseSmart/ C++
How to Program, 7/E, 8/E & 8/E Simply
C++: An App-Driven Tutorial Approach
Java™ How to Program, 7/E, 8/E & 9/E
Simply Visual Basic 2010: An App-Driven
Approach, 4/E
Visual Basic® 2012 How to Program
Visual Basic® 2010 How to Program
Visual C#® 2012 How to Program,
5/E Visual C#® 2010 How to
Program, 4/E

Deitel® Developer Series

C++ for Programmers, 2/E
Android for Programmers: An
App-Driven Approach
C# 2010 for Programmers, 3/E
Dive Into® iOS 6: An App-Driven Approach
iOS 6 for Programmers: An App-Driven
Approach Java™ for Programmers, 2/E
JavaScript for Programmers

LiveLessons Video Learning Products

www.deitel.com/books/LiveLessons/
Android® App Development Fundamentals
C++ Fundamentals
C# Fundamentals
iOS 6 App Development
Fundamentals Java™ Fundamentals
JavaScript Fundamentals
Visual Basic Fundamentals

To receive updates on Deitel publications, Resource Centers, training courses, partner offers and more, please register for the free *Deitel® Buzz Online* e-mail newsletter at:

www.deitel.com/newsletter/subscribe.html

and join the Deitel communities on Twitter®

@deitel

Facebook®

facebook.com/DeitelFan

and Google+

gplus.to/deitel

To communicate with the authors, send e-mailto:

deitel@deitel.com

For information on government and corporate *Dive-Into® Series* on-site seminars offered by Deitel & Associates, Inc. worldwide, visit:

www.deitel.com/training/

or write to

deitel@deitel.com

For continuing updates on Prentice Hall/Deitel publications visit:

www.deitel.com
www.pearsonhighered.com/deitel/

Visit the Deitel Resource Centers that will help you master programming languages, software development, Android and iPhone/iPad app development, and Internet- and web-related topics:
www.deitel.com/ResourceCenters.html

Paul Deitel

Deitel & Associates, Inc.

Harvey Deitel

Deitel & Associates, Inc.

Vice President and Editorial Director: ***Marcia J. Horton***
Executive Editor: ***Tracy Johnson***
Associate Editor: ***Carole Snyder***
Director of Marketing: ***Christy Lesko***
Marketing Manager: ***Yezan Alayan***
Marketing Assistant: ***Jon Bryant***
Director of Production: ***Erin Gregg***
Managing Editor: ***Scott Disanno***
Associate Managing Editor: ***Robert Engelhardt***
Operations Specialist: ***Lisa McDowell***
Art Director: ***Anthony Gemmellaro***
Cover Design: ***Abbey S. Deitel, Harvey M. Deitel, Anthony Gemmellaro***
Cover Photo Credit: ***© Shutterstock/Sean Gladwell***
Media Project Manager: ***Renata Butera***

Credits and acknowledgments borrowed from other sources and reproduced, with permission, in this textbook appear on page vi.

The authors and publisher of this book have used their best efforts in preparing this book. These efforts include the development, research, and testing of the theories and programs to determine their effectiveness. The authors and publisher make no warranty of any kind, expressed or implied, with regard to these programs or to the documentation contained in this book. The authors and publisher shall not be liable in any event for incidental or consequential damages in connection with, or arising out of, the furnishing, performance, or use of these programs.

Library of Congress Cataloging-in-Publication Data on file.

PEARSON

10 9 8 7 6 5 4 3 2 1

ISBN-10: 0-13-337871-3
ISBN-13: 978-0-13-337871-9

In memory of Dennis Ritchie, creator of the C programming language— one of the key languages that inspired C++.

Paul and Harvey Deitel

Trademarks

Chapters 24–26 and Appendices F–K are PDF documents posted online at the book's Companion Website, which is accessible from www.pearsonhighered.com/deitel.

10 Operator Overloading; Class string 433

11 Object-Oriented Programming: Inheritance 482

12 Object-Oriented Programming: Polymorphism 517

Online Chapters and Appendices

Chapters 24–26 and Appendices F–K are PDF documents posted online at the book's Companion Website, which is accessible from www.pearsonhighered.com/deitel.

25 ATM Case Study, Part 1:

Object-Oriented Design with the UML 25-1

26 ATM Case Study, Part 2:

Implementing an Object-Oriented Design 26-1

F C Legacy Code Topics F-1

G UML 2: Additional Diagram Types G-1

H Using the Visual Studio Debugger H-1

[*Note:* The test drives for Windows and Linux are in Chapter 1.]

"The chief merit of language is clearness …"
—Galen

Welcome to the C++ computer programming language and *C++ How to Program, Ninth Edition*. This book presents leading-edge computing technologies. It's appropriate for in troductory course sequences based on the curriculum recommendations of two key pro fessional organizations—the ACM and the IEEE. If you haven't already done so, please read the back cover and inside back cover—these capture the essence of the book concise ly. In this Preface we provide more detail for students, instructors and professionals.

At the heart of the book is the Deitel signature *live-code approach*—we present con cepts in the context of complete working programs followed by sample executions, rather than in code snippets. Read the online Before You Begin section (www.deitel.com/books/cpphtp9/cpphtp9_BYB.pdf) to learn how to set up your Linux-based, Windows based or Apple OS X-based computer to run the hundreds of code examples. All the source code is available at www.deitel.com/books/cpphtp9 and www.pearsonhighered.com/ deitel. Use the source code we provide to *run each program* as you study it.

We believe that this book and its support materials will give you an informative, chal lenging and entertaining introduction to C++. As you read the book, if you have questions, we're easy to reach at deitel@deitel.com—we'll respond promptly. For book updates, visit www.deitel.com/books/cpphtp9, join our social media communities on Facebook (www.deitel.com/DeitelFan), Twitter (@deitel), Google+ (gplus.to/deitel) and LinkedIn (bit.ly/DeitelLinkedIn), and subscribe to the *Deitel® Buzz Online* newsletter (www.deitel.com/newsletter/subscribe.html).

C++11 Standard

The new C++11 standard, published in 2011, motivated us to write *C++ How to Program, 9/e*. Throughout the book, each new C++11 feature is marked with the "11" icon you see here in the margin. These are some of the key C++11 features of this new edition:

- ***Conforms to the new C++11 standard.*** Extensive coverage of the new C++11 features (Fig. 1).
- ***Code thoroughly tested on three popular industrial-strength C++11 compilers.*** We tested the code examples on GNU™ C++ 4.7, Microsoft® Visual C++® 2012 and Apple® LLVM in Xcode® 4.5.
- ***Smart pointers.*** Smart pointers help you avoid dynamic memory management errors by providing additional functionality beyond that of built-in pointers. We discuss unique_ptr in Chapter 17, and shared_ptr and weak_ptr in Chapter 24.

all_of algorithm
any_of algorithm
array container
auto for type inference
begin/end functions
cbegin/cend container member functions
Compiler fix for >> in template types
copy_if algorithm
copy_n algorithm
crbegin/crend container member functions
decltype
Default type arguments in function templates
defaulted member functions
Delegating constructors
deleted member functions
explicit conversion operators
final classes
final member functions
find_if_not algorithm
forward_list container
Immutable keys in associative containers
In-class initializers

Inheriting base-class constructors insert container member functions return iterators
is_heap algorithm
is_heap_until algorithm
Keywords new in C++11 Lambda expressions
List initialization of key–value pairs
List initialization of pair objects
List initialization of return values List initializing a dynamically allocated array
List initializing a vector List initializers in constructor calls
long long int type
min and max algorithms with initializer_list parameters
minmax algorithm
minmax_element algorithm
move algorithm
Move assignment operators move_backward algorithm Move constructors

noexcept
Non-deterministic random number generation
none_of algorithm
Numeric conversion functions
nullptr
override keyword
Range-based for statement
Regular expressions
Rvalue references
Scoped enums
shared_ptr smart pointer
shrink_to_fit vector/deque member function
Specifying the type of an enum's constants
static_assert objects for file names
string objects for file names
swap non-member function
Trailing return types for functions
tuple variadic template
unique_ptr smart pointer
Unsigned long long int
weak_ptr smart pointer

Fig. 1 | A sampling of C++11 features in *C++ How to Program, 9/e*.

- ***Earlier coverage of Standard Library containers, iterators and algorithms, enhanced with C++11 capabilities.*** We moved the treatment of Standard Library containers, iterators and algorithms from Chapter 22 in the previous edition to Chapters 15 and 16 and enhanced it with additional C++11 features. The vast majority of your data structure needs can be fulfilled by *reusing* these Standard Library capabilities. We'll show you how to build your own *custom* data structures in Chapter 19.

- ***Online Chapter 24, C++11: Additional Topics.*** In this chapter, we present additional C++11 topics. The new C++11 standard has been available since 2011, but not all C++ compilers have fully implemented the features. If all three of our key compilers already implemented a particular C++11 feature at the time we wrote this book, we generally integrated a discussion of that feature into the text with a live-code example. If any of these compilers had *not* implemented that feature, we included a bold italic heading followed by a brief discussion of the feature. Many of those discussions are expanded in online Chapter 24 as the features are implemented. This chapter includes discussions of regular expressions, `shared_ptr` and `weak_ptr` smart pointers, move semantics and more.

- ***Random Number generation, simulation and game playing.*** To help make programs more secure, we've added a treatment of C++11's new non-deterministic random-number generation capabilities.

Object-Oriented Programming

- ***Early-objects approach.*** The book introduces the basic concepts and terminology of object technology in Chapter 1. You'll develop your first customized classes and objects in Chapter 3. Presenting objects and classes early gets you "thinking about objects" immediately and mastering these concepts more thoroughly.[1]

- ***C++ Standard Library* `string`.** C++ offers *two* types of strings—`string` class objects (which we begin using in Chapter 3) and C strings. We've replaced most oc currences of C strings with instances of C++ class `string` to make programs more robust and eliminate many of the security problems of C strings. We continue to discuss C strings later in the book to prepare you for working with thelegacy code that you'll encounter in industry. In new development, you should favor `string` objects.

- ***C++ Standard Library* `array`.** Our primary treatment of arrays now uses the Standard Library's `array` class template instead of built-in, C-style, pointer-based arrays. We still cover built-in arrays because they remain useful in C++ and so that you'll be able to read legacy code. C++ offers *three* types of arrays—`array`s and `vector`s (which we start using in Chapter 7) and C-style, pointer-based arrays which we discuss in Chapter 8. As appropriate, we use class template `array` instead of C arrays throughout the book. In new development, you should favor class template `array` objects.

- ***Crafting valuable classes.*** A key goal of this book is to prepare you to build valu able classes. In the Chapter 10 case study, you'll build your own custom Array class, then in the Chapter 18 exercises you'll convert it to a class template. You'll truly appreciate the class concept. Chapter 10 begins with a test-drive of class template string so you can see an elegant use of operator overloading before you implement your own customized class with overloaded operators.

- ***Case studies in object-oriented programming.*** We provide case studies that span multiple sections and chapters and cover the software development lifecycle. These include the GradeBook class in Chapters 3–7, the Time class in Chapter 9 and the Employee class in Chapters 11–12. Chapter 12 contains a detailed diagram and explanation of how C++ can implement polymorphism, virtual functions and dynamic binding "under the hood."

- ***Optional case study: Using the UML to develop an object-oriented design and C++ implementation of an ATM.***The UML™ (Unified Modeling Language™) is the

[1] For courses that require a late-objects approach, consider *C++ How to Program, Late Objects Version*, which begins with six chapters on programming fundamentals (including two on control statements) and continues with seven chapters that gradually introduce object-oriented programming concepts.

industry-standard graphical language for modeling object-oriented systems. We introduce the UML in the early chapters. Online Chapters 25 and 26 include an *optional* case study on object-oriented design using the UML. We design and im plement the software for a simple automated teller machine (ATM). We analyze a typical requirements document that specifies the system to be built. We determine the classes needed to implement that system, the attributes the classes need to have, the behaviors the classes need to exhibit and wespecify how the classes must interact with one another to meet the system requirements. From the design we produce a complete C++ implementation. Students often report that the case study helps them "tie it all together" and truly understand object orientation.

- ***Exception handling.*** We integrate basic exception handling *early* in the book. In structors can easily pull more detailed material forward from Chapter 17, Excep tion Handling: A Deeper Look.

- ***Custom template-based data structures.*** We provide a rich multi-chapter treat ment of data structures—see the Data Structures module in the chapter depen dency chart (Fig. 6).

- ***Three programming paradigms.*** We discuss *structured programming*, *object-orient ed programming* and *generic programming*.

Pedagogic Features

- ***Rich coverage of C++ fundamentals.***We include a clear two-chapter treatment of control statements and algorithm development.

- ***Chapter 2 provides a simple introduction to C++ programming.***

- ***Examples.*** We include a broad range of example programs selected from comput er science, business, simulation, game playing and other topics (Fig. 2).

Array class case study
Author class
Bank account program
Bar chart printing program
BasePlusCommissionEmployee class
Binary tree creation and traversal
BinarySearch test program
Card shuffling and dealing
ClientData class
CommissionEmployee class
Comparing strings
Compilation and linking process
Compound interest calculations with for
Converting string objects to C strings
Counter-controlled repetition

Craps dice game simulation
Credit inquiry program
Date class
Downcasting and runtime type information
Employee class
explicit constructor
fibonacci function
fill algorithms
Function-template specializations of function template printArray
generate algorithms
GradeBook Class
Initializing an array in a declaration
Input from an istringstream object
Iterative factorial solution

Fig. 2 | A sampling of the book's examples. (Part 1 of 2.)

Lambda expressions
Linked list manipulation
map class template
Mathematical algorithms of the Standard
Library maximum function template
Merge sort program
multiset class template
new throwing bad_alloc on failure
PhoneNumber class
Poll analysis program
Polymorphism demonstration
Preincrementing and postincrementing
priority_queue adapter class
queue adapter class
Random-access files
Random number generation
Recursive function factorial
Rolling a six-sided die 6,000,000 times
SalariedEmployee class

SalesPerson class
Searching and sorting algorithms of the Standard Library
Sequential files
set class template
shared_ptr program
stack adapter class
Stack class
Stack unwinding
Standard Library string class program
Stream manipulator showbase
string assignment and concatenation
string member function substr
Summing integers with the for statement
Time class
unique_ptr object managing dynamically allocated memory
Validating user input with regular expressions
vector class template

Fig. 2 | A sampling of the book's examples. (Part 2 of 2.)

