

Deitel® How to Program Series Cover Theme

The cover theme for the DEITEL® HOW TO PROGRAM SERIES emphasizes social consciousness issues such as going green, clean energy, recycling, sustainability and more. Within the text, in addition to conventional programming exercises, we've included our Making a Difference exercise set to raise awareness of issues such as global warming, population growth, affordable healthcare, accessibility, privacy of electronic records and more. In this book, you'll use C++ to program applications that relate to these issues.

We hope that what you learn in *C++ How to Program, 8/e* will help you to make a difference.



Rainforests

The world's rainforests are often referred to as the "Earth's lungs," the "jewels of the Earth" and the "world's largest pharmacy." Approximately 50% of the world's tropical rainforests are in Central and South America, over 33% are in Asia and Oceania (which consists of Australia, New Zealand and various South Pacific Islands), and 15% are in Africa. Rainforests absorb from the atmosphere vast amounts of carbon dioxide—a gas that many scientists blame for global warming—and they provide approximately 40% of the world's oxygen. They regulate water flow to surrounding areas preventing mudslides and crop loss. Rainforests also support the livelihoods of 1.6 billion people, providing food, fresh water, medicines and more. Approximately 25% of Western medicines used to treat infections, viruses, cancer and more are derived from plants

found in rainforests. The U.S. National Cancer Institute has found about 2100 rainforest plant species that are effective against cancer cells. Fewer than one percent of rainforest plant species have been

tested for medical use.

Rainforests are being deforested at an alarming rate. According to a March 2010 report by the United Nations Food and Agriculture Organization, deforestation has slowed over the last 10 years, but more than 30 million acres of forests are still lost annually, and they're not easily renewed. The United Nations Environment Programme Plant for the Planet: Billion Tree Campaign is one of many reforestation initiatives. To learn more about how you can make a difference, visit www.unep.org/billiontreecampaign/index.asp. For further information visit:

www.rain-tree.com/facts.htm

www.savetherainforest.org/savetherainforest_007.htm

en.wikipedia.org/wiki/Rainforest

www.rainforestfoundation.org/

About Deitel & Associates, Inc.

Deitel & Associates, Inc., is an internationally recognized authoring and corporate training organization. The company offers instructor-led courses delivered at client sites worldwide on programming languages and other software topics such as C++, Visual C++[®], C, Java[™], C#[®], Visual Basic[®], Objective-C[®], XML[®], Python[®], JavaScript, object technology, Internet and web programming, and Android and iPhone app development. The company's clients include many of the world's largest companies, as well as government agencies, branches of the military and academic institutions. To learn more about Deitel Pearson Higher Education publications and Dive Into[®] Series corporate training, e-mail deitel@deitel.com or visit www.deitel.com/training/. Follow Deitel on Facebook[®] at www.deitel.com/deitelfan/ and on Twitter[®] @deitel.

Deitel[®] Series Page

How To Program Series

C++ How to Program, 8/E

C How to Program, 6/E

Java[™] How to Program, 9/E

Java[™] How to Program, Late Objects Version,

8/E Internet & World Wide Web How to

Program, 4/E Visual C++[®] 2008 How to

Program, 2/E Visual Basic[®] 2010 How to

Program Visual C#[®] 2010 How to Program,

3/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 C#: An App-Driven Tutorial Approach

Simply Visual Basic[®] 2008, 3/E: An

App-Driven Tutorial Approach

CourseSmart Web Books

www.deitel.com/books/CourseSmart/ C++

How to Program, 5/E, 6/E, 7/E & 8/E Simply

C++: An App-Driven Tutorial Approach Java[™]

How to Program, 6/E, 7/E, 8/E&9/E *(continued)*

(continued)

Simply Visual Basic 2008: An App-Driven
Tutorial Approach, 3/E

Visual Basic[®] 2010 How to Program

Visual Basic[®] 2008 How to Program

Visual C#[®] 2010 How to Program,

4/E Visual C#[®] 2008 How to

Program, 3/E

Deitel[®] Developer Series

C++ for Programmers

AJAX, Rich Internet Applications and Web
Development for Programmers

Android for Programmers: An

App-Driven Approach

C# 2010 for Programmers, 3/E

iPhone for Programmers: An App-Driven

Approach Java[™] for Programmers

JavaScript for Programmers

LiveLessonsVideo Learning Products

www.deitel.com/books/LiveLessons/ C++

Fundamentals

Java[™] Fundamentals

next column)

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

follow us on Twitter®

@deitel

and become a Deitel & Associates fan on Facebook®

www.deitel.com/deitelfan/

To communicate with the authors, send e-mail to:

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/

Check out our Resource Centers for valuable web resources that will help you master C++, other important programming languages, software, 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*

Editor-in-Chief: *Michael Hirsch*

Associate Editor: *Carole Snyder*

Vice President, Marketing: *Patrice Jones*

Marketing Manager: *Yezan Alayan*

Senior Marketing Coordinator: *Kathryn Ferranti*

Vice President, Production: *Vince O'Brien*

Managing Editor: *Jeff Holcomb*

Associate Managing Editor: *Robert Engelhardt*

Operations Specialist: *Lisa McDowell*

Art Director: *Linda Knowle*

Cover Design: *Abbey S. Deitel, Harvey M. Deitel, Marta Samsel*

Cover Photo Credit: © *James Hardy/PhotoAlto/Getty Images*

Media Editor: *Daniel Sandin*

Media Project Manager: *Wanda Rockwell*

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.

Copyright © 2012, 2008, 2005, 2003, 2001 Pearson Education, Inc., publishing as Prentice Hall. All rights reserved. Manufactured in the United States of America. This publication is protected by Copyright, and permission should be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. To obtain permission(s) to use

material from this work, please submit a written request to Pearson Education, Inc., Permissions Department, 501 Boylston Street, Suite 900, Boston, Massachusetts 02116.

Many of the designations by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed in initial caps or all caps.

Library of Congress Cataloging-in-Publication Data

Deitel, Paul J.

C++ : how to program / P.J. Deitel, H.M. Deitel. -- 8th ed.

p. cm.

Includes index.

ISBN 978-0-13-266236-9

1. C++ (Computer program language) I. Deitel, Harvey M. II. Title.

QA76.73.C153D45 2012

005.13'3--dc22

2011000245

10 9876 543 2 1

ISBN-10: 0-13-266236-1

ISBN-13: 978-0-13-266236-9

*In memory of Ken Olsen,
Founder of Digital Equipment Corporation (DEC):*

*We are deeply grateful for the
opportunities DEC extended to us,
enabling us to form and grow Deitel &
Associates, Inc.*

Paul and Harvey Deitel

Trademarks

DEITEL, the double-thumbs-up bug and DIVE INTO are registered trademarks of Deitel and Associates, Inc.

Microsoft and the Windows logo are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

UNIX is a registered trademark of The Open Group.

Throughout this book, trademarks are used. Rather than put a trademark symbol in every occurrence of a trademarked name, we state that we are using the names in an editorial fashion only and to the benefit of the trademark owner, with no intention of infringement of the trademark.

Chapters 25–26 and Appendices F–I are PDF documents posted online at the book’s Companion Website, which is accessible from www.pearsonhighered.com/deitel.

Preface xxi

1 Introduction to Computers and C++ 1 1.1

Introduction 2 1.2 Computers: Hardware and Software 5 1.3 Data Hierarchy 6
1.4 Computer Organization 7 1.5 Machine Languages, Assembly Languages and
High-Level Languages 9 1.6 Introduction to Object Technology 10 1.7
Operating Systems 13 1.8 Programming Languages 15 1.9 C++ and a Typical
C++ Development Environment 17 1.10 Test-Driving a C++ Application 21
1.11 Web 2.0: Going Social 27 1.12 Software Technologies 29 1.13 Future of
C++: TR1, the New C++ Standard and the Open Source
Boost Libraries 31 1.14 Keeping Up-to-Date with Information
Technologies 32 1.15 Wrap-Up 32

2 Introduction to C++ Programming 37 2.1 Introduction

38 2.2 First Program in C++: Printing a Line of Text 38 2.3 Modifying Our First
C++ Program 42 2.4 Another C++ Program: Adding Integers 43 2.5 Memory
Concepts 47 2.6 Arithmetic 48 2.7 Decision Making: Equality and Relational
Operators 51 2.8 Wrap-Up 55
viii [Contents](#)

3 Introduction to Classes, Objects and Strings 64

3.1 Introduction 65 3.2 Defining a Class with a Member Function 65 3.3
Defining a Member Function with a Parameter 68 3.4 Data Members, *set*
Functions and *get* Functions 71 3.5 Initializing Objects with Constructors 77 3.6
Placing a Class in a Separate File for Reusability 81 3.7 Separating Interface from
Implementation 84 3.8 Validating Data with *set* Functions 90 3.9 Wrap-Up 95

4 Control Statements: Part 1 101 4.1 Introduction 102 4.2

Algorithms 102 4.3 Pseudocode 103 4.4 Control Structures 104 4.5 if Selection
Statement 107 4.6 if...else Double-Selection Statement 108 4.7 while Repetition
Statement 113 4.8 Formulating Algorithms: Counter-Controlled Repetition 114
4.9 Formulating Algorithms: Sentinel-Controlled Repetition 120 4.10
Formulating Algorithms: Nested Control Statements 130 4.11 Assignment
Operators 134 4.12 Increment and Decrement Operators 135 4.13 Wrap-Up 138

5 Control Statements: Part 2 152 5.1 Introduction 153 5.2

Essentials of Counter-Controlled Repetition 153 5.3 **for** Repetition Statement 155 5.4 Examples Using the **for** Statement 158 5.5 **do...while** Repetition Statement 162 5.6 **switch** Multiple-Selection Statement 164 5.7 **break** and **continue** Statements 173 5.8 Logical Operators 174 5.9 Confusing the Equality (**==**) and Assignment (**=**) Operators 179 5.10 Structured Programming Summary 180 5.11 Wrap-Up 185

[Contents ix](#)

6 Functions and an Introduction to Recursion 194

6.1 Introduction 195 6.2 Program Components in C++ 196 6.3 Math Library Functions 197 6.4 Function Definitions with Multiple Parameters 198 6.5 Function Prototypes and Argument Coercion 203 6.6 C++ Standard Library Headers 205 6.7 Case Study: Random Number Generation 207 6.8 Case Study: Game of Chance; Introducing **enum** 212 6.9 Storage Classes 215 6.10 Scope Rules 218 6.11 Function Call Stack and Activation Records 221 6.12 Functions with Empty Parameter Lists 225 6.13 Inline Functions 225 6.14 References and Reference Parameters 227 6.15 Default Arguments 231 6.16 Unary Scope Resolution Operator 232 6.17 Function Overloading 234 6.18 Function Templates 236 6.19 Recursion 239 6.20 Example Using Recursion: Fibonacci Series 242 6.21 Recursion vs. Iteration 245 6.22 Wrap-Up 248