- ***Audience.*** The examples are accessible to computer science, information technology, software engineering and business students in novice-level and intermediate level C++ courses. The book is also used by professional programmers.
- ***Self-Review Exercises and Answers.*** Extensive self-review exercises *and* answers

are included for self-study.

- ***Interesting, entertaining and challenging exercises.*** Each chapter concludes with a substantial set of exercises, including simple recall of important terminology and concepts, identifying the errors in code samples, writing individual program statements, writing small portions of C++ classes and member and non-member functions, writing complete programs and implementing major projects. Figure 3 lists a sampling of the book's exercises, including our *Making a Differ ence* exercises, which encourage you to use computers and the Internet to research and solve significant social problems. We hope you'll approach these exercises with *your own* values, politics and beliefs.

Airline Reservations
System Advanced String
Manipulation Exercises

Bubble Sort
Build Your Own Compiler
Build Your Own Computer
Calculating Salaries

CarbonFootprint
Abstract Class:
Polymorphism

Fig. 3 | A sampling of the book's exercises. (Part 1 of 2.)

Dice Rolling

Hierarchy

Card Shuffling and Dealing
Computer-Assisted
Instruction
Computer-Assisted Instruc tion: Difficulty Levels
Computer-Assisted
Instruction:
Monitoring Student
Performance
Computer-Assisted
Instruction: Reducing
Student Fatigue
Computer-Assisted
Instruction: Varying the
Types of Problems
Cooking with Healthier Ingre dients
Craps Game Modification
Credit Limits
Crossword Puzzle
Generator
Cryptograms
De Morgan's Laws

Eight Queens
Emergency Response
Enforcing Privacy with
Cryptography
Facebook User Base
Growth Fibonacci Series
Gas Mileage
Global Warming Facts
Quiz Guess the Number
Game Hangman Game
Health Records Knight's
Tour Limericks
Maze Traversal:
Generating Mazes
Randomly
Morse Code
Payroll System
Modification
Peter Minuit Problem
Phishing Scanner
Pig Latin
Polymorphic Banking
Program Using Account

Pythagorean Triples
Salary Calculator
Sieve of
Eratosthenes Simple
Decryption
Simple Encryption
SMS Language
Spam Scanner
Spelling Checker
Target-Heart-Rate
Calculator
Tax Plan
Alternatives; The
"Fair Tax"
Telephone number
word generator
"The Twelve Days
of Christmas" Song
Tortoise and the
Hare Simulation
Towers of Hanoi
World Population Growth

Fig. 3 | A sampling of the book's exercises. (Part 2 of 2.)

- ***Illustrations and figures.*** Abundant tables, line drawings, UML diagrams, programs and program outputs are included. A sampling of the book's drawings and diagrams is shown in (Fig. 4).

Main text drawings and diagrams

Data hierarchy
Compilation and linking process for multiple source file programs
Order in which a second-degree polynomial is evaluated
GradeBook class diagrams if single-selection statement activity diagram
if...else double-selection

statement activity diagram
while repetition statement UML activity diagram
for repetition statement UML activity diagram
do...while repetition statement UML activity diagram
switch multiple-selection statement activity diagram
C++'s single-entry/single-exit sequence, selection and repetition statements

Pass-by-value and pass-by-reference analysis of a program
Inheritance hierarchy diagrams
Function-call stack and activation records
Recursive calls to function fibonacci
Pointer arithmetic diagrams
CommunityMember
Inheritance hierarchy
Shape inheritance hierarchy

Fig. 4 | A sampling of the book's drawings and diagrams. (Part 1 of 2.)

class objects linked together

represented graphically

public, protected and private inheritance
Employee hierarchy UML class diagram
How virtual function calls work
Stream-I/O template hierarchy Two self-referential

Graphical representation of a list
Operation insertAtFront represented graphically
Operation insertAtBack represented graphically
Operation removeFromFront

Operation removeFromBack represented graphically
Circular, singly linked list
Doubly linked list
Circular, doubly linked list
Graphical representation of a binary tree

ATM Case Study drawings and diagrams

Use case diagram for the ATM system from the User's per spective
Class diagram showing an asso ciation among classes
Class diagram showing compo sition relationships
Class diagram for the ATM system model
Classes with attributes
State diagram for the ATM
Activity diagram for a Balance Inquiry transaction

Activity diagram for a Withdrawal transaction
Classes in the ATM system with attributes and operations
Communication diagram of the ATM executing a bal ance inquiry
Communication diagram for executing a balance inquiry
Sequence diagram that models a Withdrawal executing
Use case diagram for a modified version of our ATM system that also allows users to

transfer money between accounts
Class diagram showing composition relationships of a class
Car
Class diagram for the ATM system model including class Deposit
Activity diagram for a Deposit transaction
Sequence diagram that models a Deposit executing

Fig. 4 | A sampling of the book’s drawings and diagrams. (Part 2 of 2.)

- ***VideoNotes.*** The Companion Website includes many hours of *VideoNotes* in which co-author Paul Deitel explains in detail key programs in the core chapters. We’ve created a jump table that maps each VideoNote to the corresponding figures in the book (www.deitel.com/books/cpphtp9/jump_table.pdf).

Other Features

- ***Pointers.*** We provide thorough coverage of the built-in pointer capabilities and the intimate relationship among built-in pointers, C strings and built-in arrays.
- ***Visual presentation of searching and sorting, witha simple explanation of Big O.***
- ***Printed book contains core content; additional content is online.*** A few online chapters and appendices are included. These are available in searchable PDF for mat on the book’s password-protected Companion Website—see the access card information on the inside front cover.
- ***Debugger appendices.*** We provide three debugger appendices on the book’s Companion Website—Appendix H, Using the Visual Studio Debugger, Appendix I, Using the GNU C++ Debugger and Appendix J, Using the Xcode Debugger.

Secure C++ Programming

It’s difficult to build industrial-strength systems that stand up to attacks from viruses, worms, and other forms of “malware.” Today, via the Internet, such attacks can be instan taneous and global in scope. Building security into software from the beginning of the de velopment cycle can greatly reduce vulnerabilities.

The CERT® Coordination Center (www.cert.org) was created to analyze and respond promptly to attacks. CERT—the Computer Emergency Response Team—is a government-funded organization within the Carnegie Mellon University Software Engineering Institute™. CERT publishes and promotes secure coding standards for various popular programming languages to help software developers implement industrial strength systems that avoid the programming practices that leave systems open to attacks.

We’d like to thank Robert C. Seacord, Secure Coding Manager at CERT and an adjunct professor in the Carnegie Mellon University School of Computer Science. Mr. Sea cord was a technical reviewer for our book, *C How to Program, 7/e*, where he scrutinized our C programs from a security standpoint, recommending that we adhere to the *CERT C Secure Coding Standard.*

We’ve done the same for *C++ How to Program, 9/e*, adhering to the *CERT C++ Secure Coding Standard*, which you can find at:

www.securecoding.cert.org

We were pleased to discover that we’ve already been recommending many of these coding practices in our books. We upgraded our code and discussions to conform to these practices, as appropriate for an introductory/intermediate-level textbook. If you'll be building industrial-strength C++ systems, consider reading *Secure Coding in C and C++, Second Edition* (Robert Seacord, Addison-Wesley Professional).

Online Content

The book's Companion Website, which is accessible at

www.pearsonhighered.com/deitel

(see the inside front cover of the book for an access code) contains the following chapters and appendices in searchable PDF format:

- Chapter 24, C++11 Additional Topics
- Chapter 25, ATM Case Study, Part 1: Object-Oriented Design with the UML •

Chapter 26, ATM Case Study, Part 2: Implementing an Object-Oriented Design •

Appendix F, C Legacy Code Topics

- Appendix G, UML 2: Additional Diagram Types
- Appendix H, Using the Visual Studio Debugger
- Appendix I, Using the GNU C++ Debugger
- Appendix J, Using the Xcode Debugger
- Appendix K, Test Driving a C++ Program on Mac OS X. (The test drives for Windows and Linux are in Chapter 1.)

The Companion Website also includes:

- Extensive *VideoNotes*—watch and listen as co-author Paul Deitel discusses key code examples in the core chapters of the book.
- Building Your Own Compiler exercise descriptions from Chapter 19 (posted at the Companion Website and at www.deitel.com/books/cpphtp9).
- Chapter 1 test-drive for Mac OS X.

Dependency Chart

The chart in Fig. 6 shows the dependencies among the chapters to help instructors plan their syllabi. *C++ How to Program, 9/e* is appropriate for CS1 and many CS2 courses. The chart shows the book's modular organization.

Teaching Approach

C++ How to Program, 9/e, contains a rich collection of examples. We stress program clarity and concentrate on building well-engineered software.

Live-code approach. The book is loaded with "live-code" examples—most new concepts are presented in *complete working C++ applications*, followed by one or more executions showing program inputs and outputs. In the few cases where we use a code snippet, to ensure that it's correct we tested it in a complete working program, then copied and pasted it into the book.

Syntax coloring. For readability, we syntax color all the C++ code, similar to the way most C++ integrated-development environments and code editors syntax color code.

Our col oring conventions are as follows:

```
comments appear like this
keywords appear like this
constants and literal values appear like this
all other code appears in black
```

Code highlighting. We place light-blue shaded rectangles around key code segments.

Using fonts for emphasis. We color the defining occurrence of each key term in **bold blue** text for easy reference. We emphasize on-screen components in the bold Helvetica font (e.g., the File menu) and C++ program text in the `Lucida` font (for example, `intx= 5;`).

Objectives. The opening quotes are followed by a list of chapter objectives.

Programming tips. We include programming tips to help you focus on key aspects of pro gram development. These tips and practices represent the best we've gleaned from a com bined seven decades of teaching and industry experience.

Good Programming Practices

The Good Programming Practices *call attention to techniques that will help you pro duce programs that are clearer, more understandable and more maintainable.*

Common Programming Errors

Pointing out these Common Programming Errors *reduces the likelihood that you'll make them.*

Fig. 6 Chapter Dependency Chart

[*Note:* Arrows pointing into a chapter indicate that chapter's dependencies.]

Streams, Files and Strings

13 Stream Input/Output: A Deeper Look[1]

Introduction

1 Introduction to Computers and C++

Intro to Programming, Classes and Objects

2 Intro to C++ Programming, Input/Output and Operators

3 Intro to Classes, Objects and Strings

Control Statements, Methods and

Legacy C Topics

22 Bits, Characters, C-Strings and `struct`s

Object-Oriented Design with the UML

25 (Optional) Object-Oriented Design with the UML

26 (Optional) Implementing an Object-Oriented Design

Arrays

4 Control Statements: Part 1; Assignment, ++ and -- Operators

5 Control Statements: Part 2; Logical Operators

6 Functions and an Intro to Recursion

7 Class Templates `array` and `vector`; Catching Exceptions

8 Pointers

Object-Oriented Programming

9 Classes: A Deeper Look; Throwing Exceptions

10 Operator Overloading; Class `string`

11 OOP: Inheritance

12 OOP: Polymorphism

17 Exception Handling: A Deeper Look

Other Topics and C++11 Features

14 File Processing

21 Class `string` and String

Stream Processing: A Deeper Look

Data Structures

15 Standard Library

Additional

23 Other Topics Features

24 C++11:

Containers and Iterators

16 Standard Library

Algorithms

6.20–6.22 Recursion

18 Intro to Custom Templates

19 Custom Templatized Data Structures

20 Searching and Sorting

[1] Most of Chapter 13 is readable after Chapter 7. A small portion requires Chapters 11 and 18.

Error-Prevention Tips

These tips contain suggestions for exposing and removing bugs from your programs; many describe aspects of C++ that prevent bugs from getting intoprograms in the first place.

Performance Tips

These tips highlight opportunities for making your programs run faster or minimizing the amount of memory that they occupy.

Portability Tips

The Portability Tips *help you write code that will run on a variety of platforms.*

Software Engineering Observations

The Software Engineering Observations *highlight architectural and design issues that affect the construction of software systems, especially large-scalesystems.*

Summary Bullets. We present a section-by-section, bullet-list summary of the chapter. We include the page number of each term's defining occurrence in the chapter for easy reference.

Index. We've included an extensive index, with defining occurrences of key terms highlighted with a **bold blue** page number.

Obtaining the Software Used in *C++ How to Program, 9/e*

We wrote the code examples in *C++ How to Program, 9/e* using the following C++ development tools:

- Microsoft's free Visual Studio Express 2012 for Windows Desktop, which includes Visual C++ and other Microsoft development tools. This runs on Windows 7 and 8 and is available for download at

  ```
  www.microsoft.com/visualstudio/eng/downloads#
      d-express-windows-desktop
  ```

- GNU's free GNU C++ (gcc.gnu.org/install/binaries.html), which is al ready installed on most Linux systems and can also be installed on Mac OS X and Windows systems.
- Apple's free Xcode, which OS X users can download from the Mac App Store.

Instructor Supplements

The following supplements are available to *qualified instructors only* through Pearson Education's Instructor Resource Center (www.pearsonhighered.com/irc):

- ***Solutions Manual*** contains solutions to *most* of the end-of-chapter exercises. We've added many *Making a Difference*exercises, most with solutions. **Please do not write to us requesting access to the Pearson Instructor's Resource Center. Access is re stricted to college instructors teaching from the book.** Instructors may obtain ac cess only through their Pearson representatives. If you're not a registered faculty member, contact your Pearson representative or visit www.pearsonhighered.com/ educator/replocator/. Exercise Solutions are *not* provided for "project" exercis

  es. Check out our Programming Projects Resource Center for lots of additional ex ercise and project possibilities

  ```
  www.deitel.com/ProgrammingProjects
  ```

- ***Test Item File*** of multiple-choice questions (approximately two per book section)
- ***Customizable PowerPoint® slides*** containing all the code and figures in the text, plus bulleted items that summarize the key points in the text

Online Practice and Assessment with

MyProgrammingLab™ MyProgrammingLab™ helps students fully grasp the logic, semantics, and syntax of pro gramming. Through practice exercises and immediate, personalized feedback, MyPro grammingLab improves the programming competence of beginning students who often struggle with the basic concepts and paradigms of popular high-level programming lan guages.

A self-study and homework tool, a MyProgrammingLab course consists of hundreds of small practice problems organized around the structure of this textbook. For students, the system automatically detects errors in the logic and syntax of their code submissions and offers targeted hints that enable students to figure out what went wrong—and why. For instructors, a comprehensive gradebook tracks correct and incorrect answers and stores the code inputted by students for review.