7 Arrays and Vectors 267

7.1 Introduction 268 7.2 Arrays 269 7.3 Declaring Arrays 270 7.4 Examples Using Arrays 271 7.4.1 Declaring an Array and Using a Loop to Initialize the Array's Elements 271 7.4.2 Initializing an Array in a Declaration with an Initializer List 272 7.4.3 Specifying an Array's Size with a Constant Variable and Setting Array Elements with Calculations 273 7.4.4 Summing the Elements of an Array 275 7.4.5 Using Bar Charts to Display Array Data Graphically 276 7.4.6 Using the Elements of an Array as Counters 277 7.4.7 Using Arrays to Summarize Survey Results 278 7.4.8 Static Local Arrays and Automatic Local Arrays 281

[x Contents](#)

7.5 Passing Arrays to Functions 283 7.6 Case Study: Class **GradeBook** Using an Array to Store Grades 287 7.7 Searching Arrays with Linear Search 293 7.8 Sorting Arrays with Insertion Sort 294 7.9 Multidimensional Arrays 297 7.10 Case Study: Class **GradeBook** Using a Two-Dimensional Array 300 7.11 Introduction to C++ Standard Library Class Template **vector** 307 7.12 Wrap-Up 313

8 Pointers 330

8.1 Introduction 331 8.2 Pointer Variable Declarations and Initialization 331 8.3 Pointer Operators 332 8.4 Pass-by-Reference with Pointers 335 8.5 Using **const** with Pointers 339 8.6 Selection Sort Using Pass-by-Reference 343 8.7 **sizeof** Operator 347 8.8 Pointer Expressions and

Pointer Arithmetic 349 8.9 Relationship Between Pointers and Arrays 352 8.10
Pointer-Based String Processing 354 8.11 Arrays of Pointers 357 8.12 Function
Pointers 358 8.13 Wrap-Up 361

9 Classes: A Deeper Look, Part 1 379 9.1 Introduction
380 9.2 Time Class Case Study 381 9.3 Class Scope and Accessing Class
Members 388 9.4 Separating Interface from Implementation 389 9.5 Access
Functions and Utility Functions 390 9.6 Time Class Case Study: Constructors
with Default Arguments 393 9.7 Destructors 398 9.8 When Constructors and
Destructors Are Called 399 9.9 Time Class Case Study: A Subtle
Trap—Returning a Reference to a
 private Data Member 402 9.10 Default Memberwise Assignment 405
9.11 Wrap-Up 407

10 Classes: A Deeper Look, Part 2 414 10.1
Introduction 415

[Contents xi](#)

10.2 **const** (Constant) Objects and **const** Member Functions 415 10.3
Composition: Objects as Members of Classes 423 10.4 **friend** Functions and
friend Classes 429 10.5 Using the **this** Pointer 431 10.6 **static** Class Members 436
10.7 Proxy Classes 441 10.8 Wrap-Up 445

11 Operator Overloading; Class string 451 11.1
Introduction 452 11.2 Using the Overloaded Operators of Standard Library
Class **string** 453 11.3 Fundamentals of Operator Overloading 456 11.4
Overloading Binary Operators 457 11.5 Overloading the Binary Stream
Insertion and Stream Extraction
 Operators 458 11.6 Overloading Unary Operators 462 11.7 Overloading
the Unary Prefix and Postfix **++** and **--** Operators 463 11.8 Case Study: A **Date**
Class 464 11.9 Dynamic Memory Management 469 11.10 Case Study: **Array**
Class 471
 11.10.1 Using the **Array** Class 472 11.10.2 **Array** Class Definition 475 11.11
 Operators as Member Functions vs. Non-Member Functions 483 11.12
Converting between Types 483 11.13 **explicit** Constructors 485 11.14 Building a
 String Class 487 11.15 Wrap-Up 488

12 Object-Oriented Programming: Inheritance 499 12.1 Introduction 500 12.2 Base Classes and Derived Classes 500 12.3
protected Members 503 12.4 Relationship between Base Classes and Derived
Classes 503
 12.4.1 Creating and Using a **CommissionEmployee** Class 504 12.4.2
 Creating a **BasePlusCommissionEmployee** Class Without Using
 Inheritance 508 12.4.3 Creating a **CommissionEmployee**–
 BasePlusCommissionEmployee Inheritance Hierarchy 514 12.4.4

12.4.5 CommissionEmployee–BasePlusCommissionEmployee Inheritance Hierarchy Using <code>private</code> Data	522
12.5 Constructors and Destructors in Derived Classes	527
12.6 <code>public</code> , <code>protected</code> and <code>private</code> Inheritance	527
12.7 Software Engineering with Inheritance	528
12.8 Wrap-Up	529

13 Object-Oriented Programming: Polymorphism

534 13.1 Introduction	535
13.2 Introduction to Polymorphism: Polymorphic Video Game	536
13.3 Relationships Among Objects in an Inheritance Hierarchy	536
13.3.1 Invoking Base-Class Functions from Derived-Class Objects	537
13.3.2 Aiming Derived-Class Pointers at Base-Class Objects	540
13.3.3 Derived-Class Member-Function Calls via Base-Class Pointers	541
13.3.4 Virtual Functions	543
13.4 Type Fields and <code>switch</code> Statements	549
13.5 Abstract Classes and Pure <code>virtual</code> Functions	549
13.6 Case Study: Payroll System Using Polymorphism	551
13.6.1 Creating Abstract Base Class <code>Employee</code>	552
13.6.2 Creating Concrete Derived Class <code>SalariedEmployee</code>	556
13.6.3 Creating Concrete Derived Class <code>CommissionEmployee</code>	558
13.6.4 Creating Indirect Concrete Derived Class <code>BasePlusCommissionEmployee</code>	560
13.6.5 Demonstrating Polymorphic Processing	562
13.7 (Optional) Polymorphism, Virtual Functions and Dynamic Binding “Under the Hood”	566
13.8 Case Study: Payroll System Using Polymorphism and Runtime Type Information with Downcasting, <code>dynamic_cast</code> , <code>typeid</code> and <code>typeid</code>	569
13.9 Virtual Destructors	573
13.10 Wrap-Up	573

14 Templates 579

14.1 Introduction	580
14.2 Function Templates	580
14.3 Overloading Function Templates	583
14.4 Class Templates	584
Contents	xiii

14.5 Nontype Parameters and Default Types for Class Templates	590
14.6 Wrap-Up	591

15 Stream Input/Output 595

15.1 Introduction	596
15.2 Streams	597
15.2.1 Classic Streams vs. Standard Streams	597
15.2.2 <code>iostream</code> Library Headers	598
15.2.3 Stream Input/Output Classes and Objects	598
15.3 Stream Output	601
15.3.1 Output of <code>char *</code> Variables	601
15.3.2 Character Output Using Member Function <code>put</code>	601
15.4 Stream Input	602
15.4.1 <code>get</code> and <code>getline</code> Member Functions	602
15.4.2 <code>istream</code> Member Functions <code>peek</code> , <code>putback</code> and <code>ignore</code>	605
15.4.3 Type-Safe I/O	605

15.5 Unformatted I/O Using `read`, `write` and `gcount` 605 15.6 Introduction to Stream Manipulators 606 15.6.1 Integral Stream Base: `dec`, `oct`, `hex` and `setbase` 607 15.6.2 Floating-Point Precision (`precision`, `setprecision`) 607 15.6.3 Field Width (`width`, `setw`) 609 15.6.4 User-Defined Output Stream Manipulators 610 15.7 Stream Format States and Stream Manipulators 612 15.7.1 Trailing Zeros and Decimal Points (`showpoint`) 612 15.7.2 Justification (`left`, `right` and `internal`) 613 15.7.3 Padding (`fill`, `setfill`) 615 15.7.4 Integral Stream Base (`dec`, `oct`, `hex`, `showbase`) 616 15.7.5 Floating-Point Numbers; Scientific and Fixed Notation (`scientific`, `fixed`) 617 15.7.6 Uppercase/Lowercase Control (`uppercase`) 618 15.7.7 Specifying Boolean Format (`boolalpha`) 618 15.7.8 Setting and Resetting the Format State via Member Function `flags` 619 15.8 Stream Error States 620 15.9 Tying an Output Stream to an Input Stream 622 15.10 Wrap-Up 623

16 Exception Handling: A Deeper Look 632 16.1

Introduction 633

xiv [Contents](#)

16.2 Example: Handling an Attempt to Divide by Zero 633 16.3 When to Use Exception Handling 639 16.4 Rethrowing an Exception 640 16.5 Exception Specifications 641 16.6 Processing Unexpected Exceptions 642 16.7 Stack Unwinding 642 16.8 Constructors, Destructors and Exception Handling 644 16.9 Exceptions and Inheritance 645 16.10 Processing `new` Failures 645 16.11 Class `unique_ptr` and Dynamic Memory Allocation 648 16.12 Standard Library Exception Hierarchy 650 16.13 Wrap-Up 652

17 File Processing 658 17.1

Introduction 659 17.2 Files and Streams 659 17.3 Creating a Sequential File 660 17.4 Reading Data from a Sequential File 664 17.5 Updating Sequential Files 669 17.6 Random-Access Files 670 17.7 Creating a Random-Access File 671 17.8 Writing Data Randomly to a Random-Access File 675 17.9 Reading from a Random-Access File Sequentially 677 17.10 Case Study: A Transaction-Processing Program 679 17.11 Object Serialization 686 17.12 Wrap-Up 686

18 Class `string` and String Stream Processing 696

18.1 Introduction 697 18.2 `string` Assignment and Concatenation 698 18.3 Comparing `strings` 700 18.4 Substrings 703 18.5 Swapping `strings` 703 18.6 `string` Characteristics 704 18.7 Finding Substrings and Characters in a `string` 706 18.8 Replacing Characters in a `string` 708 18.9 Inserting Characters into a `string` 710 18.10 Conversion to C-Style Pointer-Based `char *` Strings 711 18.11 Iterators 713

[Contents](#) xv

18.12 String Stream Processing 714 18.13 Wrap-Up 717

19 Searching and Sorting 724

19.1 Introduction 725 19.2 Searching Algorithms 725

19.2.1 Efficiency of Linear Search 726 19.2.2 Binary Search 727 19.3 Sorting Algorithms 732 19.3.1 Efficiency of Selection Sort 733 19.3.2 Efficiency of Insertion Sort 733 19.3.3 Merge Sort (A Recursive Implementation) 733 19.4 Wrap-Up 740

20 Custom Templated Data Structures 746

20.1 Introduction 747 20.2 Self-Referential Classes 748 20.3 Dynamic Memory Allocation and Data Structures 749 20.4 Linked Lists 749 20.5 Stacks 764 20.6 Queues 768 20.7 Trees 772 20.8 Wrap-Up 780

21 Bits, Characters, C Strings and structs 791

21.1 Introduction 792 21.2 Structure Definitions 792 21.3 `typedef` 794 21.4 Example: Card Shuffling and Dealing Simulation 794 21.5 Bitwise Operators 797 21.6 Bit Fields 806 21.7 Character-Handling Library 810 21.8 Pointer-Based String Manipulation Functions 815 21.9 Pointer-Based String-Conversion Functions 822 21.10 Search Functions of the Pointer-Based String-Handling Library 827 21.11 Memory Functions of the Pointer-Based String-Handling Library 831 21.12 Wrap-Up 835

[xvi Contents](#)