For a full demonstration, to see feedback from instructors and students or to get started using MyProgrammingLab in your course, visit www.myprogramminglab.com.

Acknowledgments

We'd like to thank Abbey Deitel and Barbara Deitel of Deitel & Associates, Inc. for long hours devoted to this project. Abbey co-authored Chapter 1 and she and Barbara painstakingly researched the new capabilities of C++11.

We're fortunate to have worked with the dedicated team of publishing professionals at Pearson Higher Education. We appreciate the guidance, wisdom and energy of Tracy Johnson, Executive Editor, Computer Science. Carole Snyder did an extraordinary job recruiting the book's reviewers and managing the review process. Bob Engelhardt did a wonderful job bringing the book to publication.

Reviewers

We wish to acknowledge the efforts of our reviewers. The book was scrutinized by current and former members of the C++ standards committee that developed C++11, academics teaching C++ courses and industry experts. They provided countless suggestions for im proving the presentation. Any remaining flaws in the book are our own.

Ninth Edition reviewers: Dean Michael Berris (Google, Member ISO C++ Committee), Danny Kalev (C++ expert, certified system analyst and former member of the C++ Standards Committee), Linda M. Krause (Elmhurst College), James P. McNellis (Micro soft Corporation), Robert C. Seacord (Secure Coding Manager at SEI/CERT, author of *Secure Coding in C and C++*) and José Antonio González Seco (Parliament of Andalusia).

Other recent edition reviewers: Virginia Bailey (Jackson State University), Thomas J. Borrelli (Rochester Institute of Technology), Ed Brey (Kohler Co.), Chris Cox (Adobe

Systems), Gregory Dai (eBay), Peter J. DePasquale (The College of New Jersey), John Dibling (SpryWare), Susan Gauch (University of Arkansas), Doug Gregor (Apple, Inc.), Jack Hagemeister (Washington State University), Williams M. Higdon (University of Indiana), Anne B. Horton (Lockheed Martin), Terrell Hull (Logicalis Integration Solutions), Ed James-Beckham (Borland), Wing-Ning Li (University of Arkansas), Dean Mathias (Utah State University), Robert A. McLain (Tidewater Community College), Robert Myers (Florida State University), Gavin Osborne (Saskatchewan Inst. of App. Sci. and Tech.), Amar Raheja (California State Polytechnic University, Pomona), April Reagan (Microsoft), Raymond Stephenson (Microsoft), Dave Topham (Ohlone College), Anthony Williams (author and C++ Standards Committee member) and Chad Willwerth (University Washington, Tacoma).

As you read the book, we'd sincerely appreciate your comments, criticisms and suggestions for improving the text. Please address all correspondence to:

deitel@deitel.com

We'll respond promptly. We enjoyed writing *C++ How to Program, Ninth Edition*. We hope you enjoy reading it!

Paul Deitel
Harvey Deitel

About the Authors

Paul Deitel, CEO and Chief Technical Officer of Deitel & Associates, Inc., is a graduate of MIT, where he studied Information Technology. Through Deitel & Associates, Inc., he has delivered hundreds of programming courses to industry clients, including Cisco, IBM, Siemens, Sun Microsystems, Dell, Fidelity, NASA at the Kennedy Space Center, the National Severe Storm Laboratory, White Sands Missile Range, Rogue Wave Software, Boeing, SunGard Higher Education, Nortel Networks, Puma, iRobot, Invensys and many more. He and his co-author, Dr. Harvey M. Deitel, are the world's best-selling program ming-language textbook/professional book/video authors.

Dr. Harvey Deitel, Chairman and Chief Strategy Officer of Deitel & Associates, Inc., has 50 years of experience in the computer field. Dr. Deitel earned B.S. and M.S. degrees in Electrical Engineering from MIT and a Ph.D. in Mathematics from Boston University. He has extensive college teaching experience, including earning tenure and serving as the Chairman of the Computer Science Department at Boston College before founding Deitel & Associates, Inc., in 1991 with his son, Paul Deitel. The Deitels' publications have earned international recognition, with translations published in Chinese, Korean, Japa nese, German, Russian, Spanish, French, Polish, Italian, Portuguese, Greek, Urdu and Turkish. Dr. Deitel has delivered hundreds of programming courses to corporate, aca demic, government and military clients.

Corporate Training from Deitel & Associates, Inc.

Deitel & Associates, Inc., founded by Paul Deitel and Harvey Deitel, is an internationally recognized authoring and corporate training organization, specializing in computer programming languages, object technology, mobile app development and Internet and web

software technology. The company's clients include many of the world's largest companies, government agencies, branches of the military, and academic institutions. The company offers instructor-led training courses delivered at client sites worldwide on major programming languages and platforms, including C++, Visual C++®, C, Java™, Visual C#®, Visual Basic®, XML®, Python®, object technology, Internet and web programming, Android app development, Objective-C and iPhone app development and a growing list of additional programming and software development courses.

Through its 36-year publishing partnership with Prentice Hall/Pearson, Deitel & Associates, Inc., publishes leading-edge programming college textbooks, professional books and *LiveLessons* video courses. Deitel & Associates, Inc. and the authors can be reached at:

deitel@deitel.com

To learn more about Deitel's *Dive-Into*® *Series* Corporate Training curriculum, visit:

www.deitel.com/training

To request a proposal for worldwide on-site, instructor-led training at your organization, e-mail deitel@deitel.com.

Individuals wishing to purchase Deitel books and *LiveLessons* video training can do so through www.deitel.com. Bulk orders by corporations, the government, the military and academic institutions should be placed directly with Pearson. For more information, visit

www.pearsonhighered.com/information/index.page

Introduction to Computers and C++

In this chapter you'll learn:

- Exciting recent developments in the computer field.
- Computer hardware, soft ware and networking basics.
- The data hierarchy.
- The different types of programming languages.
- Basic object-technology concepts.
- Some basics of the Internet and the World Wide Web.
- A typical C++ program development environment.
- To test-drive a C++ application.
- Some key recent software technologies.
- Howcomputers can help you make a difference.

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

Good design is good business. —Thomas J. Watson, Founder of IBM

How wonderful it is that nobody need wait a single moment before starting to improve the world.
—Anne Frank

Objectives

*2 -) &2-(5 2!(& 2((%*52 -. & ,,

+-+ *73+ : 7%+*

+-9 +)-:7 34 * 7# *7 3* 7 %* * :473> * 4 3 #

+-8 3 < 3 * + 7< 3

,080, ++3 24 <

,0809 +)-:7 3 3" *%? 7%+*

+- 7 % 3 3 #>

+- #%* *": " 4 44) (> *": " 4 * %"#$; (*": " 4 +-5 11

+-4 3+"3))%*" *": " 4
+- *73+ : 7%+* 7+ & 7 #*+(+">
+-) >-% (11 ; (+-) *7 *;%3+*) *7
+-+@ 47$ 3%;%*" 11 --(%
7%+* +-++ - 3 7%*" >47)4
,0,,0, %* +<4
,0,,09 %*:=
,0,,08 --(24
,0,,0! ++"(24 * 3+%
+-+9 # *7 3* 7 * +3(%

+-+8 +)
> + 7< 3 ; (+-) *7 3)%*+(+">
+-+ 11,, * 7# - * +:3 ++47 %
3 3% 4
+-+
-%*" - 7+ 7 <%7#
* +3) 7%+*
#*+(+"% 4
+-+5 4+:3 4

+-+ (70* : 7#*(

)8$=9) T= DD, ?=Z)L/X8 $=9?XT)L ?L=2L 995:2 8 :2X 2) T3 TJN ??L=?L5 T) /=L T)$3:5$ 88\ =L5):T)' ?)=?8) Z5T3 85TT8) =L := ?L=2L 995:2)[?)L5):$)& :' /=L)[?)L5):$)' ?L=2L 99)LN T= XN) 5: X58'5:2 NX NT :T5 8 5:/=L9 T5=: N\NT)9NC =XJL) 8L) '\ / 9585 L Z5T3 T3) ?=Z)L/X8 T N7N $=9?XT)LN ?)L/=L9C N5:2 T35N T)[T ==7& \=XJ88 ZL5T) 5:NTLX$T5=:N $=99 :'5:2 $=9?XT)LN T= ?)L/=L9 T3=N) 75:'N =/ T N7NC * ;? 5 @5C)C& T3) 5:NTLX$T5=:N \=X ZL5T)A $=:TL=8N ! 5 ? 5 @5C)C& $=9?XT)LNAC

=XJ88 8) L: * $;"*5# (; ,5* 5 "#(,T=' \JN 7)\ ?L=2L 995:2 9)T3='=8=2\C =XJ88 $L) T) 9 :\ 7* ;? 5 * $;7 T3 T 9=')8 ;!#(7 5: T3) L) 84Z=L8'C DD 5N =:) =/ T=' \JN 9=NT ?=?X8 L N=/TZ L) ')Y)8=?9):T 8 :2X 2)NC 35N T)[T ?L=4 Y5')N : 5:TL='X$T5=: T= ?L=2L 995:2 5: DD>>,T3) 8 T)NT Y)LN5=: NT :' L'5])' T3L=X23 T3) %3 /% 3 &% # / % : 3 &% &/ 3 % / : 3 &% @ A :' T3) %3 /% 3 &% # # 3/& 3 % # &$$ 00 &% @ AC

: XN) T=' \ L) 9=L) T3 : 5885=: 2):)L 84?XL?=N) $=9?XT)LN :' 5885=:N 9=L) $)88 ?3=:)N& N9 LT?3=:)N :' 3 :'3)8' ')Y5$)N @NX$3 N T 8)T $=9?XT)LNAC $$=L'5:2 T= NTX'\ \ ) L7)T)L& T3) :X9)L =/ 9= 58) :T)L:)T XN)LN Z588 L) $3 ??L=[59 T)8\ >U1 95885=: \ W^>UC[>] 9 LT?3=:) N 8)N NXL? NN)' ?)LN=: 8 $=9?XT)L N 8)N 5: W^>>C[W] 8)T N 8)N L))[?)$T)' T= $$=X:T /=L =Y)L W^B =/ 88 ?)LN=: 8 $=9?XT)L N 8)N \ W^>0C[U] \ W^>1& T3) N9 LT?3=:) ??85$ T5=:N 9 L7)T 5N)[?)$T)' T=)[$))' (1^ 5885=:C[1] 35N)[?8=4 N5Y) 2L=ZT3 5N $L) T5:2 N52:5/5$:T =??=LTX:5T5)N /=L ?L=2L 995:2 9= 58) ??85$ T5=:NC

+-9 *',:7 02 (7! (7 0(7 #((:270> (2 0 !

3)N) L))[$5T5:2 T59)N 5: T3) $=9?XT)L /5)8'C :\ =/ T3) 9=NT 5:/8X):T5 8 :' NX$$)NN/X8 XN5:)NN)N =/ T3) 8 NT TZ= ')$ ')N L) T)$3:=8=2\ $=9? :5)N& 5:$8X'5:2 ??8)& &)Z4

>C !!!"#$%#&'$("#)*+,)-.-+*)/$&'0$1.'%1'.02-'%-0.)0%'3#405670*$&&$)10/809:56+C WC

lett Packard, Dell, Intel, Motorola, Cisco, Microsoft, Google, Amazon, Facebook, Twitter, Groupon, Foursquare, Yahoo!, eBay and many more. These companies are major employers of people who study computer science, computer engineering, information systems or related disciplines. At the time of this writing, Apple was the most valuable company in the world. Figure 1.1 provides a few examples of the ways in which computers are improving people's lives in research, industry and society.

Electronic health records

Human Genome Project

These might include a patient's medical history, prescriptions, immunizations, lab results, allergies, insurance information and more. Making this information available to health care providers across a secure network improves patient care, reduces the probability of error and increases overall efficiency of the health care system. The Human Genome Project was founded to identify and analyze the 20,000+ genes in human DNA. The project used computer programs to analyze complex genetic data, determine the sequences of the billions of chemical base pairs that make up human DNA and store the information in databases which have been made available over the Internet to researchers in many fields.

AMBER™ Alert The AMBER (America's Missing: Broadcast Emergency Response) Alert System is used to find abducted children. Law enforcement notifies TV and radio broadcasters and state transportation officials, who then broadcast alerts on TV, radio, computerized highway
signs, the Internet and wireless devices. AMBER Alert recently partnered with Facebook, whose users can "Like" AMBER Alert pages by location to receive alerts in their news feeds.

World Community Grid

Cloud
computing

People worldwide can donate their unused computer processing power by installing a free secure software program that allows the World Community Grid (www.worldcommunitygrid.org) to harness unused capacity. This computing power, accessed over the Internet, is used in place of expensive supercomputers to conduct scientific research projects that are making a difference—providing clean water to third-world countries, fighting cancer, growing more nutritious rice for regions fighting hunger and more.

Cloud computing allows you to use software, hardware and information stored in the "cloud"—i.e., accessed on remote computers via the Internet and available on demand—rather than having it stored on your personal computer. These services allow you to increase or decrease resources to meet your needs at any given time, so they can be more cost effective than purchasing expensive hardware to ensure that you have enough storage and processing power to meet your needs at their peak levels. Using cloud computing services shifts the burden of managing these applications from the business to the service provider, saving businesses money.

Fig. 1.1 | A few uses for computers. (Part 1 of 3.)

Medical imaging X-ray computed tomography (CT) scans, also called CAT (computerized axial tomography) scans, take X-rays of the body from hundreds of different angles. Computers are used to adjust the intensity of the X-rays, optimizing the scan for each type of tissue, then to combine all of the information to create a 3D image. MRI scanners use a technique called magnetic resonance imaging, also to produce internal images non-invasively.

GPS Global Positioning System (GPS) devices use a network of satellites to retrieve location-based information. Multiple satellites send time stamped signals to the GPS device, which calculates the distance to each satellite based on the time the signal left the satellite and the time the signal arrived. This information is used to determine the exact location of the device. GPS devices can provide step-by-step directions and help you locate nearby businesses (restaurants, gas stations, etc.) and points of interest. GPS is used in numerous location-based Internet services such as check-in apps to help you find your friends (e.g., Foursquare and Facebook), exercise apps such as RunKeeper that track the time, distance and average speed of your outdoor jog, dating apps that help you find a match nearby and apps that dynamically update changing traffic conditions.

Robots Robots can be used for day-to-day tasks (e.g., iRobot's Roomba vacuuming robot), entertainment (e.g., robotic pets), military combat, deep sea and space exploration (e.g., NASA's Mars rover Curiosity) and more. RoboEarth (www.roboearth.org) is "a World Wide Web for robots." It allows robots to learn from each other by sharing information and thus improving their abilities to perform tasks, navigate, recognize objects and more.

E-mail, Instant Messaging, Video Chat and FTP

Internet-based servers support all of your online messaging. E-mail messages go through a mail server that also stores the messages. Instant Messaging (IM) and Video Chat apps, such as AIM, Skype, Yahoo! Messenger, Google Talk, Trillian, Microsoft's Messenger and others allow you to communicate with others in real time by sending your messages and live video through servers. FTP (file transfer protocol) allows you to exchange files between multiple computers (e.g., a client computer such as your desktop and a file server) over the Internet.

Internet TV Internet TV set-top boxes (such as Apple TV, Google TV and TiVo) allow you to access an enormous amount of content on demand, such as games, news, movies, television shows and more, and they help ensure that the content is streamed to your TV smoothly.

Streaming music services

Streaming music services (such as Pandora, Spotify, Last.fm and more) allow you listen to large catalogues of music over the web, create customized "radio stations" and discover new music based on your feedback.

Fig. 1.1 | A few uses for computers. (Part 2 of 3.)

Game programming	Analysts expect global video game revenues to reach $91 billion by 2015 (www.vg247.com/2009/06/23/global-industry-analysts predicts-gaming-market-to-reach-91-billion-by-2015/). The most sophisticated games can cost as much as $100 million to develop. Activision's *Call of Duty: Black Ops*—one of the best-selling games of all time—earned $360 million in just one day (www.forbes.com/sites/insertcoin/2011/03/11/call-of-duty black-ops-now-the-best-selling-video-game-of-all-time/)! Online *social gaming*, which enables users worldwide to compete with one another over the Internet, is growing rapidly. Zynga—creator of popular online games such as *Words with Friends*, *CityVille* and others—was founded in 2007 and already has over 300 million monthly users. To accommodate the growth in traffic, Zynga is adding nearly 1,000 servers each week (techcrunch.com/2010/09/22/zynga-moves-1-petabyte-of-data-daily-adds-1000-servers-a week/)!

Fig. 1.1 | A few uses for computers. (Part 3 of 3.)

1.3 Hardware and Software

Computers can perform calculations and make logical decisions phenomenally faster than human beings can. Many of 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! IBM's Sequoia supercomputer can perform over 16 quadrillion calculations per second (16.32 *petaflops*)![5] To put that in perspective, *the IBM Sequoia supercomputer can perform in one second about 1.5 million calculations for every person on the planet!* And—these "upper limits" are growing quickly!

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**. The programs that run on a computer are referred to as **software**. In this book, you'll learn a key programming methodology that's enhancing programmer productivity, thereby reducing software development costs—*object-oriented programming.*

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. 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. Ironically, silicon is one of the most abundant materials on Earth—it's an ingredient in common sand. Silicon-chip technology has made computing so economical that computers have become a commodity.

[5] www.top500.org/.

1.3.1 Moore's Law

Every year, you probably expect to pay at least a little more for most products and services. The opposite has been the case in the computer and communications fields, especially with regard to the costs of hardware supporting these technologies. For many decades, hardware costs have fallen rapidly. Every year or two, the capacities of computers have approximately *doubled* inexpensively. This remarkable trend often is called **Moore's Law**, named for the person who identified it in the 1960s, Gordon Moore, co-founder of In tel—the leading manufacturer of the processors in today's computers and embedded sys tems. 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). Sim ilar growth has occurred in the communications field, in which costs have plummeted as enormous demand for communications bandwidth (i.e., information-carrying capacity) has attracted intense competition. We know of no other fields in which technology im proves so quickly and costs fall so rapidly. Such phenomenal improvement is truly foster ing the *Information Revolution*.

1.3.2 Computer Organization

Regardless of differences in *physical* appearance, computers can be envisioned as divided into various **logical units** or sections (Fig. 1.2).

Input unit 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 information 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 accelerome ter in a smartphone or game controller (such as Microsoft® Kinect™, Wii™ Remote and Sony's PlayStation® Move).

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's output from computers today is dis played on screens, printed on paper ("going green" discourages this), played as audio or video on PCs and media players (such as Apple's popular iPods) and giant screens in sports stadiums, transmitted over the Internet or used to control other devices, such as robots and "intelligent" appliances.

Fig. 1.2 | Logical units of a computer. (Part 1 of 2.)

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**. Main memories on desktop and notebook computers commonly contain as much as 16 GB (GB stands for gigabytes; a gigabyte is approximately one billion bytes).

Arithmetic and logic unit (ALU)

Central processing unit (CPU)

Secondary storage unit

This "manufacturing" section performs *calculations*, such as addition, sub traction, multiplication and division. It also contains the *decision* mecha nisms 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.

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 *quad-core processor* has four CPUs. Today's desktop computers have processors that can execute billions of instructions per second.

This is the long-term, high-capacity "warehousing" section. Programs or data not actively being used by the other units normally are placed on sec ondary 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 the cost per unit of secondary storage is much less than that of primary memory. Examples of secondary storage devices include CD drives, DVD drives and flash drives, some of which can hold up to 768 GB. Typical hard drives on desktop and note book computers can hold up to 2 TB (TB stands for terabytes; a terabyte is approximately one trillion bytes).

Fig. 1.2 | Logical units of a computer. (Part 2 of 2.)

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. Figure 1.3 illustrates a portion of the data hierarchy. Figure 1.4 summarizes the data hierarchy's levels.

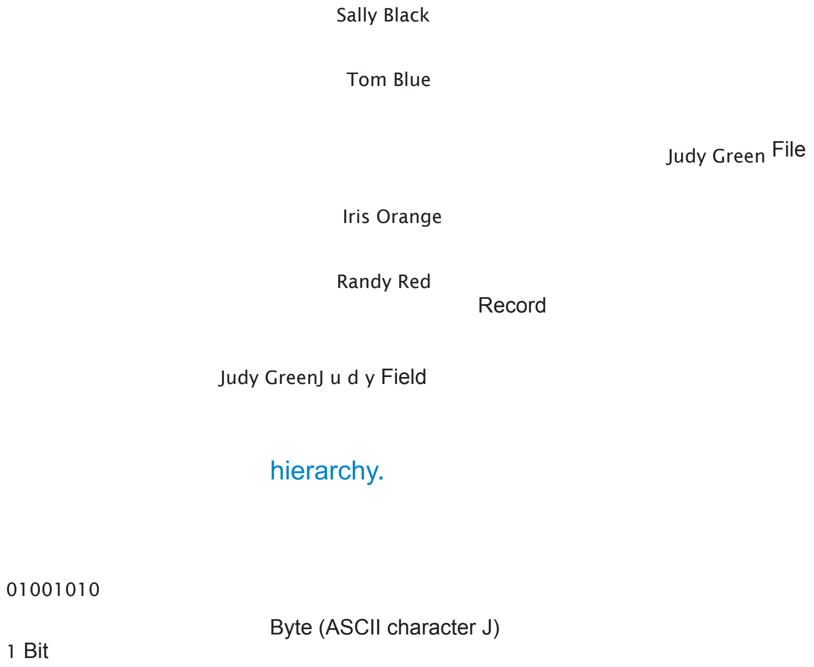


Fig. 1.3 | Data hierarchy.

Bits The smallest data item in a computer can assume the value `0` or the value `1`. Such a data item is called a **bit** (short for "binary digit"—a digit that can assume one of two values). It's remarkable that the impressive functions performed by computers involve only the simplest manipulations of `0`s and `1`s—*examining a bit's value*, *setting a bit's value* and *reversing a bit's value* (from `1` to `0` or from `0` to `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**. The computer's **character set** is the set of all the characters used to write programs and represent data items. Computers process only `1`s and `0`s, so every character is represented as a pattern of `1`s and `0`s. The **Unicode®** character set contains characters for many of the world's languages. C++ supports several character sets, including 16-bit Unicode® characters that are composed of two **bytes**, each composed of eight bits. See Appendix B for more information on the **ASCII (American Standard Code for Information Interchange)** character set—the popular subset of Unicode that represents uppercase and lowercase letters, digits and some common special characters.

Fields Just as characters are composed of bits, **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, and a field consisting of decimal digits could represent a person's age.

Fig. 1.4 | Levels of the data hierarchy. (Part 1 of 2.)

Records Several related fields can be used to compose a **record**. In a payroll system, for example, the record for an employee might consist of the following fields (possible types for these fields are shown in parentheses):

- Employee identification number (a whole number)
- Name (a string of characters)
- Address (a string of characters)
- Hourly pay rate (a number with a decimal point)
- Year-to-date earnings (a number with a decimal point)
- Amount of taxes withheld (a number with a decimal point)

Thus, a record is a group of related fields. In the preceding example, all the fields belong to the same employee. A company might have many employees and a payroll record for each one.

Files A **file** is a group of related records. [*Note:* 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.] It's not unusual for an organization to have many files, some containing billions, or even trillions, of characters of information.

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*. A table includes *records* and *fields*. For example, a table of students might include first name, last name, major, year, student ID number and grade point average. The data for each student is a record, and the individual pieces of information in each record are the fields. You can search, sort and manipulate the data based on its relationship to multiple tables or databases. For example, a university might use data from the student database in combination with databases of courses, on-campus housing, meal plans, etc.

Fig. 1.4 | Levels of the data hierarchy. (Part 2 of 2.)

1.5 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.

Machine Languages

Any computer can directly understand only its own **machine language** (also called *machine code*), defined by its hardware architecture. Machine languages generally consist of numbers (ultimately reduced to 1s and 0s). Such languages are cumbersome for humans.

Assembly Languages

Programming in machine language was simply too slow and tedious for most programmers. Instead, they began using English-like *abbreviations* to represent elementary operations. These abbreviations formed the basis of **assembly languages**. *Translator programs* called **assemblers** were developed to convert assembly-language programs to machine language. Although assembly-language code is clearer to humans, it's incomprehensible to computers until translated to machine language.

High-Level Languages

To speed up the programming process further, **high-level languages** were developed in which single statements could be written to accomplish substantial tasks. High-level languages, such as C++, Java, C# and Visual Basic, allow you to write instructions that look more like everyday English and contain commonly used mathematical expressions. Translator programs called **compilers** convert high-level language programs into machine language.

The process of compiling a large high-level language program into machine language can take a considerable amount of computer time. **Interpreter** programs were developed to execute high-level language programs directly (without the need for compilation), although more slowly than compiled programs. **Scripting languages** such as the popular web languages JavaScript and PHP are processed by interpreters.

Performance Tip 1.1

Interpreters have an advantage over compilers in Internet scripting. An interpreted program can begin executing as soon as it's downloaded to the client's machine, without needing to be compiled before it can execute. On the downside, interpreted scripts generally run slower than compiled code.

1.6 C++

C++ evolved from C, which was developed by Dennis Ritchie at Bell Laboratories. C is available for most computers and is hardware independent. With careful design, it's possible to write C programs that are **portable** to most computers.

The widespread use of C with various kinds of computers (sometimes called **hardware platforms**) unfortunately led to many variations. A standard version of C was needed. The American National Standards Institute (ANSI) cooperated with the International Organization for Standardization (ISO) to standardize C worldwide; the joint standard document was published in 1990 and is referred to as *ANSI/ISO 9899: 1990*.

C11 is the latest ANSI standard for the C programming language. It was developed to evolve the C language to keep pace with increasingly powerful hardware and ever more demanding user requirements. C11 also makes C more consistent with C++. For more information on C and C11, see our book *C How to Program, 7/e* and our C Resource Center (located at www.deitel.com/C).

C++, an extension of C, was developed by Bjarne Stroustrup in 1979 at Bell Laboratories. Originally called "C with Classes", it was renamed to C++ in the early 1980s. C++ provides a number of features that "spruce up" the C language, but more importantly, it provides capabilities for object-oriented programming.

You'll begin developing customized, reusable classes and objects in Chapter 3, Intro duction to Classes, Objects and Strings. The book is object oriented, where appropriate, from the start and throughout the text.

We also provide an *optional* automated teller machine (ATM) case study in Chapters 25–26, which contains a complete C++ implementation. The case study presents a carefully paced introduction to object-oriented design using the UML—an industry standard graphical modeling language for developing object-oriented systems. We guide you through a friendly design experience intended for the novice.

C++ Standard Library

C++ programs consist of pieces called **classes** and **functions.** You can program each piece yourself, but most C++ programmers take advantage of the rich collections of classes and functions in the **C++ Standard Library**. Thus, there are really two parts to learning the C++ "world." The first is learning the C++ language itself; the second is learning how to use the classes and functions in the C++ Standard Library. We discuss many of these classes and functions. P. J. Plauger's book, *The Standard C Library* (Upper Saddle River, NJ: Prentice Hall PTR, 1992), is a must read for programmers who need a deep understanding of the ANSI C library functions included in C++. 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 typically will 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.

The advantage of creating your own functions and classes is that you'll know exactly how they work. You'll be able to examine the C++ code. The disadvantage is the time-con suming and complex effort that goes into designing, developing and maintaining new functions and classes that are correct and that operate efficiently.

Performance Tip 1.2

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 1.1

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

1.7 Programming Languages

In this section, we provide brief comments on several popular programming languages (Fig. 1.5).

Fortran Fortran (FORmula TRANslator) was developed by IBM Corpo ration in the mid-1950s to be used for scientific and engineering applications that require complex mathematical computations. It’s still widely used and its latest versions support object-oriented programming.

COBOL COBOL (COmmon Business Oriented Language) was devel oped in the late 1950s by computer manufacturers, the U.S. government and industrial computer users based on a language developed by Grace Hopper, a career U.S. Navy officer and computer scientist. COBOL is still widely used for commercial applications that require precise and efficient manipulation of large amounts of data. Its latest version supports object-oriented programming.

Pascal Research in the 1960s resulted in *structured programming*—a dis ciplined approach to writing programs that are clearer, easier to test and debug and easier to modify than large programs produced with previous techniques. One of the more tangible results of this research was the development of Pascal byProfessor Niklaus Wirth in 1971. It was designed for teaching structured programming and was popular in college courses for several decades.