22 Standard Template Library (STL) 850

22.1 Introduction to the Standard Template Library (STL) 851 22.2 Introduction to Containers 853 22.3 Introduction to Iterators 856 22.4 Introduction to Algorithms 861 22.5 Sequence Containers 863

22.5.1 `vector` Sequence Container 864 22.5.2 `list` Sequence Container 871

22.5.3 `deque` Sequence Container 875

22.6 Associative Containers 877 22.6.1 `multiset` Associative Container 877 22.6.2

`set` Associative Container 880 22.6.3 `multimap` Associative Container 881

22.6.4 `map` Associative Container 883

22.7 Container Adapters 885 22.7.1 `stack` Adapter 885 22.7.2 `queue` Adapter 887

22.7.3 `priority_queue` Adapter 888

22.8 Algorithms 890 22.8.1 `fill`, `fill_n`, `generate` and `generate_n` 890 22.8.2 `equal`,

`mismatch` and `lexicographical_compare` 892 22.8.3 `remove`, `remove_if`,

`remove_copy` and `remove_copy_if` 895 22.8.4 `replace`, `replace_if`,

`replace_copy` and `replace_copy_if`

879

22.8.5 Mathematical Algorithms 900 22.8.6 Basic Searching and Sorting

Algorithms 903 22.8.7 `swap`, `iter_swap` and `swap_ranges` 905 22.8.8

`copy_backward`, `merge`, `unique` and `reverse` 906 22.8.9 `inplace_merge`,

`unique_copy` and `reverse_copy` 909 22.8.10 Set Operations 910 22.8.11

lower_bound, upper_bound and equal_range 913 22.8.12 Heapsort 915
22.8.13 min and max 918 22.8.14 STL Algorithms Not Covered in This
Chapter 919
22.9 Class `bitset` 920 22.10 Function Objects 924 22.11 Wrap-Up 927

23 Boost Libraries, Technical Report 1 and C++0x 936

23.1 Introduction 937

[Contents](#) xvii

23.2 Deitel Online C++ and Related Resource Centers 937 23.3 Boost Libraries 937
23.4 Boost Libraries Overview 938 23.5 Regular Expressions with the `regex` Library 941
23.5.1 Regular Expression Example 942 23.5.2 Validating User Input with Regular Expressions 944
23.5.3 Replacing and Splitting Strings 947
23.6 Smart Pointers 950 23.6.1 Reference Counted `shared_ptr` 950 23.6.2 `weak_ptr`: `shared_ptr` Observer 954
23.7 Technical Report 1 960 23.8 C++0x 961 23.9 Core Language Changes 962
23.10 Wrap-Up 967

24 Other Topics 974

24.1 Introduction 975 24.2 `const_cast` Operator 975 24.3 `mutable` Class Members 977 24.4 `namespaces` 979 24.5 Operator Keywords 982
24.6 Pointers to Class Members (`*` and `->*`) 984 24.7 Multiple Inheritance 986 24.8 Multiple Inheritance and `virtual` Base Classes 991
24.9 Wrap-Up 996

Chapters on the Web 1001

A Operator Precedence and Associativity 1002

B ASCII Character Set 1004

C Fundamental Types 1005

D Number Systems 1007

D.1 Introduction 1008
xviii [Contents](#)

D.2 Abbreviating Binary Numbers as Octal and Hexadecimal Numbers 1011 D.3 Converting Octal and Hexadecimal Numbers to Binary Numbers 1012 D.4 Converting from Binary, Octal or Hexadecimal to Decimal 1012 D.5 Converting from Decimal to Binary, Octal or Hexadecimal 1013 D.6 Negative Binary Numbers: Two's Complement Notation 1015

E Preprocessor 1020 E.1 Introduction 1021 E.2 `#include` Preprocessor Directive 1021 E.3 `#define` Preprocessor Directive: Symbolic Constants 1022 E.4 `#define` Preprocessor Directive: Macros 1022 E.5 Conditional Compilation 1024 E.6 `#error` and `#pragma` Preprocessor Directives 1025 E.7 Operators `#` and `##` 1026 E.8 Predefined Symbolic Constants 1026 E.9 Assertions 1027 E.10 Wrap-Up 1027

Appendices on the Web 1033 Index 1035

Chapters 25–26 and Appendices F–I 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 25.1

Introduction 25-2 25.2 Introduction to Object-Oriented Analysis and Design 25-2 25.3 Examining the ATM Requirements Document 25-3 25.4 Identifying the Classes in the ATM Requirements Document 25-10 25.5 Identifying Class Attributes 25-17 25.6 Identifying Objects' States and Activities 25-21 25.7 Identifying Class Operations 25-25 25.8 Indicating Collaboration Among Objects 25-32 25.9 Wrap-Up 25-39

[Contents](#) xix

26 ATM Case Study, Part 2:

Implementing an Object-Oriented Design 26-1

26.1 Introduction 26-2 26.2 Starting to Program the Classes of the ATM System 26-2 26.3 Incorporating Inheritance into the ATM System 26-8 26.4 ATM Case Study Implementation 26-15
26.4.1 Class **ATM** 26-16 26.4.2 Class **Screen** 26-23 26.4.3 Class **Keypad** 26-25
26.4.4 Class **CashDispenser** 26-26 26.4.5 Class **DepositSlot** 26-28 26.4.6 Class **Account** 26-29 26.4.7 Class **BankDatabase** 26-31 26.4.8 Class **Transaction** 26-35
26.4.9 Class **BalanceInquiry** 26-37 26.4.10 Class **Withdrawal** 26-39 26.4.11 Class **Deposit** 26-44 26.4.12 Test Program **ATMCaseStudy.cpp** 26-47 26.5 Wrap-Up 26-47

F C Legacy Code Topics F-1 F.1 Introduction F-2 F.2

Redirecting Input/Output on UNIX/Linux/Mac OS X

and Windows Systems F-2 F.3 Variable-Length Argument Lists F-3 F.4 Using Command-Line Arguments F-5 F.5 Notes on Compiling Multiple-Source-File Programs F-7 F.6 Program Termination with `exit` and `atexit` F-9 F.7 Type Qualifier `volatile` F-10 F.8 Suffixes for Integer and

Floating-Point Constants F-10 F.9 Signal Handling F-11 F.10 Dynamic Memory Allocation with `calloc` and `realloc` F-13 F.11 Unconditional Branch: `goto` F-14 F.12 Unions F-15 F.13 Linkage Specifications F-18 F.14 Wrap-Up F-19

G UML 2: Additional Diagram Types G-1 G.1

Introduction G-1 G.2 Additional Diagram Types G-2

xx [Contents](#)

H Using the Visual Studio Debugger H-1 H.1

Introduction H-2 H.2 Breakpoints and the Continue Command H-2 H.3 Locals and Watch Windows H-8 H.4 Controlling Execution Using the Step Into, Step Over, Step Out

and Continue Commands H-11 H.5 Autos Window H-13 H.6 Wrap-Up H-14

I Using the GNU C++ Debugger I-1 I.1 Introduction I-2 I.2

Breakpoints and the run, stop, continue and print Commands I-2 I.3 print and set Commands I-8 I.4 Controlling Execution Using the step, finish and

next Commands I-10 I.5 watch Command I-13 I.6 Wrap-Up I-15

“The chief merit of language is cleanness ...”
—Galen

For the Student

Welcome to the C++ computer programming language and C++ *How to Program, Eighth Edition*! This book presents leading-edge computing technologies, and is particularly appropriate for introductory course sequences based on the curriculum recommendations of two key professional organizations—the ACM and the IEEE.

The new Chapter 1 presents intriguing facts and figures. Our goal is to get you excited about studying computers and programming. The chapter includes a table of some of the research made possible by computers; current technology trends and hardware discussions; the data hierarchy; social networking; a table of business and technology publications and websites that will help you stay up-to-date with the latest technology news, trends and career opportunities; additional Making a Difference exercises and more.

We focus on software engineering best practices. At the heart of the book is our signature “live-code approach”—programming concepts are presented in the context of complete working programs, rather than in code snippets. Each C++ code example is accompanied by live sample executions, so you can see exactly what each program does when it’s run on a computer. All the source code is available at www.deitel.com/books/cpphttp8/ and www.pearsonhighered.com/deitel/.

Much of this Preface is addressed to instructors. Please be sure to read the sections entitled Pedagogic Features; Teaching Approach; Software Used in C++ *How to Program, 8/e*; C++ IDE Resource Kit and CourseSmart Web Books.

We believe that this book and its support materials will give you an informative, interesting, challenging and entertaining C++ educational experience. As you read the book, if you have questions, send an e-mail to deitel@deitel.com—we’ll respond promptly. For updates on this book, visit www.deitel.com/books/cpphttp8/, follow us on Facebook (www.deitel.com/deitelfan) and Twitter (@deitel), and subscribe to the *Deitel® Buzz* Online newsletter (www.deitel.com/newsletter/subscribe.html). Good luck!

New and Updated Features

Here are the updates we’ve made for C++ *How to Program*, 8/e.

Impending New C++ Standard

- **Optional sections.** We cover various features of the new standard (sometimes called C++0x and due late in 2011 or early in 2012) in *optional modular sections* and in Chapter 23. These are *easy to include or omit*. Popular compilers such as Microsoft Visual C++ 2010 and GNU C++ 4.5 already implement many of these features. To

xxii Preface

enable the new standard features in GNU C++, use the `-std=C++0x` flag when you compile the corresponding programs.

- **Boost C++ Libraries, Technical Report 1 (TR1) and C++0x.** In Chapter 23, we introduce the Boost C++ Libraries, Technical Report 1 (TR1) and C++0x. The free Boost open source libraries are created by members of the C++ community. Technical Report 1 describes the proposed changes to the C++ Standard Library, many of which are based on current Boost libraries. The C++ Standards Committee is revising the C++ Standard—the main goals are to make C++ easier to learn, improve library building capabilities, and increase compatibility with the C programming language. The new standard will include many of the libraries in TR1 and changes to the core language. We overview the Boost libraries and provide code examples for the “regular expression” and “smart pointer” libraries. Regular expressions are used to match specific character patterns in text. They can be used, for example, 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 programming capability you’ll study in Chapter 8. Smart pointers help you avoid errors by providing additional functionality to standard pointers.
- **unique_ptr vs. auto_ptr.** We replaced our `auto_ptr` example with the impending standard’s class `unique_ptr`, which fixes various problems that were associated with class `auto_ptr`. Use of `auto_ptr` is deprecated and `unique_ptr` is already implemented in many popular compilers, including Visual C++ 2010 and GNU C++ 4.5.
- **Initializer lists for user-defined types.** These enable objects of your own types to be initialized using the same syntax as built-in arrays.
- **Range-based for statement.** A version of the `for` statement that iterates over all the elements of an array or container (such as an object of the `vector` class).
- **Lambda expressions.** These enable you to create anonymous functions that can be passed to other functions as arguments.
- **auto storage class specifier.** The keyword `auto` can no longer be used as a storage

class specifier.

- **auto.** This keyword now deduces the type of a variable from its initializer.
- **nullptr.** This keyword is a replacement for assigning zero to a null pointer.
- **static_assert.** This capability allows you to test certain aspects of the program at compile time.
- **New long long and unsigned long long types.** These new types were introduced for use with 64-bit machines.

Pedagogic Features

- **Enhanced Making a Difference exercises set.** We encourage you to use computers and the Internet to research and solve significant social problems. These exercises are meant to increase awareness and discussion of important issues the world is facing. We hope you'll approach them with your own values, politics and beliefs.

New and Updated Features xxiii

Check out our new Making a Difference Resource Center at www.deitel.com/MakingADifference for additional ideas you may want to investigate further.

- **Page numbers for key terms in chapter summaries.** For key terms that appear in the chapter summaries, we include the page number of each term's defining occurrence in the chapter.
- **VideoNotes.** The Companion Website includes 15+ hours of VideoNotes in which co-author Paul Deitel explains in detail most of the programs in the core chapters. Instructors have told us that their students find the VideoNotes valuable for preparing for and reviewing lectures.
- **Modular presentation.** We've grouped the chapters into teaching modules. The Chapter Dependency Chart (later in this Preface) reflects the modularization.

Object Technology

- **Object-oriented programming and design.** We introduce the basic concepts and terminology of object technology in Chapter 1. Students develop their first customized classes and objects in Chapter 3. Presenting objects and classes early gets students "thinking about objects" immediately and mastering these concepts more thoroughly. [For courses that require a late-objects approach, consider *C++ How to Program, Late Objects Version, Seventh Edition*, 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.]
- **Integrated case studies.** We provide several case studies that span multiple sections and chapters. These include development of the **GradeBook** class in Chapters 3–7, the **Time** class in Chapters 9–10, the **Employee** class in Chapters 12–13, and the optional OOD/UML ATM case study in Chapters 25–26.
- **Integrated GradeBook case study.** The **GradeBook** case study uses classes and objects in Chapters 3–7 to incrementally build a **GradeBook** class that represents an instructor's grade book and performs various calculations based on a set of student grades, such as calculating the average grade, finding the maximum and minimum, and printing a bar chart.

- **Exception handling.** We integrate basic exception handling early in the book. Instructors can easily pull more detailed material forward from Chapter 16, Exception Handling: A Deeper Look.
- **Prefer vectors to C arrays.** C++ offers two types of arrays—vector class objects (which we start using in Chapter 7) and C-style, pointer-based arrays. As appropriate, we use class template `vector` instead of `Carrays` throughout the book. However, we begin by discussing C arrays in Chapter 7 to prepare you for working with legacy code and to use as a basis for building your own customized Array class in Chapter 11.
- **Prefer string objects to C strings.** Similarly, C++ offers two types of strings—string class objects (which we use starting in Chapter 3) and C-style, pointer-based strings. We continue to include some early discussions of C strings to give

xxiv Preface

you practice with pointer manipulations, to illustrate dynamic memory allocation with `new` and `delete` and to prepare you for working with C strings in the legacy code that you'll encounter in industry. In new development, you should favor string class objects. We've replaced most occurrences of C strings with instances of C++ class `string` to make programs more robust and eliminate many of the security problems that can be caused by using C strings.

- **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 industry-standard graphical language for modeling object-oriented systems. Chapters 25–26 include an *optional* online case study on object-oriented design using the UML. We design and implement 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 specify 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 having a “light-bulb moment”—the case study helps them “tie it all together” and really understand object orientation.
- **Standard Template Library (STL).** This might be one of the most important topics in the book in terms of your appreciation of software reuse. The STL defines powerful, template-based, reusable components that implement many common data structures and algorithms used to process those data structures. Chapter 22 introduces the STL and discusses its three key components—containers, iterators and algorithms. The STL components provide tremendous expressive power, often reducing many lines of code to a single statement.

Other Features

- **Printed book contains core content; additional chapters are online.** Several online chapters are included for more advanced courses and for professionals. These are available in searchable PDF format on the book's password-protected Companion Website—see the access card in the front of this book.
- **Reorganized Chapter 11, Operator Overloading: Class string.** We reorganized

this chapter to begin with standard library class `string` so readers can see an elegant use of operator overloading before they implement their own. We also moved the section on proxy classes to the end of Chapter 10, where it's a more natural fit.

- **Enhanced use of `const`.** We increased the use of `const` book-wide to encourage better software engineering.
- **Software engineering concepts.** Chapter 1 briefly introduces very current software engineering terminology, including agile software development, Web 2.0, Ajax, SaaS (Software as a Service), PaaS (Platform as a Service), cloud computing, web services, open source software, design patterns, refactoring, LAMP and more.
- **Compilation and linking process for multiple-source-file programs.** Chapter 3 includes a detailed diagram and discussion of the compilation and linking process that produces an executable program.

Our Text + Digital Approach to Content xxv

- **Function Call Stack Explanation.** In Chapter 6, we provide a detailed discussion with illustrations of the function call stack and activation records to explain how C++ is able to keep track of which function is currently executing, how automatic variables of functions are maintained in memory and how a function knows where to return after it completes execution.
- **Tuned Treatment of Inheritance and Polymorphism.** Chapters 12–13 have been carefully tuned using a concise `Employee` class hierarchy. We use this same treatment in our C++, Java, C# and Visual Basic books—one of our reviewers called it the best he had seen in 25 years as a trainer and consultant.
- **Discussion and illustration of how polymorphism works “under the hood.”** Chapter 13 contains a detailed diagram and explanation of how C++ can implement polymorphism, virtual functions and dynamic binding internally. This gives students a solid understanding of how these capabilities work.
- **ISO/IEC C++ standard compliance.** We’ve audited our presentation against the ISO/IEC C++ standard document.
- **Debugger appendices.** We provide two Using the Debugger appendices on the book’s Companion Website—Appendix H, Using the Visual Studio Debugger, and Appendix I, Using the GNU C++ Debugger.
- **Code tested on multiple platforms.** We tested the code examples on various popular C++ platforms including GNU C++ on Linux and Microsoft Windows, and Visual C++ on Windows. For the most part, the book’s examples port to popular standard-compliant compilers.
- **Game Programming.** Because of limited interest, we’ve removed from the book Chapter 27, Game Programming with Ogre (which covers only Linux). For instructors who would like to continue using this material with C++ *How to Program*, 8/e, we’ve included the version from C++ *How to Program*, 7/e on the book’s Companion Website.

Our Text + Digital Approach to Content

We surveyed hundreds of instructors teaching C++ courses and learned that most want a book with content focused on their introductory courses. With that in mind, we

moved various advanced chapters to the web. Having this content in digital format makes it easily searchable, and gives us the ability to fix errata and add new content as appropriate. The book's Companion Website, which is accessible at

www.pearsonhighered.com/deitel/

(see the access card at the front of the book) contains the following chapters in *searchable* PDF format:

- 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
- Game Programming with Ogre (from *C++ How to Program*, 7/e)
- Appendix F, C Legacy Code Topics

xxvi Preface

- Appendix G, UML 2: Additional Diagram Types
- Appendix H, Using the Visual Studio Debugger
- Appendix I, Using the GNU C++ Debugger

The Companion Website also includes:

- Extensive VideoNotes—watch and listen as co-author Paul Deitel discusses the key features of the code examples in Chapters 2–13 and portions of Chapters 16 and 17.
- Two true/false questions per section with answers for self-review.
- Solutions to approximately half of the solved exercises in the book.

The following materials are posted at the Companion Website and at www.deitel.com/books/cpphttp8/:

- An array of function pointers example and additional function pointer exercises (from Chapter 8).
- String Class Operator Overloading Case Study (from Chapter 11).
- Building Your Own Compiler exercise descriptions (from Chapter 20).

Dependency Chart

The chart on the next page shows the dependencies among the chapters to help instructors plan their syllabi. *C++ How to Program*, 8/e is appropriate for CS1 and CS2 courses.

Teaching Approach

C++ How to Program, 8/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 the context of *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, 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 coloring 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 each program's key code segments.

Using fonts for emphasis. We place the key terms and the index's page reference for each defining occurrence 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, `int x = 5;`).

Chapter Dependency Chart

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

Introduction

1 Introduction to
Computers and C++

Intro to Programming, Classes and Objects

2 Intro to C++ Programming

3 Intro to Classes and Objects

Control Statements, Methods and Arrays

4 Control Statements: Part 1

5 Control Statements: Part 2

6 Functions and an
Intro to Recursion

7 Arrays and Vectors

Legacy C Topics

Teaching Approach xxvii

21 Bits, Characters, 8 Pointers C-Strings and
structs

Object-Oriented Programming

Input/Output¹

OOP: Polymorphism

9 Classes: A
Deeper Look,
Part 1

14 Templates

10 Classes: A
Deeper Look,
Part 2

16 Exception
Handling: A
Deeper Look

11 Operator

Overloading 12

OOP: Inheritance 13

Object-Oriented Design with the UML

25 (Optional)

Object-Oriented Design
with the UML

26 (Optional)

Implementing an
Object-Oriented Design

Streams, Files and Strings

15 Stream

Data Structures

19 Searching and	20 Custom	22 Standard Template Library
17 File	Templatized Data	
Processing	Chapter 7. A small	
18 Class	portion requires	
string and String	Chapters 12 and 14.	
Stream Processing	23 Boost Libraries,	
	Technical Report 1	
	and C++0x	
Other Topics	24 Other Topics	
	1. Most of Chapter 15	
	is readable after	

xxviii [Preface](#)

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

Illustrations/ figures. Abundant tables, line drawings, UML diagrams, programs and program outputs are included.

Programming tips. We include programming tips to help you focus on important aspects of program development. These tips and practices represent the best we've gleaned from a combined seven decades of programming and teaching experience.

Good Programming Practices

The Good Programming Practices call attention to techniques that will help you produce 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.

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 into programs 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-scale systems.

Summary bullets. We present a section-by-section bullet-list summary of the chapter with the page references to the defining occurrence for many of the key terms in each section.

Self-review exercises and answers. Extensive self-review exercises and answers are included for self study. All of the exercises in the optional ATM case study are fully solved.

Exercises. Each chapter concludes with a substantial set of exercises including:

- simple recall of important terminology and concepts

- What's wrong with this code?
- What does this code do?
- writing individual statements and small portions of functions and classes •
- writing complete functions, classes and programs
- major projects.

Please do not write to us requesting access to the Pearson Instructor's Resource Center which contains the book's instructor supplements, including the exercise solutions. Ac

Software Used in C++ *How to Program*, 8/e xxix

cess is limited strictly to college instructors teaching from the book. Instructors may obtain access only through their Pearson representatives. Solutions are *not* provided for “project” exercises. Check out our Programming Projects Resource Center for lots of additional exercise and project possibilities (www.deitel.com/ProgrammingProjects/).

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

Software Used in C++ *How to Program*, 8/e

We wrote *C++ How to Program*, 8/e using Microsoft's free Visual C++ Express Edition (which is available free for download at www.microsoft.com/express/downloads/) and the free GNU C++ (gcc.gnu.org/install/binaries.html), which is already installed on most Linux systems and can be installed on Mac OS X and Windows systems. Apple includes GNU C++ in their Xcode development tools, which Mac OS X users can download from developer.apple.com/technologies/tools/xcode.html.