Ada Ada, based on Pascal, was developed under the sponsorship of the U.S. Department of Defense (DOD) during the 1970s and early 1980s. The DOD wanted a single language that would fill most of its needs. The Pascal-based language was named after Lady Ada Lovelace, daughter of the poet Lord Byron. She’s credited with writing the world’s first computer program in the early 1800s (for the Analytical Engine mechanical computing device designed by Charles Babbage). Ada also supports object-oriented programming.

Basic Basic was developed in the 1960s at Dartmouth College to famil iarize novices with programming techniques. Many of its latest versions are object oriented.

C C was implemented in 1972 by Dennis Ritchie at Bell Laborato ries. It initially became widely known as the UNIX operating system’s development language. Today, most of the code for general purpose operating systems is written in C or C++.

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).

Fig. 1.5 | Some other programming languages. (Part 1 of 3.)

Java Sun Microsystems in 1991 funded an internal corporate research project led by James Gosling, which resulted in the C++-based object-oriented programming language called Java. A key goal of Java is to be able to write programs that will run on a great vari ety of computer systems and computer-control 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 (e.g., smartphones, tablets, television set-top boxes, appliances, automobiles and more) and for many other purposes. Java is also the key language for developing Android smartphone and tablet apps.

Visual Basic Microsoft's Visual Basic language was introduced in the early 1990s to simplify the development of Microsoft Windows appli cations. Its latest versions support object-oriented programming.

C# Microsoft's three object-oriented primary programming lan guages are Visual Basic (based on the original 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).

PHP PHP is an object-oriented, "open-source" (see Section 1.11.2) "scripting" language supported by a community of users and developers and is used by numerous websites including Wikipedia and Facebook. PHP is platform independent—implementations exist for all major UNIX, Linux, Mac and Windows operating systems. PHP also supports many databases, including MySQL.

Perl Perl (Practical Extraction and Report Language), one of the most widely used object-oriented scripting languages for web programming, was developed in 1987 by Larry Wall. It features rich text-processing capabilities and flexibility.

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.

JavaScript JavaScript is the most widely used scripting language. It's primarily used to add programmability to web pages—for example, ani

	mations and interactivity with the user. It's provided with all major web browsers.

Fig. 1.5 | Some other programming languages. (Part 2 of 3.)

Ruby on Rails	Ruby—created in the mid-1990s by Yukihiro Matsumoto—is an open-source, object-oriented programming language with a simple syntax that's similar to Perl and Python. Ruby on Rails combines the scripting language Ruby with the Rails web application framework developed by 37Signals. Their book, *Getting Real* (available free at gettingreal.37signals.com/toc.php), is a must read for web developers. Many Ruby on Rails developers have reported productivity gains over other languages when developing database-intensive web applications. Ruby on Rails was used to build Twitter's user interface.
Scala	Scala (www.scala-lang.org/node/273)—short for "scalable language"—was designed by Martin Odersky, a professor at École Polytechnique Fédérale de Lausanne (EPFL) in Switzerland. Released in 2003, Scala uses both the object-oriented programming and functional programming paradigms and is designed to integrate with Java. Programming in Scala can reduce the amount of code in your applications significantly. Twitter and Foursquare use Scala.

Fig. 1.5 | Some other programming languages. (Part 3 of 3.)

1.8 Introduction to 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—as we'll see in Chapter 3—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). Software developers have discovered 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

Let's begin with a simple analogy. Suppose you want to *drive a car and make it go faster by pressing its accelerator pedal*. What must happen before you can do this? Well, 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. 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.

Member Functions and Classes

Let's use our car example to introduce some key object-oriented programming concepts. Performing a task in a program requires a **member function**. The member function houses the program statements that actually perform its task. It hides these statements from its user, just as the accelerator pedal of a car hides from the driver the mechanisms of making the car go faster. In C++, we create a program unit called a **class** to house the set of member functions that perform the class's tasks. For example, a class that represents a bank account might contain one member function 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.

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.

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*. Just as the notion of *interchangeable parts* was crucial to the Industrial Revolution, reusable classes are crucial to the software revolution that has been spurred by object technology.

Messages and Member Function 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 **member function call** that tells a member function of the object to perform its task. For example, a program might call a particular bank account object's *deposit* member function to increase the account's balance.

Attributes and Data Members

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. 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. 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 **data members**.

Encapsulation

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

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" cer tainly *is an* object of the more *general* class "automobile," but more*specifically*, the roof can be raised or lowered.

Object-Oriented Analysis and Design (OOAD)

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

To create the best solutions, you should follow a detailed **analysis** process for determining your project's **requirements** (i.e., defining *what* the system is supposed to do) and developing a **design** that satisfies them (i.e., deciding *how* the system should do it). Ideally, you'd go through this process and carefully review the design (and have your design reviewed by other software professionals) before writing any code. If this process involves analyzing and designing your system from an object-oriented point of view, it's called an **object-oriented analysis and design (OOAD) process**. Languages like C++ are object ori ented. Programming in such a language, called **object-oriented programming (OOP)**, allows you to implement an object-oriented design as a working system.

The UML (Unified Modeling Language)

Although many different OOAD processes exist, a single graphical language for communicating the results of *any* OOAD process has come into wide use. This language, known as the Unified Modeling Language (UML), is now the most widely used graphical scheme for modeling object-oriented systems. We present our first UML diagrams in Chapters 3 and 4, then use them in our deeper treatment of object-oriented programming through Chapter 12. In our *optional* ATM Software Engineering Case Study in Chapters 25–26 we present a simple subset of the UML's features as we guide you through an object-oriented design experience.

1.9 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. The following discussion explains a typical C++ program development environment.

Phase 1: Editing a Program

Phase 1 consists of editing a file with an *editor program*, normally known simply as an *editor* (Fig. 1.6). You type a C++ program (typically referred to as **source code**) using the editor, make any necessary corrections and save the program on a secondary storage device, such as your hard drive. C++ source code 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. See the documentation for your C++ compiler for more information on file-name extensions.

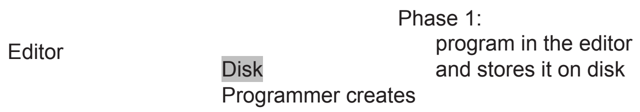


Fig. 1.6 | Typical C++ development environment—editing phase.

Two editors widely used on Linux systems are `vi` and `emacs`. C++ software packages for Microsoft Windows such as Microsoft Visual C++ (`microsoft.com/express`) have editors integrated into the programming environment. You can also use a simple text editor, such as Notepad in Windows, to write your C++ code.

For organizations that develop substantial information systems, **integrated development environments (IDEs)** are available from many major software suppliers. IDEs provide tools that support the software-development process, including editors for writing and editing programs and debuggers for locating **logic errors**—errors that cause programs to execute incorrectly. Popular IDEs include Microsoft® Visual Studio 2012 Express Edition, Dev C++, NetBeans, Eclipse, Apple's Xcode and CodeLite.

Phase 2: Preprocessing a C++ Program

In Phase 2, you give the command to **compile** the program (Fig. 1.7). In a C++ system, a **preprocessor** program executes automatically before the compiler's translation phase begins (so we call preprocessing Phase 2 and compiling Phase 3). The C++ preprocessor obeys commands called **preprocessing 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 preprocessing directives are discussed in the early chapters; a detailed discussion of preprocessor features appears in Appendix E, Preprocessor.

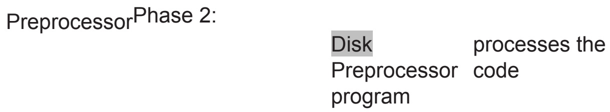


Fig. 1.7 | Typical C++ development environment—preprocessor phase.

Phase 3: Compiling a C++ Program

In Phase 3, the compiler translates the C++ program into machine-language code—also referred to as object code (Fig. 1.8).

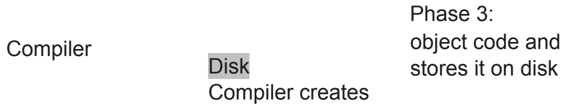


Fig. 1.8 | Typical C++ development environment—compilation phase.

Phase 4: Linking

Phase 4 is called **linking.** C++ programs typically contain references to functions and data defined elsewhere, such as in the standard libraries or in the private libraries of groups of pro grammers working on a particular project (Fig. 1.9). 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 program** (with no missing pieces). If the program compiles and links correctly, an executable image is produced.

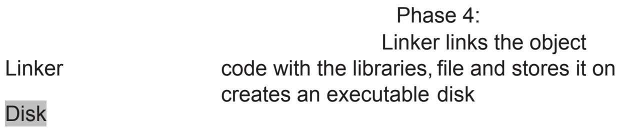

Phase 4:
Linker links the object
Linker
code with the libraries, file and stores it on
creates an executable disk
Disk

Fig. 1.9 | Typical C++ development environment—linking phase.

Phase 5: Loading

Phase 5 is called **loading.** Before a program can be executed, it must first be placed in memory (Fig. 1.10). 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.

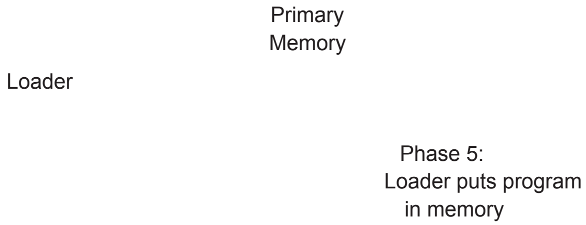


Fig. 1.10 | Typical C++ development environment—loading phase.

Phase 6: Execution

Finally, the computer, under the control of its CPU, **executes** the program one instruction at a time (Fig. 1.11). Some modern computer architectures can execute several instructions in parallel.

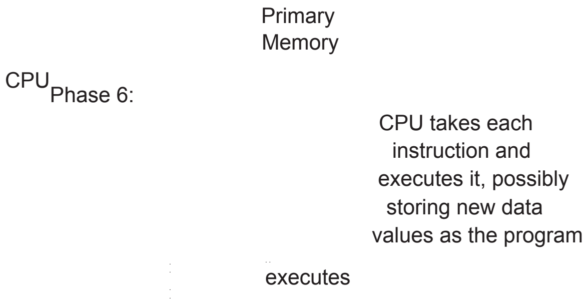

Primary
Memory
CPU
Phase 6:
CPU takes each instruction and executes it, possibly storing new data values as the program executes

Fig. 1.11 | Typical C++ development environment—execution phase.

Problems That May Occur at Execution Time

Programs might not work on the first try. Each of the preceding phases can fail because of various errors that we'll discuss throughout this book. For example, an executing program might try to divide by zero (an illegal operation for integer arithmetic in C++). This would cause the C++ program to display an error message. If this occurred, 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 fixed the problem(s). [*Note:* Most programs in C++ input 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 hard copy printers. There is also a **standard error stream** referred to as `cerr`. The `cerr` stream (normally connected to the screen) is used for displaying error messages.

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.*

1.10 Test-Driving a C++ Application

In this section, you'll run and interact with your first C++ application. You'll begin by run ning 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,

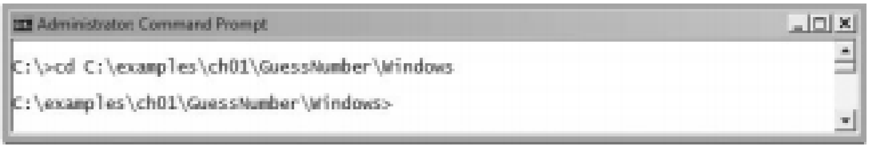
the application indicates whether your guess is higher or lower than the correct number. There is no limit on the number of guesses you can make. [*Note:* For this test drive only, we've modified this application from the exercise you'll be asked to create in Chapter 6, Functions and an Introduction to Recursion. Normally this application randomly selects the correct answer as you execute the program. The modified application uses the same correct answer every time the program executes (though this may vary by compiler), so you can use the *same* guesses we use in this section and see the *same* results as we walk you through interacting with your first C++ application.]

We'll demonstrate running a C++ application using the Windows Command Prompt and a shell on Linux. The application runs similarly on both platforms. Many develop ment environments are available in which you can compile, build and run C++ applica tions, such as GNU C++, Microsoft Visual C++, Apple Xcode, Dev C++, CodeLite, NetBeans, Eclipse etc. Consult your instructor for information on your specific develop ment environment.

In the following steps, you'll run the application and enter various numbers to guess the correct number. The elements and functionality that you see in this application are typical of those you'll learn to program in this book. We use fonts to distinguish between features you see on the screen (e.g., the Command Prompt) and elements that are not directly related to the screen. We emphasize screen features like titles and menus (e.g., the File menu) in a semibold sans-serif Helvetica font and emphasize filenames, text displayed by an application and values you should enter into an application (e.g., `GuessNumber` or `500`) in a `sans-serif Lucida` font. As you've noticed, the **defining occurrence** of each term is set in blue, bold type. For the figures in this section, we point out significant parts of the application. To make these features more visible, we've modified the background color of the Command Prompt window (for the Windows test drive only). To modify the Command Prompt colors on your system, open a Command Prompt by selecting Start > All Programs > Accessories > Command Prompt, then right click the title bar and select Prop erties. In the "Command Prompt" Properties dialog box that appears, click the Colors tab, and select your preferred text and background colors.

Running a C++ Application from the Windows Command Prompt 1. ***Checking your setup.*** It's important to read the Before You Begin section at `www.deitel.com/books/cpphtp9/` 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. 1.12). The command `cd` is used to change directories.



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

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

Fig. 1.12 | Opening a Command Prompt window and changing the directory.

3. ***Running the* GuessNumber *application.*** Now that you are in the directory that contains the `GuessNumber` application, type the command `GuessNumber` (Fig. 1.13) and press *Enter*. [*Note:* `GuessNumber.exe` is the actual name of the application; however, Windows assumes the `.exe` extension by default.]

Fig. 1.13 | 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. 1.13). At the prompt, enter `500` (Fig. 1.14).

Fig. 1.14 | 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. 1.15). The application again displays `"Too high. Try again."`, because the value you entered is still greater than the number that the application chose as the correct guess.

Fig. 1.15 | 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. 1.16).

Fig. 1.16 | Entering additional guesses and guessing the correct number.

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. 1.16). At the "Would you like to play again (y or n)?" prompt, entering the one character y causes the application to choose a new number and displays the message "Please type your first guess." followed by a question mark prompt (Fig. 1.17) so you can make your first guess in the new game. Entering the character n ends the application and returns you to the application's directory at the Command Prompt (Fig. 1.18). Each time you execute this application from the beginning (i.e., *Step 3*), it will choose the same numbers for you to guess.