C++ IDE Resource Kit

Your instructor may have ordered through your college bookstore a Value Pack edition of *C++ How to Program*, 8/e that comes bundled with the C++ IDE Resource Kit. This kit contains CD or DVD versions of:

- Microsoft[®] Visual Studio 2010 Express Edition (www.microsoft.com/express/) •
- Dev C++ (www.bloodshed.net/download.html)
- NetBeans (netbeans.org/downloads/index.html)
- Eclipse (eclipse.org/downloads/)
- CodeLite (codelite.org/LiteEditor/Download)

You can download these software packages from the websites specified above. The C++ IDE Resource Kit also includes access to a Companion Website containing step-by-step written instructions and VideoNotes to help you get started with each development environment. If your book did not come with the C++ IDE Resource Kit, you can purchase access to the Resource Kit's Companion Website from www.pearsonhighered.com/cppidekit/.

CourseSmart Web Books

Today's students and instructors have increasing demands on their time and money. Pearson has responded to that need by offering digital texts and course materials online through CourseSmart. CourseSmart allows faculty to review course materials online, say

ing time and costs. It offers students a high-quality digital version of the text for less than the cost of a print copy of the text. Students receive the same content offered in the print textbook enhanced by search, note-taking, and printing tools. For more information, visit www.coursesmart.com.

Instructor Supplements

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

xxx Preface

- *Solutions Manual* with solutions to the vast majority of the end-of-chapter exercises and Lab Manual exercises. We've added dozens of Making a Difference exercises, most with solutions.
- *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

If you're not already a registered faculty member, contact your Pearson representative or visit www.pearsonhighered.com/educator/relocator/.

Acknowledgments2

We'd like to thank Abbey Deitel and Barbara Deitel of Deitel & Associates, Inc. for long hours devoted to this project. We're fortunate to have worked with the dedicated team of publishing professionals at Pearson. We appreciate the guidance, savvy and energy of Michael Hirsch, Editor-in-Chief of Computer Science. Carole Snyder recruited the book's reviewers and managed the review process. Bob Engelhardt managed the book's production.

Reviewers

We wish to acknowledge the efforts of our seventh and eighth edition reviewers. They scrutinized the text and the programs and provided countless suggestions for improving the presentation: Virginia Bailey (Jackson State University), Thomas J. Borrelli (Rochester Institute of Technology), 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), Wing-Ning Li (University of Arkansas), Dean Mathias (Utah State University), Robert A. McLain (Tidewater Community College), April Reagan (Microsoft), José Antonio González Seco (Parliament of Andalusia, Spain), Dave Topham (Ohlone College) and Anthony Williams (author and C++ Standards Committee member).

Well, there you have it! As you read the book, we would sincerely appreciate your comments, criticisms, corrections and suggestions for improving the text. Please address all correspondence to:

deitel@deitel.com

We'll respond promptly. We hope you enjoy working with *C++ How to Program, Eighth Edition* as much as we enjoyed writing it!

Paul and Harvey Deitel

About the Authors

Paul J. 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 C++, Java, C#, Visual Basic, C and Internet programming courses to industry clients, including Cisco, IBM, Siemens, Sun Microsystems, Dell, Lu

[About Deitel & Associates, Inc.](#) xxxi

cent Technologies, Fidelity, NASA at the Kennedy Space Center, the National Severe Storm Laboratory, White Sands Missile Range, Rogue Wave Software, Boeing, SunGard Higher Education, Stratus, Cambridge Technology Partners, One Wave, Hyperion Software, Adra Systems, Entergy, CableData Systems, Nortel Networks, Puma, iRobot, Invensys and many more. He and his co-author, Dr. Harvey M. Deitel, are the world's best selling programming-language textbook authors.

Dr. Harvey M. 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 from MIT in Electrical Engineering and a Ph.D. in Mathematics from Boston University—at both he studied computing before separate computer science degree programs were created. 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., with his son, Paul J. Deitel. He and Paul are the co-authors of dozens of books and LiveLessons multimedia packages. With translations published in Japanese, German, Russian, Chinese, Spanish, Korean, French, Polish, Italian, Portuguese, Greek, Urdu and Turkish, the Deitels' texts have earned international recognition. Dr. Deitel has delivered hundreds of professional programming language seminars to major corporations, academic institutions, government organizations and the military.

About Deitel & Associates, Inc.

Deitel & Associates, Inc., is an internationally recognized corporate training and authoring organization specializing in computer programming languages, Internet and web software technology, object-technology and Android™ and iPhone® education and applications development. The company provides instructor-led courses delivered at client sites worldwide on major programming languages and platforms, such as C++, Visual C++®, C, Java™, Visual C#®, Visual Basic®, XML®, Python®, object technology, Internet and web programming, Android and iPhone app development, and a growing list of additional programming and software-development courses. The founders of Deitel & Associates, Inc., are Paul J. Deitel and Dr. Harvey M. Deitel. The company's clients include many of the world's largest corporations, government agencies, branches of the military, and academic institutions. Through its 35-year publishing partnership with Prentice Hall/ Pearson Higher Education, Deitel&Associates, Inc., publishes leading-edge programming textbooks, professional books, interactive multimedia *Cyber Classrooms*, and *LiveLessons* DVD-based and web-based video courses. Deitel&Associates, Inc., and the authors can be reached via e-mail at:

deitel@deitel.com

To learn more about Deitel&Associates, Inc., its publications and its *Dive Into*® Series

Corporate Training curriculum delivered at client locations worldwide, visit:

www.deitel.com/training/

subscribe to the free *Deitel[®] Buzz Online* e-mail newsletter at:

www.deitel.com/newsletter/subscribe.html

and follow the authors on Facebook (www.deitel.com/deitelfan) and Twitter (@deitel).
xxxii Preface

Individuals wishing to purchase Deitel books, and *LiveLessons* DVD and web-based training courses 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

Introduction to Computers and C++

*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

In this chapter you'll learn:

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

2 Chapter 1 Introduction to Computers and C++

Introduction to Object Technology

1.7 Operating Systems

1.8 Programming Languages

1.9 C++ and a Typical C++
Development Environment

1.10 Test-Driving a C++
Application 1.11 Web 2.0:
Going Social

1.12 Software Technologies

1.1 Introduction

1.2 Computers: Hardware and
Software 1.3 Data Hierarchy

1.4 Computer Organization

1.5 Machine Languages,
Assembly Languages and
High-Level Languages 1.6

Self-Review Exercises | Answers to Self-Review Exercises | Exercises | Making a Difference | Making a Difference Resources

1.1 Introduction

Welcome to C++—a powerful computer programming language that’s appropriate for technically oriented people with little or no programming experience, and for experienced programmers to use in building substantial information systems. You’re already familiar with the powerful tasks computers perform. Using this textbook, you’ll write instructions commanding computers to perform those kinds of tasks. *Software* (i.e., the instructions you write) controls *hardware* (i.e., computers).

You’ll learn *object-oriented programming*—today’s key programming methodology. You’ll create and work with many *software objects* in this text.

C++ is one of today’s most popular software development languages. This text provides an introduction to programming in the version of C++ standardized in the United States through the **American National Standards Institute (ANSI)** and worldwide through the efforts of the **International Organization for Standardization (ISO)**.

In use today are more than a billion general-purpose computers and billions more cell phones, smartphones and handheld devices (such as tablet computers). According to a study by eMarketer, the number of mobile Internet users will reach approximately 134 million by 2013.¹ Other studies have projected smartphone sales to surpass personal computer sales in 2011² and tablet sales to account for over 20% of all personal computer sales by 2015.³ By 2014, the smartphone applications market is expected to exceed \$40 billion,⁴ which is creating significant opportunities for programming mobile applications.

Computing in Industry and Research

These are exciting times in the computer field. Many of the most influential and successful businesses of the last two decades are technology companies, including Apple, IBM, Hew

1. www.circleid.com/posts/mobile_internet_users_to_reach_134_million_by_2013/. 2. www.pcworld.com/article/171380/more_smartphones_than_desktop_pcs_by_2011.html. 3. www.forrester.com/ER/Press/Release/0,1769,1340,00.html. 4. *Inc.*, December 2010/January 2011, pages 116–123.

lett Packard, Dell, Intel, Motorola, Cisco, Microsoft, Google, Amazon, Facebook, Twitter, Groupon, Foursquare, Yahoo!, eBay and many more—these are major employers of people who study computer science, information systems or related disciplines. At the time of this writing, Apple was the second most valuable company in the world and *the* most valuable technology company.⁵ Computers are also used extensively in academic and industrial research. Figure 1.1 provides just a few examples of exciting ways in which

computers are used in research and industry.

Internet The Internet—a global network of computers—was made possible by the *convergence of computing and communications*. It has its roots in the 1960s, when research funding was supplied by the U.S. Department of Defense. Originally designed to connect the main computer systems of about a dozen universities and research organizations, the Internet today is accessible by billions of computers and computer-controlled devices worldwide.

Computers break lengthy transmissions into packets at the sending end, route the packets to their intended receivers and ensure that those packets are received in sequence and without error at the receiving end. According to a study by Forrester Research, the average U.S. online consumer now spends as much time online as watching television (forrester.com/rb/Research/understanding_changing_needs_of_us_online_consumer,/q/id/57861/t/2).

Human Genome Project	tremendous innovation and growth in the biotechnology industry.
World Community Grid	World Community Grid (www.worldcommunitygrid.org) is a non-profit computing grid. People worldwide donate their unused computer processing power by installing a free secure software program that allows the World Community Grid to harness the excess power when the computers are idle. The computing power is used in place of supercomputers to conduct scientific research projects that are making a difference, including developing affordable solar energy, providing clean water to the developing world, fighting cancer, curing muscular dystrophy, finding influenza antiviral drugs, growing more nutritious rice for regions fighting hunger and more. 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-ray, optimizing the scan for each type of tissue, then to combine all of the information to create a 3D image.
Medical imaging	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 to researchers in many fields. This research has led to

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

5. www.zdnet.com/blog/apple/apple-becomes-worlds-second-most-valuable-company/9047.

4 Chapter 1 Introduction to Computers and C++

GPS Global Positioning System (GPS) devices use a network of satellites to retrieve location-based information. Multiple satellites send time-stamped signals to the device GPS device, which calculates the distance to each satellite based on the time the signal left the satellite and the time the signal was received. The location of each satellite and the distance to each are used to determine the exact location of the device. Based on your location, GPS

devices can provide step by-step directions, help you easily find nearby businesses (restaurants, gas stations, etc.) and points of interest, or help you find your friends.	
Microsoft's SYNC®	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	AMBER Alert recently partnered with Facebook. Facebook users can "Like" AMBER Alert pages by location to receive alerts in their news feeds.
Many Ford cars now feature Microsoft's SYNC technology, providing speech synthesis (for reading text messages to you) and speech-recognition capabilities that allow you to use voice commands to browse music, request traffic alerts and more.	
The AMBER (America's Missing: Broadcast	
Robots	Robots are computerized machines that can perform tasks (including physical tasks), respond to stimuli and more. They can be used for day-to-day tasks (e.g., iRobot's Roomba vacuum), entertainment (such as robotic pets), military combat, space and deep sea exploration, manufacturing and more. In 2004, NASA's remote-controlled Mars rover—which used Java technology—explored the surface to learn about the history of water on the planet.
One Laptop Per Child (OLPC)	North America and the U.K. (news.cnet.com/8301-13772_3-10396593-52.html?tag=mncol;txt)! Online <i>social gaming</i> , which enables users worldwide to compete with one another, is growing rapidly. Zynga—creator of popular online games such as <i>Farmville</i> and <i>Mafia Wars</i> —was founded in 2007 and already has over 215 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-peta-byte-of-data-daily-adds-1000-servers-a-week/)!
Game programming	Video game consoles are also becoming increasingly sophisticated. The Wii Remote uses an <i>accelerometer</i> (to detect tilt and acceleration) and a sensor that determines where the device is pointing, allowing the device to respond to motion. By gesturing with the Wii Remote in hand, you can control the video game on the screen.
One Laptop Per Child (OLPC) is providing low-power, inexpensive, Internet-enabled laptops to poor children worldwide—enabling learning and reducing the digital divide (one.laptop.org). By providing these educational resources, OLPC is increasing the opportunities for poor children to learn and make a difference in their communities.	
The computer game business is larger than the first-run movie business. The most sophisticated video games can cost as much as \$100 million to develop. Activision's <i>Call of Duty 2: Modern Warfare</i> , released in November 2009, earned \$310 million in just one day in	

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

1.2 Computers: Hardware and Software 5

(cont.) With Microsoft's Kinect for Xbox 360, you—the player—become the controller.

Kinect uses a camera, depth sensor and sophisticated software to follow your body movement, allowing you to control the game (en.wikipedia.org/wiki/Kinect). Kinect games include dancing, exercising, playing sports, training virtual animals and more.

Internet TV Internet TV set-top boxes (such as Apple TV and Google TV) give you access to content—such as games, news, movies, television shows and more—allowing you to access an enormous amount of content on demand; you no longer need to rely on cable or satellite television providers to get content.

1.2 Computers: Hardware and Software

A computer is a device that can perform computations 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! To put that in perspective, a quadrillion-instruction-per-second computer can perform in one second more than 100,000 calculations *for every person on the planet!* And—these “upper limits” are growing quickly!

Computers process data under the control of sets of instructions called **computer programs**. These programs guide the computer through orderly sets of actions specified by people called computer **programmers**. The programs that run on a computer are referred to as **software**. In this book, you'll learn today's 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, DVDs 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—it's an ingredient in common sand. Silicon-chip technology has made computing so economical that more than a billion general-purpose computers are in use worldwide, and this is expected to *double* in the next few years.

Computer chips (*microprocessors*) control countless devices. These **embedded systems** include anti-lock brakes in cars, navigation systems, smart home appliances, home security systems, cell phones and smartphones, robots, intelligent traffic intersections, collision avoidance systems, video game controllers and more. The vast majority of the microprocessors produced each year are embedded in devices other than general-purpose computers.⁶

6. www.eetimes.com/electronics-blogs/industrial-control-designline-blog/4027479/Real-men-program-in-C?pageNumber=1.

6 Chapter 1 Introduction to Computers and C++

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* without any increase in price. This remarkable observation often is called **Moore's Law**, named for the person who identified the trend, Gordon Moore, co founder of Intel—a leading manufacturer of the processors in today's computers and embedded systems. Moore's Law and related observations are especially true in relation to the amount of memory that computers have for programs, the amount of secondary storage (such as disk storage) they have to hold programs and data over longer periods of time, and their processor speeds—the speeds at which computers execute their programs (i.e., do their work). Similar 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 improves so quickly and costs fall so rapidly. Such phenomenal improvement is truly fostering the *Information Revolution*.

1.3 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 1.2 illustrates a portion of the data hierarchy. Figure 1.3 summarizes the data hierarchy's levels.

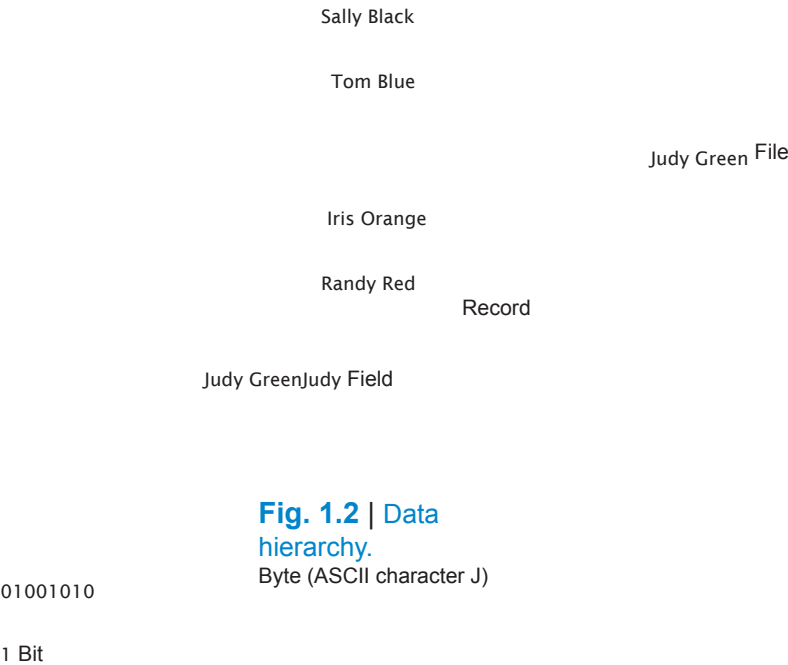


Fig. 1.2 | Data hierarchy.
Byte (ASCII character J)

1.4 Computer Organization 7

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 0s and 1s—*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 1s and 0s, so a computer’s character set represents every character as a pattern of 1s and 0s. C++ uses the **ASCII (American**

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 can be used to represent a person's name, and a field consisting of decimal digits could represent a person's age.

Records Several related fields can be used to compose a **record** (implemented as a **class** in Java). 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.

Fig. 1.3 | Levels of the data hierarchy.

1.4 Computer Organization

Regardless of differences in physical appearance, computers can be envisioned as divided into various **logical units** or sections (Fig. 1.4).

8 Chapter 1 Introduction to Computers and C++

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 speaking to your computer, scanning images and barcodes, reading from secondary storage devices (like 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 download videos from YouTube™ or e-books from Amazon). Newer forms of input include reading position data from a GPS device, and motion and orientation information from an accelerometer in a smartphone or game controller.

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 displayed on screens, printed on paper, played as audio or video on portable 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.

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 <i>volatile</i> —it’s typically lost when the computer’s power is turned off. The memory unit is often called either memory or primary memory . Typical main memories on desktop and notebook computers contain between 1 GB and 8 GB (GB stands for gigabytes; a gigabyte is approximately one billion bytes).
Arithmetic and logic unit (ALU)	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 <i>dual-core processor</i> has two CPUs and a <i>quad core processor</i> has four CPUs. Today’s desktop computers have processors that can execute billions of instructions per second.
Central processing unit (CPU)	This is the long-term, high-capacity “warehousing” section. Programs or data not actively being used by the other units normally are placed on secondary
Secondary storage unit	This “manufacturing” section performs <i>calculations</i> , such as addition, subtraction, multiplication and division. It also contains the <i>decision</i> mechanisms that allow the computer, for example, to compare two items from the memory unit to determine whether they’re equal. In today’s systems, the ALU is usually implemented as part of the next logical unit, the CPU.

Fig. 1.4 | Logical units of a computer. (Part 1 of 2.)

1.5 Machine Languages, Assembly Languages and High-Level

Languages 9

Secondary storage unit (cont.) storage devices (e.g., your <i>hard drive</i>) until they’re again needed, possibly hours, days, months or even years later. Information on secondary storage devices is <i>persistent</i> —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 128 GB. Typical hard drives on desktop and notebook computers can hold up to 2 TB (TB stands for terabytes; a terabyte is approximately one trillion bytes).
--	---

Fig. 1.4 | Logical units of a computer. (Part 2 of 2.)

1.5 Machine Languages, Assembly Languages and High-Level Level

Languages

Programmers write instructions in various programming languages, some directly understandable by computers and others requiring intermediate *translation* steps. Hundreds of such languages are in use today. These may be divided into three general types:

1. Machine languages
2. Assembly languages
3. High-level languages

Any computer can directly understand only its own **machine language**, defined by its hardware design. Machine languages generally consist of strings of numbers (ultimately reduced to 1s and 0s) that instruct computers to perform their most elementary operations one at a time. Machine languages are *machine dependent* (a particular machine language can be used on only one type of computer). Such languages are cumbersome for humans. For example, here's a section of an early machine-language program that adds overtime pay to base pay and stores the result in gross pay:

```
+1300042774
+1400593419
+1200274027
```

Programming in machine language was simply too slow and tedious for most programmers. Instead of using the strings of numbers that computers could directly understand, programmers began using English-like abbreviations to represent elementary operations. These abbreviations formed the basis of **assembly languages**. *Translator programs* called **assemblers** were developed to convert early assembly-language programs to machine language at computer speeds. The following section of an assembly-language program also adds overtime pay to base pay and stores the result in gross pay:

```
load basepay
add overpay
store grosspay
```

10 Chapter 1 Introduction to Computers and C++

Although such code is clearer to humans, it's incomprehensible to computers until translated to machine language.

Computer usage increased rapidly with the advent of assembly languages, but programmers still had to use many instructions to accomplish even the simplest tasks. To speed the programming process, **high-level languages** were developed in which single statements could be written to accomplish substantial tasks. Translator programs called **compilers** convert high-level language programs into machine language. High-level languages allow you to write instructions that look almost like everyday English and contain commonly used mathematical notations. A payroll program written in a high-level language might contain a *single* statement such as

```
grossPay = basePay + overTimePay
```

From the programmer's standpoint, high-level languages are preferable to machine and assembly languages. C++, C, Microsoft's .NET languages (e.g., Visual Basic, Visual C++ and Visual C#) and Java are among the most widely used high-level programming languages.

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 delay of compilation), although slower than compiled programs run.

1.6 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 are discovering that using a modular, object-oriented design and implementation approach can make software-development groups much more productive than was possible with earlier popular techniques like “structured programming”—object-oriented programs are often easier to understand, correct and modify.

The Automobile as an Object

To help you understand objects and their contents, 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.

Just as you cannot cook meals in the kitchen of a blueprint, you cannot drive a car’s engineering drawings. 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

1.6 Introduction to Object Technology 11

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 carexample to introduce some key object-oriented programming concepts. Performing a task in a program requires a **member function**, which houses the program statements that actually perform its task. The member function 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* of a class before a program can perform the tasks that the class’s member functions 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.



Software Engineering Observation 1.1

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

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

12 Chapter 1 Introduction to Computers and C++

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 implemented—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” certainly *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 software system to control thousands of automated teller machines for a major bank? Or suppose you were asked to work on a team of 1,000 software developers building the next U.S. air traffic control system? For projects so large and complex, you should not simply 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 oriented. 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

1.7 Operating Systems 13

Chapter 13. 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.