8. ***Close the* Command Prompt *window.***

Fig. 1.17 | Playing the game again.

Fig. 1.18 | Exiting the game.

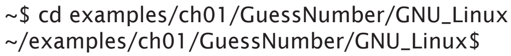
Running a C++ Application Using GNU C++ with Linux

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

1. ***Locating the completed application.*** From a Linux shell, change to the completed GuessNumber application directory (Fig. 1.19) by typing

   ```
   cd Examples/ch01/GuessNumber/GNU_Linux
   ```

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



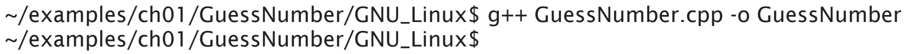
```
~$ cd examples/ch01/GuessNumber/GNU_Linux
~/examples/ch01/GuessNumber/GNU_Linux$
```

Fig. 1.19 | Changing to the GuessNumber application's directory.

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

   ```
   g++ GuessNumber.cpp -o GuessNumber
   ```

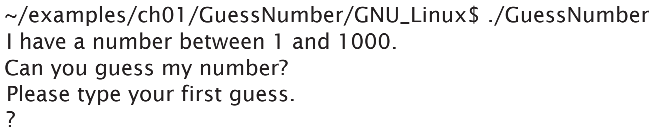
   as in Fig. 1.20. This command compiles the application and produces an execut able file called GuessNumber.



```
~/examples/ch01/GuessNumber/GNU_Linux$ g++ GuessNumber.cpp -o GuessNumber
~/examples/ch01/GuessNumber/GNU_Linux$
```

Fig. 1.20 | Compiling the GuessNumber application using the g++ command.

3. ***Running the GuessNumber application.*** To run the executable file GuessNumber, type ./GuessNumber at the next prompt, then press *Enter* (Fig. 1.21).



```
~/examples/ch01/GuessNumber/GNU_Linux$ ./GuessNumber
I have a number between 1 and 1000.
Can you guess my number?
Please type your first guess.
?
```

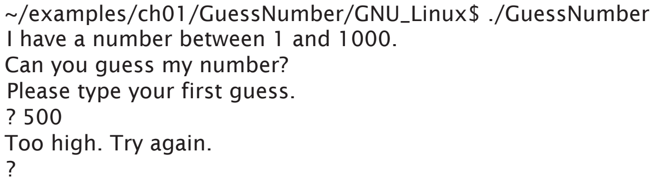
Fig. 1.21 | Running the GuessNumber application.

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. 1.21). At the prompt, enter 500 (Fig. 1.22). [*Note:* This is the same appli cation that we modified and test-drove for Windows, but the outputs could vary based on the compiler being used.]

5. ***Entering another guess.*** The application displays "Too high. Try again.", mean

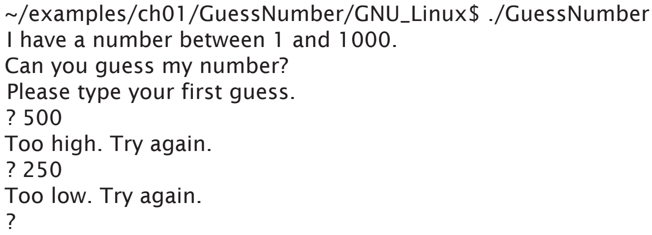
ing that the value you entered is greater than the number the application chose as the correct guess (Fig. 1.22). At the next prompt, enter 250 (Fig. 1.23). This time the application displays "Too low. Try again.", because the value you entered is less than the correct guess.

6. ***Entering additional guesses.*** Continue to play the game (Fig. 1.24) by entering values until you guess the correct number. When you guess correctly, the application displays "Excellent! You guessed the number."



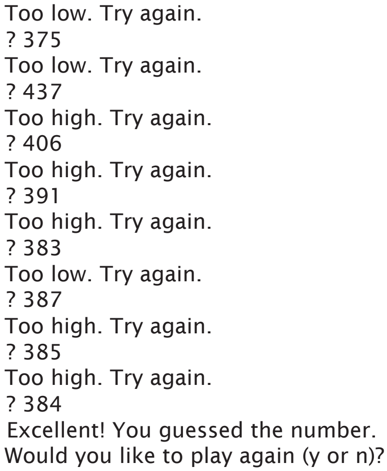
```
~/examples/ch01/GuessNumber/GNU_Linux$ ./GuessNumber
I have a number between 1 and 1000.
Can you guess my number?
Please type your first guess.
? 500
Too high. Try again.
?
```

Fig. 1.22 | Entering an initial guess.



```
~/examples/ch01/GuessNumber/GNU_Linux$ ./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.
?
```

Fig. 1.23 | Entering a second guess and receiving feedback.



```
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 (y or n)?
```

Fig. 1.24 | Entering additional guesses and guessing the correct number.

7. ***Playing the game again or exiting the application.*** After you guess the correct number, the application asks if you'd like to play another game. At the "Would you like to play again (y or n)?" prompt, entering the one character y causes the

application to choose a new number and displays the message "Please type your first guess." followed by a question mark prompt (Fig. 1.25) so you can make your first guess in the new game. Entering the character n ends the application and returns you to the application's directory in the shell (Fig. 1.26). Each time you execute this application from the beginning (i.e., *Step 3*), it will choose the same numbers for you to guess.

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

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

Fig. 1.25 | Playing the game again.

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

~/examples/ch01/GuessNumber/GNU_Linux$
```

Fig. 1.26 | Exiting the game.

1.11 Operating Systems

Operating systems are software systems that make using computers more convenient for users, application developers and system administrators. They 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 and OS X (formerly called Mac OS X)—we used all three in developing this book. 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. You can develop applications in C++ for all of the following key operating systems, including several of the latest mobile operating systems.

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 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 8 is Microsoft's latest operating system—its features include enhancements to the user interface, faster startup times, further refinement of security features, touch-screen and multitouch support, and more. Windows is a *proprietary* operating system—it's controlled by Microsoft exclusively. Windows is by far the world's most widely used desktop operating system.

1.11.2 Linux—An Open-Source Operating System

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 dom inated 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*. Open-source code is often scrutinized by a much larger audience than proprietary software, so errors of ten get removed faster. Open source also encourages innovation. Enterprise systems com panies, such as IBM, Oracle and many others, have made significant investments in Linux open-source development.

Some key organizations in the open-source community are the Eclipse Foundation (the Eclipse Integrated Development Environment helps programmers conveniently develop software), the Mozilla Foundation (creators of the Firefox web browser), the Apache Software Foundation (creators of the Apache web server used to develop web based applications) and SourceForge (which provides tools for managing open-source projects—it has hundreds of thousands of them under development). 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.

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 OS X, Linux source code (the program code) is available to the public for examination and modification and is free to download and install. As a result, Linux users benefit from a community of developers actively debugging and improving the kernel, and the ability to customize the operating system to meet specific needs.

A variety of issues—such as Microsoft's market power, the small number of user friendly Linux applications and the diversity of Linux distributions, such as Red Hat Linux, Ubuntu Linux and many others—have prevented widespread Linux use on desktop computers. Linux has become extremely popular on servers and in embedded sys tems, such as Google's Android-based smartphones.

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

Apple, founded in 1976 by Steve Jobs and Steve Wozniak, quickly became a leader in personal computing. In 1979, 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. At the time of this writing, Objective-C was comparable in popularity to C++.[6] 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 inter face and Interface Builder—used to construct graphical user interfaces.

Jobs returned to Apple in 1996 when Apple bought NeXT. Apple's OS X operating system is a descendant of NeXTSTEP. Apple's proprietary operating system, **iOS**, is derived from Apple's 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. Experienced Java programmers can quickly dive into Android de velopment. One benefit of developing Android apps is the openness of the platform. The operating system is open source and free.

The Android operating system was developed by Android, Inc., which was acquired by Google in 2005. In 2007, the Open Handset Alliance™—a consortium of 34 companies initially and 84 by 2011—was formed to continue developing Android. As of June 2012, more than 900,000 Android devices were being activated each day![7] Android smartphones are now outselling iPhones in the United States.[8] The Android operating system is used in numerous smartphones (such as the Motorola Droid, HTC One S, Samsung Galaxy Nexus and many more), e-reader devices (such as the Kindle Fire and Barnes and Noble Nook™), tablet computers (such as the Dell Streak and the Samsung Galaxy Tab), in-store touch-screen kiosks, cars, robots, multimedia players and more.

1.12 The Internet and World Wide Web

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 Re search Projects Agency) rolled out blueprints for networking the main computer systems of about a dozen ARPA-funded universities and research institutions. Academic research was about to take a giant leap forward. ARPA proceeded to implement the **ARPANET**, which eventually evolved into today's **Internet**. It rapidly became clear that communicating quickly and easily via electronic mail was the key early benefit of the ARPANET. This is true even today on the Internet, which facilitates communications of all kinds among the world's Internet users.

Packet Switching

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). The network op erated with a technique called **packet switching**, in which digital data was sent in small bundles called **packets**. The packets contained *address*, *error-control* and *sequencing* information. The address information allowed packets to be *routed* to their destinations. The sequencing information helped in *reassembling* the packets—which, because of complex

[6] www.tiobe.com/index.php/content/paperinfo/tpci/index.html.

[7] mashable.com/2012/06/11/900000-android-devices/.

[8] www.pcworld.com/article/196035/android_outsells_the_iphone_no_big_surprise.html.

routing mechanisms, could arrive out of order—into their original order for presentation to the recipient. Packets from different senders were intermixed on the *same* lines to efficiently use the available bandwidth. This packet-switching technique greatly reduced transmission costs, as compared with the cost of *dedicated* communications lines.

The network was designed to operate without centralized control. If a portion of the network failed, the remaining working portions would still route packets from senders to receivers over alternative paths for reliability.

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 net works. One challenge was to get these different networks to communicate. ARPA accom plished 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**.

World Wide Web, HTML, HTTP

The **World Wide Web** allows you to locate and view multimedia-based documents on al most any subject over the Internet. The web is a relatively recent creation. In 1989, Tim Berners-Lee of CERN (the European Organization for Nuclear Research) began to devel op a technology for sharing information via hyperlinked text documents. Berners-Lee called his invention the **HyperText Markup Language** (**HTML**). He also wrote commu nication 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. The **URL (Uniform Resource Locator)** specifies the address (i.e., location) of the web page displayed in the browser window. Each web page on the Internet is associated with a unique URL. **Hypertext Transfer Protocol Secure (HTTPS)** is the standard for transferring encrypted data on the web.

Mosaic, Netscape, Emergence of Web 2.0

Web use exploded with the availability in 1993 of the Mosaic browser, which featured a user-friendly graphical interface. Marc Andreessen, whose team at the National Center for Supercomputing Applications developed Mosaic, went on to found Netscape, the compa ny that many people credit with igniting theexplosive Interneteconomy of the late 1990s.

In 2003 there was a noticeable shift in how people and businesses were using the web and developing web-based applications. The term **Web 2.0** was coined by Dale Dougherty of O'Reilly Media[9]in 2003 to describe this trend. Generally, Web 2.0 companies use the web as a platform to create collaborative, community-based sites (e.g., social networking sites, blogs, wikis).

[9] T. O'Reilly, "What is Web 2.0: Design Patterns and Business Models for the Next Generation of Software." September 2005 <http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html?page=1>.

Companies with Web 2.0 characteristics are Google (web search), YouTube (video sharing), Facebook (social networking), Twitter (microblogging), Groupon (social com merce), Foursquare (mobile check-in), Salesforce (business software offered as online ser vices "in the cloud"), Craigslist (mostly free classified listings), Flickr (photo sharing), Skype (Internet telephony and video calling and conferencing) and Wikipedia (a free online encyclopedia).

Web 2.0 *involves* the users—not only do they create content, but they help organize it, share it, remix it, critique it, update it, etc. Web 2.0 is a *conversation*, with everyone having the opportunity to speak and share views. Companies that understand Web 2.0 realize that their products and services are conversations as well.

Architecture of Participation

Web 2.0 embraces an **architecture of participation**—a design that encourages user inter action and community contributions. You, the user, are the most important aspect of Web 2.0—so important, in fact, that in 2006, *TIME* magazine's "Person of the Year" was "You."[10] The article recognized the social phenomenon of Web 2.0—the shift away from a *powerful few* to an *empowered many*. Popular blogs now compete with traditional media powerhouses, and many Web 2.0 companies are built almost entirely on user-generated content. For websites like Facebook, Twitter, YouTube, eBay and Wikipedia users create the content, while the companies provide the *platforms* on which to enter, manipulate and share the information.

1.13 Some Key Software Development Terminology

Figure 1.27 lists a number of buzzwords that you'll hear in thesoftware development com munity. We've created Resource Centers on most of these topics, with more on the way.

Ajax **Ajax** is one of the premier Web 2.0 software technologies. Ajax helps Inter net-based applications perform like desktop applications—a difficult task, given that such applications suffer transmission delays as data is shuttled back and forth between your computer and servers on the Internet.

Agile software development **Agile software development** is a set of methodologies that try to get soft ware implemented faster and using fewer resources than previous method ologies. 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 include *refactoring tools* to do major portions of the reworking automatically.

Fig. 1.27 | Software technologies. (Part 1 of 2.)

[10] www.time.com/time/magazine/article/0,9171,1570810,00.html.

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.

LAMP **LAMP** is an acronym for the set of open-source technologies that many developers use to build web applications—it stands for Linux, Apache, MySQL and PHP (or Perl or Python—two other languages used for similar purposes). MySQL is an open-source database management system. PHP is the most popular open-source server-side Internet "scripting" language for developing Internet-based applications.

Software as a Service (SaaS)

Platform as a Ser vice (PaaS)

Software Develop ment Kit (SDK)

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 of the software appear, you upgrade your software, often requiring significant time and at considerable expense. This process can become cum bersome for organizations with tens of thousands of systems that must be maintained on a diverse array of computer equipment. 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 capabili ties—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. Sales force.com, Google, and Microsoft's Office Live and Windows Live all offer SaaS. SaaS is a capability of cloud computing.

Platform as a Service (PaaS), another capability of cloud computing, pro vides a computing platform for developing and running applications as a service over the web, rather than installing the tools on your computer. PaaS providers include Google App Engine, Amazon EC2, Bungee Labs and more.

Software Development Kits (SDKs) include the tools and documentation developers use to program applications.

Fig. 1.27 | Software technologies. (Part 2 of 2.)

Figure 1.28 describes software product-release categories.

Alpha An *alpha* version 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 fea tures, getting early feedback, etc.

Fig. 1.28 | Software product-release terminology. (Part 1 of 2.)

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* and (supposedly) 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. Any bugs that appear 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 generally does not have version numbers (for example, Google search or Gmail). The software, which is hosted in the cloud (not installed on your computer), is constantly evolving so that users always have the latest version.

Fig. 1.28 | Software product-release terminology. (Part 2 of 2.)

1.14 C++11 and the Open Source Boost Libraries

C++11 (formerly called C++0x)—the latest C++ programming language standard—was published by ISO/IEC in 2011. Bjarne Stroustrup, the creator of C++, expressed his vision for the future of the language—the main goals were to make C++ easier to learn, improve library building capabilities and increase compatibility with the C programming language. The new standard extends the C++ Standard Library and includes several features and enhancements to improve performance and security. The major C++ compiler vendors have already implemented many of the new C++11 features (Fig. 1.29). Throughout the book, we discuss various key features of C++11. For more information, visit the C++ Standards Committee website at www.open-std.org/jtc1/sc22/wg21/ and isocpp.org. Copies of the C++11 language specification (ISO/IEC 14882:2011) can be purchased at:

http://bit.ly/CPlusPlus11Standard

C++11 features implemented in each of the major C++ compilers.	wiki.apache.org/stdcxx/C%2B%2B0xCompilerSupport
Microsoft® Visual C++	msdn.microsoft.com/en-us/library/hh567368.aspx
GNU Compiler Collection (g++)	gcc.gnu.org/projects/cxx0x.html

Intel® C++ Compiler	software.intel.com/en-us/articles/c0x-features supported-by-intel-c-compiler/

Fig. 1.29 | C++ compilers that have implemented major portions of C++11.

IBM® XL C/C++	www.ibm.com/developerworks/mydeveloperworks/ blogs/5894415f-be62-4bc0-81c5-3956e82276f3/ entry/xlc_compiler_s_c_11_support50?lang=en
Clang	clang.llvm.org/cxx_status.html
EDG ecpp	www.edg.com/docs/edg_cpp.pdf

Fig. 1.29 | C++ compilers that have implemented major portions of C++11.

Boost C++ Libraries

The **Boost C++ Libraries** are free, open-source libraries created by members of the C++ community. They are peer reviewed and portable across many compilers and platforms. Boost has grown to over 100 libraries, with more being added regularly. Today there are thousands of programmers in the Boost open source community. Boost provides C++ pro grammers with useful libraries that work well with the existing C++ Standard Library. The Boost libraries can be used by C++ programmers working on a wide variety of platforms with many different compilers. Some of the new C++11 Standard Library features were derived from corresponding Boost libraries. We overview the libraries and provide code examples for the "regular expression" and "smart pointer" libraries, among others.

Regular expressions are used to match specific character patterns in text. They can be used to validate data to ensure that it's in a particular format, to replace parts of one string with another, or to split a string.

Many common bugs in C and C++ code are related to pointers, a powerful program ming capability that C++ absorbed from C. As you'll see, **smart pointers** help you avoid errors associated with traditional pointers.

1.15 Keeping Up to Date with Information Technologies

Figure 1.30 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.

ACM TechNews	technews.acm.org/
ACM Transactions on Accessible Computing	www.gccis.rit.edu/taccess/index.html
ACM Transactions on Internet Technology	toit.acm.org/
Bloomberg BusinessWeek	www.businessweek.com
CNET	

news.cnet.com Communications of the ACM
cacm.acm.org/

Fig. 1.30 | Technical and business publications. (Part 1 of 2.)

Computerworld www.computerworld.com

Engadget www.engadget.com

eWeek www.eweek.com

Fast Company www.fastcompany.com/

Fortune money.cnn.com/magazines/fortune/

IEEE Computer www.computer.org/portal/web/computer IEEE Internet Computing www.computer.org/portal/web/internet/home InfoWorld www.infoworld.com

Mashable mashable.com

PCWorld www.pcworld.com

SD Times www.sdtimes.com

Slashdot slashdot.org/

Smarter Technology www.smartertechnology.com

Technology Review technologyreview.com

Techcrunch techcrunch.com

Wired www.wired.com

Fig. 1.30 | Technical and business publications. (Part 2 of 2.)

1.16 Web Resources

This section provides links to our C++ and related Resource Centers that will be useful to you as you learn C++. These include blogs, articles, whitepapers, compilers, development tools, downloads, FAQs, tutorials, webcasts, wikis and links to C++ game programming resources. For updates on Deitel publications, Resource Centers, training courses, partner offers and more, follow us on Facebook® at www.facebook.com/deitelfan/, Twitter® @deitel, Google+ at gplus.to/deitel and LinkedIn at bit.ly/DeitelLinkedIn.

Deitel & Associates Websites

www.deitel.com/books/cpphtp9/
The Deitel & Associates *C++ How to Program, 9/e* site. Here you'll find links to the book's examples and other resources.

www.deitel.com/cplusplus/
www.deitel.com/visualcplusplus/
www.deitel.com/codesearchengines/
www.deitel.com/programmingprojects/
Check these Resource Centers for compilers, code downloads, tutorials, documentation, books, e

books, articles, blogs, RSS feeds and more that will help you develop C++ applications.

www.deitel.com

Check this site for updates, corrections and additional resources for all Deitel publications.

www.deitel.com/newsletter/subscribe.html

Subscribe here to the *Deitel*® *Buzz Online* e-mail newsletter to follow the Deitel & Associates pub lishing program, including updates and errata to *C++ How to Program, 9/e.*

Self-Review Exercises

1.1 Fill in the blanks in each of the following statements:

a) Computers process data under the control of sets of instructions called . b) The key logical units of the computer are the , , , , and .

c) The three types of languages discussed in the chapter are , and .

d) The programs that translate high-level language programs into machine language are called .

e) is an operating system for mobile devices based on the Linux kernel and Java. f) software is generally feature complete and (supposedly) bug free and ready for use by the community.

g) The Wii Remote, as well as many smartphones, uses a(n) which allows the de vice to respond to motion.

1.2 Fill in the blanks in each of the following sentences about the C++ environment. a) C++ programs are normally typed into a computer using a(n) program. b) In a C++ system, a(n) program executes before the compiler's translation phase begins.

c) The program combines the output of the compiler with various library func tions to produce an executable program.

d) The program transfers the executable program from disk to memory.

1.3 Fill in the blanks in each of the following statements (based on Section 1.8): a) Objects have the property of —although objects may know how to commu nicate with one another across well-defined interfaces, they normally are not allowed to know how other objects are implemented.

b) C++ programmers concentrate on creating , which contain data members and the member functions that manipulate those data members and provide services to clients. c) The process of analyzing and designing a system from an object-oriented point of view is called .

d) With , new classes of objects are derived by absorbing characteristics of existing classes, then adding unique characteristics of their own.

e) is a graphical language that allows people who design software systems to use an industry-standard notation to represent them.

f) The size, shape, color and weight of an object are considered of the object's class.

Answers to Self-Review Exercises

1.1 a) programs. b) input unit, output unit, memory unit, central processing unit, arithmetic and logic unit, secondary storage unit. c) machine languages, assembly languages, high-level lan guages. d) compilers. e) Android. f) Release candidate. g) accelerometer.

1.2 a) editor. b) preprocessor. c) linker. d) loader.

1.3 a) information hiding. b) classes. c) object-oriented analysis and design (OOAD). d) inheritance. e) The Unified Modeling Language (UML). f) attributes.

Exercises

1.4 Fill in the blanks in each of the following statements:

a) The logical unit of the computer that receives information from outside the computer for use by the computer is the .

b) The process of instructing the computer to solve a problem is called . c) is a type of computer language that uses English-like abbreviations for ma chine-language instructions.

d) is a logical unit of the computer that sends information which has already been processed by the computer to various devices so that it may be used outside the computer.

e) and are logical units of the computer that retain information. f) is a logical unit of the computer that performs calculations.

g) is a logical unit of the computer that makes logical decisions. h) languages are most convenient to the programmer for writing programs quickly and easily.

i) The only language a computer can directly understand is that computer's . j) is a logical unit of the computer that coordinates the activities of all the other logical units.

1.5 Fill in the blanks in each of the following statements:

a) initially became widely known as the development language of the Unix op erating system.

b) The programming language was developed by Bjarne Stroustrup in the early 1980s at Bell Laboratories.

1.6 Fill in the blanks in each of the following statements:

a) C++ programs normally go through six phases— , , , , and .

b) A(n) provides many tools that support the software development process, such as editors for writing and editing programs, debuggers for locating logic errors in programs, and many other features.

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

Making a Difference

Throughout the book we've included Making a Difference exercises in which you'll be asked to work on problems that really matter to individuals, communities, countries and the world. For more information about worldwide organizations working to make a difference, and for related programming project ideas, visit our Making a Difference Resource Center at www.deitel.com/makingadifference.

1.8 ***(Test Drive: Carbon Footprint Calculator)*** Some scientists believe that carbon emissions, especially from the burning of fossil fuels, contribute significantly to global warming and that this can be combatted if individuals take steps to limit their use of carbon-based fuels. Various organizations and individuals are increasingly concerned about their "carbon footprints." Websites such as TerraPass

www.terrapass.com/carbon-footprint-calculator/

and Carbon Footprint

www.carbonfootprint.com/calculator.aspx

provide carbon footprint calculators. Test drive these calculators to determine your carbon footprint. Exercises in later chapters will ask you to program your own carbon footprint calculator. To prepare for this, research the formulas for calculating carbon footprints.

1.9 ***(Test Drive: Body Mass Index Calculator)*** By recent estimates, two-thirds of the people in the United States are overweight and about half of those are obese. This causes significant increases in illnesses such as diabetes and heart disease. To determine whether a person is overweight or obese, you can use a measure called the body mass index (BMI). The United States Department of Health and Human Services provides a BMI calculator at www.nhlbisupport.com/bmi/. Use it to calculate your own BMI. An exercise in Chapter 2 will ask you to program your own BMI calculator. To prepare for this, research the formulas for calculating BMI.

1.10 ***(Attributes of Hybrid Vehicles)*** In this chapter you learned the basics of classes. Now you'll begin "fleshing out" aspects of a class called "Hybrid Vehicle." Hybrid vehicles are becoming increasingly popular, because they often get much better mileage than purely gasoline-powered vehicles. Browse the web and study the features of four or five of today's popular hybrid cars, then list as many of their hybrid-related attributes as you can. For example, common attributes include city miles-per-gallon and highway-miles-per-gallon. Also list the attributes of the batteries (type, weight, etc.).

1.11 ***(Gender Neutrality)*** Some people want to eliminate sexism in all forms of communication. You've been asked to create a program that can process a paragraph of text and replace gender-specific words with gender-neutral ones. Assuming that you've been given a list of gender-specific words and their gender-neutral replacements (e.g., replace "wife" by "spouse," "man" by "person," "daughter" by "child" and so on), explain the procedure you'd use to read through a paragraph of text and manually perform these replacements. How might your procedure generate a strange term like "woperchild," which is actually listed in the Urban Dictionary (www.urbandictionary.com)? In Chapter 4, you'll learn that a more formal term for "procedure" is "algorithm," and that an algorithm specifies the steps to be performed and the order in which to perform them.

1.12 ***(Privacy)*** Some online email services save all email correspondence for some period of time. Suppose a disgruntled employee of one of these online email services were to post all of the email correspondences for millions of people, including yours, on the Internet. Discuss the issues.

1.13 ***(Programmer Responsibility and Liability)*** As a programmer in industry, you may develop software that could affect people's health or even their lives. Suppose a software bug in one of your programs were to cause a cancer patient to receive an excessive dose during radiation therapy and that the person is either severely injured or dies. Discuss the issues.

1.14 ***(2010 "Flash Crash")*** An example of the consequences of our dependency on computers was the so-called "flash crash" which occurred on May 6, 2010, when the U.S. stock market fell precipitously in a matter of minutes, wiping out trillions of dollars of investments, and then recovered within minutes. Use the Internet to investigate the causes of this crash and discuss the issues it raises.

Making a Difference Resources

The *Microsoft Image Cup* is a global competition in which students use technology to try to solve some of the world's most difficult problems, such as environmental sustainability, ending hunger, emergency response, literacy, combating HIV/AIDS and more. For more information about the competition and to learn about previous winners' projects, visit www.imaginecup.com/about. You can also find several project ideas submitted by worldwide charitable organizations.

For additional ideas for programming projects that can make a difference, search the web for "making a difference" and visit the following websites:

www.un.org/millenniumgoals

The United Nations Millennium Project seeks solutions to major worldwide issues such as environmental sustainability, gender equality, child and maternal health, universal education and more.

www.ibm.com/smarterplanet/

The IBM® Smarter Planet website discusses how IBM is using technology to solve issues related to business, cloud computing, education, sustainability and more.

www.gatesfoundation.org/Pages/home.aspx

The Bill and Melinda Gates Foundation provides grants to organizations that work to alleviate hun ger, poverty and disease in developing countries. In the U.S., the foundation focusses on improving public education, particularly for people with few resources.

www.nethope.org/

NetHope is a collaboration of humanitarian organizations worldwide working to solve technology problems such as connectivity, emergency response and more.

www.rainforestfoundation.org/home

The Rainforest Foundation works to preserve rainforests and to protect the rights of the indigenous people who call the rainforests home. The site includes a list of things you can do to help.

www.undp.org/

The United Nations Development Programme (UNDP) seeks solutions to global challenges such as crisis prevention and recovery, energy and the environment, democratic governance and more.

www.unido.org

The United Nations Industrial Development Organization (UNIDO) seeks to reduce poverty, give developing countries the opportunity to participate in global trade, and promote energy efficiency and sustainability.

www.usaid.gov/

USAID promotes global democracy, health, economic growth, conflict prevention, humanitarian aid and more.

www.toyota.com/ideas-for-good/

Toyota's Ideas for Good website describes several Toyota technologies that are making a difference— including their Advanced Parking Guidance System, Hybrid Synergy Drive®, Solar Powered Venti lation System, T.H.U.M.S. (Total Human Model for Safety) and Touch Tracer Display. You can par ticipate in the Ideas for Good challenge by submitting a short essay or video describing how these technologies can be used for other good purposes.

Introduction to C++

Programming, Input/Output and Operators

What's in a name? that
which we call a rose
By any other name
would smell as sweet.
—William Shakespeare

High thoughts must have high
language.

—Aristophanes

One person can make a difference and every person should try.
—John F. Kennedy

Objectives

In this chapter you'll learn:

- To write simple computer programs in C++.
- To write simple input and output statements.
- To use fundamental types.
- Basic computer memory concepts.
- To use arithmetic operators.
- The precedence of arithmetic operators.
- To write simple decision making statements.

2'# !0*" 3 0 "!

2(& #0+% 3 0 %#

! 14M ,1G@4 J ! :: 6@4)@ 00,1) M*, * & ,/,G G!A ,A ,6/,1! 66@4 * G4 6@4)@ 0 !L!/460!1G9

4AG 4& G*! :: 6@4)@ 0A O4J?// AGJ O ,1 G*,A 44. 6@4 !AA G 1 ,A6/ O @!AJ/GA9 1 G*,A * 6G!@ M! 6@!A!1G &,L! !N 06/!A G* G !041AG@ G! *4M O4J@ 6@4)@ 0A 1 ,A6/ O 0!AA)!A 1 4 G ,1 G &@40 G*! JA!@

&4@ 6@4 !AA,1)9 *! &,@AG G*@!! !N 06/!A A,06/O ,A6/ O 0!AA)!A 41 G*! A @!!19 *! 1!NG 4 G ,1A GM4 1J0 !@A &@40 JA!@ / J+ / G!A G*!,@ AJ0 1 ,A6/ OA G*! @!AJ/G9 *! 406 1O,1) ,A JAA,41 A*4MA O4J *4M G4 6!@&4@0 6%;#(;% ' >' ;%+)7 1 A L! G*!,@ @!AJ/GA &4@ / G!@ JA!9 *! &,&G* !N 06/! !0+ 41AG@ G!A %7%+)$(&%)" O A*4M,1) O4J *4M G4 +(- 6 GM4 1J0 !@A G*!1 ,A6/ O 0!A+ A)!A A! 41 G*! 406 @,A41 @!AJ/GA9 ! 1 /OP! ! * 6@4)@ 0 41! /,1! G G,0! G4 *!/6 O4J ! A! O4J@ M O ,1G4 :: 6@4)@ 00,1)9

'%("$"& & 7&&"& 0' 0 %1

G !!!"#$%&$"'()*+,))-.+(//0&/1 M!?L! 64AG! L, !4A G* G !041AG@ G! 406,/,1) 1 @J11,1) 6@4)@ 0A ,1 , @4A4&G ,AJ / :: :: 1 4 !9

2(2 +,0 +% + " #)) + #0 # # % 60

41A, !@ A,06/! 6@4)@ 0 G* G 6@,1GA /,1! 4& G!NG 7 ,)9 I9589 *,A 6@4)@ 0 ,//JAG@ G!A A!L+ !@ / ,064@G 1G &! GJ@!A 4& G*! :: / 1)J)!9 *! G!NG ,1 /,1!A 5$55 ,A G*! 6@4)@ 0?A 7+>6 + 74@ + 89 *! /,1! 1J0 !@A @! 14G 6 @G 4& G*! A4J@ ! 4 !9