1.7 Operating Systems

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

Windows—A Proprietary Operating System

In the mid-1980s, Microsoft developed the **Windows operating system**, consisting of a graphical user interface built on top of DOS—an enormously popular personal-computer operating system of the time that users interacted with by typing commands. Windows borrowed from many concepts (such as icons, menus and windows) popularized by early Apple Macintosh operating systems and originally developed by Xerox PARC. Windows 7 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 multi-touch support, and more.

Windows is a *proprietary* operating system—it's controlled by one company exclusively. Windows is by far the world's most widely used operating system.

Linux—An Open-Source Operating System

The Linux operating system is perhaps the greatest success of the *open-source* movement. **Open-source software** is a software development style that departs from the *proprietary* development that dominated software's early years. With open-source development, individuals and companies contribute their efforts in developing, maintaining and evolving software in exchange for the right to use that software for their own purposes, typically at no charge. Open-source code is often scrutinized by a much larger audience than proprietary software, so errors often get removed faster. Open source also encourages more innovation.

Some organizations in the open-source community are the Eclipse Foundation (the Eclipse Integrated Development Environment helps C++ 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 the tools for managing open source projects—it has over 260,000 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 few decades ago. A great example is Facebook, which was launched from a college dorm room and built with open-source software.⁷

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

7. developers.facebook.com/opensource/.

14 Chapter 1 Introduction to Computers and C++

popular in servers, personal computers and embedded systems. Unlike that of proprietary operating systems like Microsoft's Windows and Apple's Mac OS X, Linux source code (the program code) is available to the public for examination and modification and is free to download and install. As a result, users of the operating system benefit from a community of developers actively debugging and improving the kernel, an absence of licensing fees and restrictions, and the ability to completely customize the operating system to meet specific needs.

In 1991, Linus Torvalds, a 21-year-old student at the University of Helsinki, Finland, began developing the Linux kernel as a hobby. (The name Linux is derived from "Linus" and "UNIX"—an operating system developed by Bell Labs in 1969.) Torvalds wished to improve upon the design of Minix, an educational operating system created by Professor Andrew Tanenbaum of the Vrije Universiteit in Amsterdam. The Minix source code was publicly available to allow professors to demonstrate basic operating-system implementation concepts to their students.

Torvalds released the first version of Linux in 1991. The favorable response led to the creation of a community that has continued to develop and support Linux. Developers downloaded, tested, and modified the Linux code, submitting bug fixes and feedback to Torvalds, who reviewed them and applied the improvements to the code.

The 1994 release of Linux included many features commonly found in a mature operating system, making Linux a viable alternative to UNIX. Enterprise systems companies such as IBM and Oracle became increasingly interested in Linux as it continued to stabilize and spread to new platforms.

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. But Linux has become extremely popular on servers and in embedded systems, such as Google’s Android-based smartphones.

Android

Android—the fastest growing mobile and smartphone operating system—is based on the Linux kernel and Java. One benefit of developing Android apps is the openness of the platform. The operating system is open source and free.

The Android operating system 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 79 by 2010—was formed to continue developing Android. As of December 2010, more than 300,000 Android smartphones were being activated each day!⁸ Android smartphones are now outselling iPhones.⁹ The Android operating system is used in numerous smartphones (such as the Motorola Droid, HTC EVO™ 4G, Samsung Vibrant™ and many more), e-reader devices (such as the Barnes and Noble Nook™), tablet computers (such as the Dell Streak, the Samsung Galaxy Tab and more), in-store touch-screen kiosks, cars, robots and multimedia players.

Android smartphones include the functionality of a mobile phone, Internet client (for web browsing and Internet communication), MP3 player, gaming console, digital camera

8. www.pcmag.com/article2/0,2817,2374076,00.asp.

9. mashable.com/2010/08/02/android-outselling-iphone-2/.

1.8 Programming Languages 15

and more, wrapped into handheld devices with full-color *multitouch screens*—these allow you to control the device with *gestures* involving one touch or multiple simultaneous touches. You can download apps directly onto your Android device through Android Market and other app marketplaces. As of December 2010, there were over 200,000 apps in Google’s Android Market.

1.8 Programming Languages

In this section, we provide brief comments on several popular programming languages (Fig. 1.5). In the next section we introduce C++.

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

COBOL COBOL (COMmon Business Oriented Language) was developed 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 disciplined 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 by Professor 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). Its latest version supports object-oriented programming.

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

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

16 Chapter 1 Introduction to Computers and C++

C C was implemented in 1972 by Dennis Ritchie at Bell Laboratories. 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 Mac OS X operating system and all iOS-powered devices (such as iPods, iPhones and iPads).

Java Sun Microsystems in 1991 funded an internal corporate research project 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 variety 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, television set-top boxes and more) and for many other purposes.

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

Visual C# Microsoft's three object-oriented primary programming languages are Visual Basic (based on the original Basic), Visual C++ (based on C++) and C# (based on C++ and Java, and developed for inte

- grating the Internet and the web into computer applications).
- PHP PHP is an object-oriented, “open-source” (see Section 1.7) “script ing” 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.

Fig. 1.5 | Other programming languages. (Part 2 of 3.)
1.9 C++ and a Typical C++ Development Environment

17

- JavaScript JavaScript is the most widely used scripting language. It’s primarily used to add programmability to web pages—for example, animations and interactivity with the user. It’s provided with all major web browsers.
- 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* (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 | Other programming languages. (Part 3 of 3.)

1.9 C++ and a Typical C++ Development Environment

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*.

C99 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. C99 also makes C more consistent with C++. For more information on C and C99, see our book *C How to Program, 6/e* and our C Resource Center (located at www.deitel.com/C).

C++, an extension of C, was developed by Bjarne Stroustrup in the early 1980s at Bell Laboratories. C++ provides a number of features that “spruce up” the C language, but more importantly, it provides capabilities for object-oriented programming.

18 Chapter 1 Introduction to Computers and C++

You'll begin developing customized, reusable classes and objects in Chapter 3, Introduction 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 1.2

*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 1.3

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-consuming and complex effort that goes into designing, developing and maintaining new functions and classes that are correct and that operate efficiently.

Performance Tip 1.1

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

Portability Tip 1.1

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

We now explain the commonly used steps in creating and executing a C++ application using a C++ development environment (illustrated in Figs. 1.6–1.11). 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,

1.9 C++ and a Typical C++ Development Environment 19

compile, link, load and execute. The following discussion explains a typical C++ program development environment.

Phase 1: Creating 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.

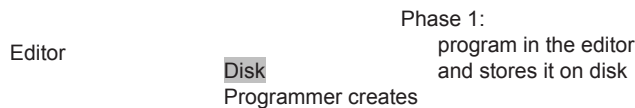


Fig. 1.6 | Typical C++ development environment—editing phase.

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 2010 Express Edition, Dev C++, NetBeans, Eclipse 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 **preprocessor directives**, which indicate that certain manipulations are to be performed on the program before compilation. These manipulations usually include other text files to be compiled, and perform various text replacements. The most common preprocessor directives are discussed in the early chapters; a detailed discussion of preprocessor features appears in Appendix E, Preprocessor.

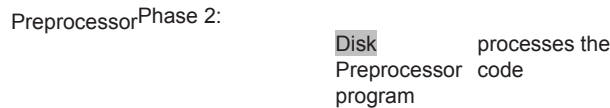


Fig. 1.7 | Typical C++ development environment—preprocessor phase.
20 Chapter 1 Introduction to Computers and C++

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).

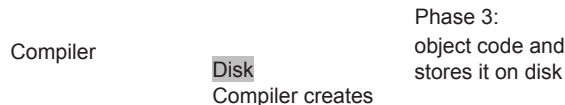


Fig. 1.8 | Typical C++ development environment—compilation phase.

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 programmers 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.

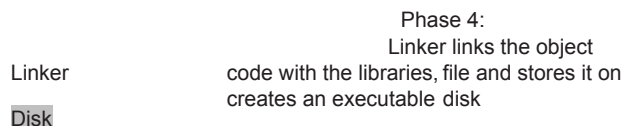


Fig. 1.9 | Typical C++ development environment—linking phase.

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.

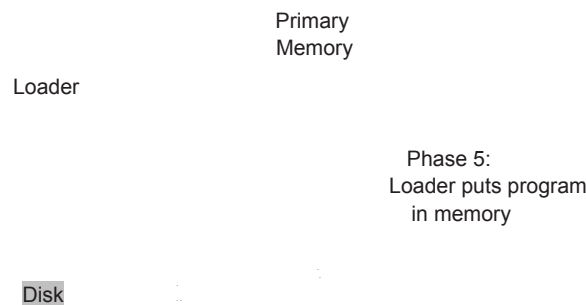


Fig. 1.10 | Typical C++ development environment—loading phase.

1.10 Test-Driving a C++ Application 21

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.

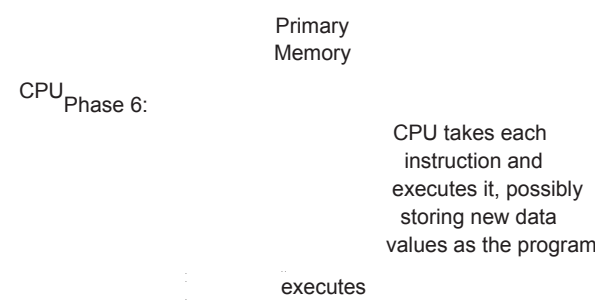


Fig. 1.11 | Typical C++ development environment—execution phase.

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 whole-number 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 hardcopy 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 running an entertaining guess-the-number game, which picks a number from 1 to 1000 and

22 Chapter 1 Introduction to Computers and C++

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 development environments are available in which you can compile, build and run C++ applications, such as GNU C++, Dev C++, Microsoft Visual C++, CodeLite, NetBeans, Eclipse etc. Consult your instructor for information on your specific development 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 to emphasize filenames, text displayed by an application and values you should enter into an application (e.g., **Guess Number** 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 **Properties**. 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/cpphttp8/ 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.



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

1.10 Test-Driving a C++ Application 23

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).

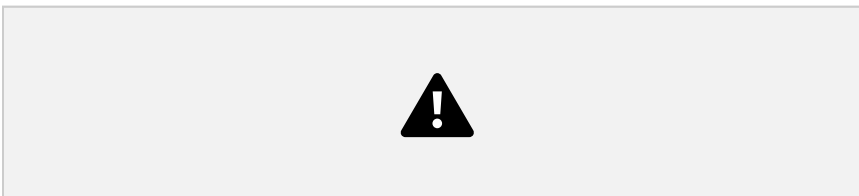


Fig. 1.14 | Entering your first guess.

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).

24 **Chapter 1 Introduction to Computers and C++**

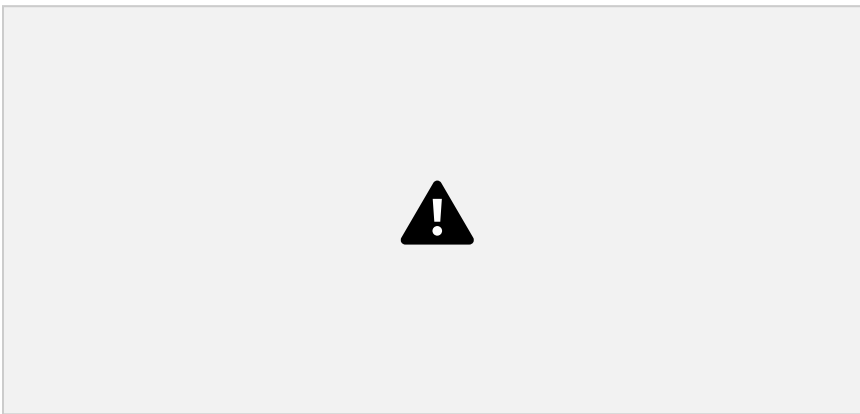


Fig. 1.16 | Entering additional guesses and guessing the correct number.