```
  ++ 2%3" 4"56 7%384985"(//

++ :$;&</=%>&%>3 /=)3=?*"
   2  @%).&=$?*A ++ ?")!. /=)3=?* &) )B&/B& #?&? &) &0$ .(=$$>
 ++ 7B>(&%)> *?%> ,$3%>. /=)3=?* $;$(B&%)>
  / *?%>CD
 E
 .&#66()B& @@ &     /  $$  &F ++ #%./'?G *$..?3$

( /2(  7F ++ %>#%(?&$ &0?&  /=)3=?* $>#$# .B(($..7B"G    H ++ $># 7B>(&%)>
*?%>
```

I$'()*$ &) JKKL

(2(& 2- #* - #*! * &

Comments

Lines 1 and 2

```
// Fig. 2.1: fig02_01.cpp
// Text-printing program.
```

each begin with //, indicating that the remainder of each line is a **comment**. You insert comments to *document* your programs and to help other people read and understand them. Comments do not cause the computer to perform any action when the program is run—they're *ignored* by the C++ compiler and do *not* cause any machine-language object code to be generated. The comment Text-printing program describes the purpose of the program. A comment beginning with // is called a **single-line comment** because it termi nates at the end of the current line. [*Note:* You also may use comments containing one or more lines enclosed in /* and */.]

Good Programming Practice 2.1

Every program should begin with a comment that describes the purpose of the program.

#include *Preprocessing Directive*

Line 3

```
#include <iostream> // allows program to output data to the screen
```

is a **preprocessing directive**, which is a message to the C++ preprocessor (introduced in Section 1.9). Lines that begin with # are processed by the preprocessor *before* the program is compiled. This line notifies the preprocessor to include in the program the contents of the **input/output stream header** <iostream>. This header is a file containing information used by the compiler when compiling any program that outputs data to the screen or in puts data from the keyboard using C++'s stream input/output. The program in Fig. 2.1 outputs data to the screen, as we'll soon see. We discuss headers in more detail in Chapter 6 and explain the contents of <iostream> in Chapter 13.

Common Programming Error 2.1

Forgetting to include the <iostream> *header in a program that inputs data from the key board or outputs data to the screen causes the compiler to issue an error message.*

Blank Lines and White Space

Line 4 is simply a *blank line*. You use blank lines, *space characters* and *tab characters* (i.e., "tabs") to make programs easier to read. Together, these characters are known as **white space**. White-space characters are normally *ignored* by the compiler.

The main *Function*

Line 5

```
// function main begins program execution
```

is another single-line comment indicating that program execution begins at the next line.

Line 6

```
int main()
```

is a part of every C++ program. The parentheses after main indicate that main is a program building block called a **function**. C++ programs typically consist of one or more functions and classes (as you'll learn in Chapter 3). Exactly *one* function in every program *must* be named main. Figure 2.1 contains only one function. C++ programs begin executing at function main, even if main is *not* the first function defined in the program. The keyword int to the left of main indicates that main "returns" an integer (whole number) value. A **keyword** is a word in code that is reserved by C++ for a specific use. The complete list of C++ keywords can be found in Fig. 4.3. We'll explain what it means for a function to "re turn a value" when we demonstrate how to create your own functions in Section 3.3. For now, simply include the keyword int to the left of main in each of your programs.

The **left brace**, {, (line 7) must *begin* the **body** of every function. A corresponding

right brace, }, (line 11) must *end* each function's body.

An Output Statement

Line 8

```
    std::cout << "Welcome to C++!\n"; // display message
```

instructs the computer to **perform an action**—namely, to print the characters contained between the double quotation marks. Together, the quotation marks and the characters between them are called a **string**, a **character string** or a **string literal**. In this book, we refer to characters between double quotation marks simply as strings. White-space charac ters in strings are not ignored by the compiler.

The entire line 8, including `std::cout`, the **<< operator**, the string `"Welcome to C++!\n"` and the **semicolon** (;), is called a **statement**. Most C++ statements end with a semicolon, also known as the**statement terminator** (we'll seesomeexceptions to this soon). Preprocessing directives (like `#include`) do not end with a semicolon. Typically, output and input in C++ are accomplished with **streams** of characters. Thus, when the preceding statement is executed, it sends the stream of characters `Welcome to C++!\n` to the **standard output stream object**—`std::cout`—which is normally "connected" to the screen.

Common Programming Error 2.2

Omitting the semicolon at the end of a C++ statement is a syntax error. The ***syntax*** *of a programming language specifies the rules for creating proper programs in that language. A* ***syntax error*** *occurs when the compiler encounters code that violates C++'s language rules (i.e., its syntax). The compiler normally issues an error message to help you locate and fix the incorrect code. Syntax errors are also called* ***compiler errors****,* ***compile-time errors*** *or* ***compilation errors****, because the compiler detects them during the compilation phase. You cannot execute your program until you correct all the syntax errors in it. As you'll see, some compilation errors are not syntax errors.*

Good Programming Practice 2.2

Indent the body of each function one level withinthe braces that delimit the function's body. This makes a program's functional structure stand out and makes the program easier to read.

Good Programming Practice 2.3

Set a convention for the size of indent you prefer,then apply it uniformly. The tab key may be used to create indents, but tab stops may vary. We prefer three spaces per level of indent.

The std *Namespace*

The `std::` before `cout` is required when we use names that we've brought into the pro gram by the preprocessing directive `#include <iostream>`. The notation `std::cout` spec ifies that we are using a name, in this case `cout`, that belongs to namespace `std`. The names `cin` (the standard input stream) and `cerr` (the standard error stream)—introduced in Chapter 1—also belong to namespace `std`. Namespaces are an advanced C++ feature that we discuss in depth in Chapter 23, Other Topics. For now, you should simply remember to include `std::` before each mention of `cout`, `cin` and `cerr` in a program. This can be cumbersome—the next example introduces `using` declarations and the `using` directive, which will enable you to omit `std::` before each use of a name in the `std` namespace.

The Stream Insertion Operator and Escape Sequences

In the context of an output statement, the << operator is referred to as the **stream insertion operator**. When this program executes, the valueto the operator's right, the right **operand**, is inserted in the output stream. Notice that the operator points in the direction of where the data goes. A string literal's characters *normally* print exactly as they appear between the double quotes. However, the characters \n are *not* printed on the screen (Fig. 2.1). The backslash (\) is called an **escape character**. It indicates that a "special" character is to be output. When a backslash is encountered in a string of characters, the next character is combined with the backslash to form an **escape sequence**. The escape sequence \n means **newline**. It causes the **cursor** (i.e., the current screen-position indicator) to move to the beginning of the next line on the screen. Some common escape sequences are listed in Fig. 2.2.

\n Newline. Position the screen cursor to the beginning of the next line.\t Horizontal tab. Move the screen cursor to the next tab stop. \r Carriage return. Position the screen cursor to the beginning of the current line; do not advance to the next line.

\a Alert. Sound the system bell.

\\ Backslash. Used to print a backslash character.

\' Single quote. Used to print a single quote character.

\" Double quote. Used to print a double quote character.

Fig. 2.2 | Escape sequences.

The return *Statement*

Line 10

```
return 0; // indicate that program ended successfully
```

is one of several means we'll use to **exit a function**. When the return **statement** is used at the end of main, as shown here, the value 0 indicates that the program has *terminated suc cessfully*. The right brace, }, (line 11) indicates the end of function main. According to the

C++ standard, if program execution reaches the end of main without encountering a re turn statement, it's assumed that the program terminated successfully—exactly as when the last statement in main is a return statement with the value 0. For that reason, we *omit* the return statement at the end of main in subsequent programs.

A Note About Comments

As you write a new program or modify an existing one, you should *keep your comments up to-date* with the program's code. You'll *often* need to make changes to existing programs—for example, to fix errors (commonly called *bugs*) that prevent a program from working correctly or to enhance a program. Updating your comments as you make code changes helps ensure that the comments accurately reflect what the code does. This will make your programs easier to understand and modify in the future.

2.3 Modifying Our First C++ Program

We now present two examples that modify the program of Fig. 2.1 to print text on one line by using multiple statements and to print text on several lines by using a single statement.

Printing a Single Line of Text with Multiple Statements

`Welcome to C++!` can be printed several ways. For example, Fig. 2.3 performs stream insertion in multiple statements (lines 8–9), yet produces the same output as the program of Fig. 2.1. [*Note:* From this point forward, we use a *light blue background* to highlight the key features each program introduces.] Each stream insertion resumes printing where the previous one stopped. The first stream insertion (line 8) prints `Welcome` followed by a space, and because this string did not end with `\n`, the second stream insertion (line 9) begins printing on the *same* line immediately following the space.

```
 1 // Fig. 2.3: fig02_03.cpp
 2 // Printing a line of text with multiple statements.
 3 #include <iostream> // allows program to output data to the screen 4
 5 // function main begins program execution
 6 int main()
 7 {
 8
            std::cout << "Welcome ";

              std::cout << "to C++!\n";

 9
10 } // end function main
```

```
Welcome to C++!
```

Fig. 2.3 | Printing a line of text with multiple statements.

Printing Multiple Lines of Text with a Single Statement

A single statement can print multiple lines by using newline characters, as in line 8 of Fig. 2.4. Each time the `\n` (newline) escape sequence is encountered in the output stream, the screen cursor is positioned to the beginning of the next line. To get a blank line in your output, place two newline characters back to back, as in line 8.

```
 1 // Fig. 2.4: fig02_04.cpp
 2 // Printing multiple lines of text with a single statement. 3 #include <iostream> //
 allows program to output data to the screen 4
 5 // function main begins program execution
 6 int main()
 7 {
 8 std::cout << "Welcome to C++!\n";
9 } // end       function main
```

```
                n           \n
Welcome
to

C++!
     \           \n
```

Fig. 2.4 | Printing multiple lines of text with a single statement.

2.4 Another C++ Program: Adding Integers

Our next program obtains two integers typed by a user at the keyboard, computes the sum of these values and outputs the result using std::cout. Figure 2.5 shows the program and sample inputs and outputs. In the sample execution, we highlight the user's input in bold. The program begins execution with function main (line 6). The left brace (line 7) begins main's body and the corresponding right brace (line 22) ends it.

```
 1 // Fig. 2.5: fig02_05.cpp
2 // Addition program that displays the sum of two integers. 3 #include <iostream>
// allows program to perform input and output 4
5 // function main begins program execution
6 int main()
7 {
8 // variable declarations
9
      int number1 = 0; // first integer to add (initialized to 0)
      int number2 = 0; // second integer to add (initialized to 0)
         int sum = 0; // sum of number1 and number2 (initialized to 0)

10
11

12
13 std::cout << "Enter first integer: "; // prompt user for data
14
         std::cin >> number1; // read first integer from user into number1

15
16 std::cout << "Enter second integer: "; // prompt user for data
17
         std::cin >> number2; // read second integer from user into number2
sum = number1 + number2; // add the numbers; store result in sum

std::cout << "Sum is " << sum << ; // display sum; end line
                                    l
} // end function main
```

```
Enter first integer: 45 Enter
second integer: 72 Sum is 117
              std::end
```

Fig. 2.5 | Addition program that displays the sum of two integers.