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 executable 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 application that we modified and test-drove for Windows, but the outputs could vary based on the compiler being used.]

26 Chapter 1 Introduction to Computers and C++

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 (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 Web 2.0: Going Social

The web literally exploded in the mid-to-late 1990s, but the “dot com” economic bust brought hard times in the early 2000s. The resurgence that began in 2004 or so has been named **Web 2.0**. Google is widely regarded as the signature company of Web 2.0. Some other companies with “Web 2.0 characteristics” are YouTube (video sharing), FaceBook (social networking), Twitter (microblogging), Groupon (social commerce), Foursquare (mobile check-in), Salesforce (business software offered as online services), Craigslist (free classified listings), Flickr (photo sharing), Second Life (a virtual world), Skype (Internet telephony) and Wikipedia (a free online encyclopedia).

Google

In 1996, Stanford computer science Ph.D. candidates Larry Page and Sergey Brin began collaborating on a new search engine. In 1997, they changed the name to Google—a play on the mathematical term *googol*, a quantity represented by the number “one” followed by 100 “zeros” (or 10^{100})—a staggeringly large number. Google’s ability to return extremely accurate search results quickly helped it become the most widely used search engine and one of the most popular websites in the world.

Google continues to be an innovator in search technologies. For example, Google Goggles is a fascinating mobile app (available on Android and iPhone) that allows you to

perform a Google search using a photo rather than entering text. You simply take pictures of a landmarks, books (covers or barcodes), logos, art or wine bottle labels, and Google Goggles scans the photo and returns search results. You can also take a picture of text (for example, a restaurant menu or a sign) and Google Goggles will translate it for you.

Ajax

Ajax is one of the premier Web 2.0 software technologies. Ajax helps Internet-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 server computers on the Internet. Using Ajax, applications like Google Maps have achieved excellent performance and approach the look-and-feel of desktop applications.

Social Applications

Over the last several years, there's been a tremendous increase in the number of social applications on the web. Even though the computer industry is mature, these sites were still able to become phenomenally successful in a relatively short period of time. Figure 1.27 discusses a few of the social applications that are making an impact.

Facebook Facebook was launched from a Harvard dorm room in 2004 by classmates Mark Zuckerberg, Chris Hughes, Dustin Moskovitz and Eduardo Saverin and is already worth an estimated \$70 billion. By January 2011, Facebook was the most active site on the Internet with more than 600 million users—nearly 9% of the Earth's population—who spend 700 billion minutes on Facebook per month (www.time.com/time/specials/packages/article/0,28804,2036683_2037183,00.html). At its current rate of growth (about 5% per month), Facebook will reach one billion users in 2012, out of the two billion people on the Internet! The activity on the site makes it extremely attractive for application developers. Each day, over 20 million applications are installed by Facebook users (www.facebook.com/press/info.php?statistics). Twitter Twitter was founded in 2006 by Jack Dorsey, Evan Williams and Isaac “Biz” Stone—all from the podcast company, Odeo. Twitter has revolutionized *microblogging*. Users post tweets—messages of up to 140 characters in length. Approximately 95 million tweets are posted per day (twitter.com/about). You can follow the tweets of friends, celebrities, businesses, government representatives (including the U.S. President, who has 6.3 million followers), etc., or you can follow tweets by subject to track news, trends and more. At the time of this writing, Lady Gaga had the most followers (over 7.7 million). Twitter has become the point of origin for many breaking news stories worldwide.

Groupon Groupon, a *social commerce* site, was launched by Andrew Mason in 2008. By January 2011, the company was valued around \$15 billion, making it the fastest growing company ever! It's now available in hundreds of markets world wide. Groupon offers one daily deal in each market for restaurants, retailers, services, attractions and more. Deals are activated only after a minimum number of people sign up to buy the product or service. If you sign up for a deal

Fig. 1.27 | Social applications. (Part 1 of 2.)

and it has yet to meet the minimum, you might be inclined to tell others about the deal by email, Facebook, Twitter, etc. If the deal does not meet the minimum sales, it's cancelled. One of the most successful national Groupon deals to date was a certificate for \$50 worth of merchandise from a major apparel company for \$25. Over 440,000 vouchers were sold in one day.

1.12 Software Technologies 29

Groupon (cont.)

Foursquare Foursquare—launched in 2009 by Dennis Crowley and Naveen Selvadurai—is a mobile *check-in* application that allows you to notify your friends of your whereabouts. You can download the app to your smartphone and link it to your Facebook and Twitter accounts so your friends can follow you from multiple platforms. If you do not have a smartphone, you can check in by text message. Foursquare uses GPS to determine your exact location. Businesses use Foursquare to send offers to users in the area. Launched in March 2009, Foursquare already has over 5 million users worldwide.

Skype Skype is a software product that allows you to make mostly free voice and video calls over the Internet using a technology called *VoIP* (*Voice over IP*; IP stands for “Internet Protocol”). Skype was founded in 2003 by Niklas Zennström and Dane Janus Friis. Just two years later, the company was sold to eBay for \$2.6 billion.

YouTube YouTube is a video-sharing site that was founded in 2005. Within one year, the company was purchased by Google for \$1.65 billion. YouTube now accounts for 10% of all Internet traffic (www.webpronews.com/topnews/2010/04/16/facebook-and-youtube-get-the-most-business-internet-traffic). Within one week of the release of Apple's iPhone 3GS—the first iPhone model to offer video—mobile uploads to YouTube grew 400% (www.hypebot.com/hypebot/2009/06/youtube-reports-1700-jump-in-mobile-video.html).

Fig. 1.27 | Social applications. (Part 2 of 2.)

1.12 Software Technologies

Figure 1.28 lists a number of buzzwords that you'll hear in the software development community. We've created Resource Centers on most of these topics, with more on the way.

Agile software development

Agile software development is a set of methodologies that try to get software

implemented faster and using fewer resources than previous methodologies. 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. 1.28 | Software technologies. (Part 1 of 2.)
30 Chapter 1 Introduction to Computers and C++

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 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. **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).

Software as a Service (SaaS)

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 capabilities—no local installation is needed. You access the service through a browser. Browsers are quite portable, so you can run the same applications on a wide variety of computers from anywhere in the world. Salesforce.com, Google, and Microsoft’s Office Live and Windows Live all offer SaaS.

Platform as a Service (PaaS)

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

Cloud computing

SaaS and PaaS are examples of **cloud computing** in which software, platforms and infrastructure (e.g., processing power and storage) are hosted on demand over the Internet. This provides users with flexibility, scalability and cost savings. For example, consider a company’s data storage needs which can fluctuate significantly over the course of a year. Rather than investing in large-scale storage hardware—which can be costly to purchase, maintain and secure, and would most likely not be used to capacity at all times—the company could purchase cloud-based services (such as Amazon S3, Google Storage, Microsoft Windows Azure™, Nirvanix™ and others) dynamically as needed.

Software

Development 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 cumbersome for organizations with

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

Libraries 31 Figure 1.29 describes software product release categories.

Alpha An *alpha* version of software is the earliest release of a software product that's still under active development. Alpha versions are often buggy, incomplete and unstable and are released to a relatively small number of developers for testing new features, getting early feedback, etc.

Beta *Beta* versions are released to a larger number of developers later in the development process after most major bugs have been fixed and new features are nearly complete. Beta software is more stable, but still subject to change.

Release candidates 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.

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

Fig. 1.29 | Software product release terminology.

1.13 Future of C++: TR1, the New C++ Standard and the Open Source Boost Libraries

Bjarne Stroustrup, the creator of C++, has expressed his vision for the future of C++. The main goals for the new standard are to make C++ easier to learn, improve library building capabilities, and increase compatibility with the C programming language.

Throughout the book, we discuss in optional sections various key features of the new C++ standard. In addition, Chapter 23 introduces the Boost C++ Libraries, Technical Report 1 (TR1) and more new C++ features.

Technical Report 1 describes the proposed changes to the C++ Standard Library. These libraries add useful functionality to C++. The C++ Standards Committee is currently finishing the revision of the C++ Standard. The last standard was published in 1998. Work on the new standard began in 2003. At that time, it was referred to as **C++0x** because the standard was scheduled to be released before the end of the decade. The new standard includes most of the libraries in TR1 and changes to the core language.

The **Boost C++ Libraries** are free, open-source libraries created by members of the C++ community. 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++ programmers 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. Several of the Boost libraries are included in TR1 and will be part of the new standard. We overview the libraries included

32 Chapter 1 Introduction to Computers and C++

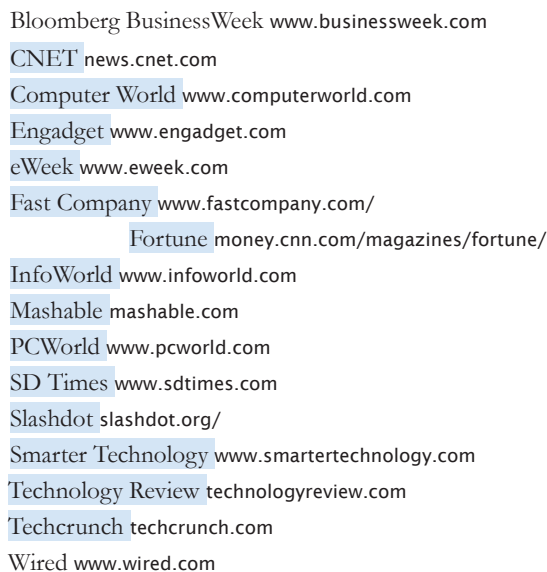
in TR1 and provide code examples for the “regular expression” and “smart pointer” libraries.

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 programming capability C++ absorbed from C. **Smart pointers** help you avoid errors by providing additional functionality, typically strengthening the process of memory allocation and deallocation.

1.14 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.



Bloomberg BusinessWeek www.businessweek.com
CNET news.cnet.com
Computer World www.computerworld.com
Engadget www.engadget.com
eWeek www.eweek.com
Fast Company www.fastcompany.com/
Fortune money.cnn.com/magazines/fortune/
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 (many are free).

1.15 Wrap-Up

In this chapter we discussed computer hardware, software, programming languages and operating systems. We introduced the basics of object technology. You learned about some of the exciting recent developments in the computer field. We overviewed a typical C++ program development environment and you test-drove a C++ application. We also discussed some key software development terminology.

In Chapter 2, you'll create your first C++ applications. You'll see how programs display messages on the screen and obtain information from the user at the keyboard for processing. You'll see several examples that demonstrate how programs display messages on the screen and obtain information from the user at the keyboard for processing.

Self-Review Exercises

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

- a) The company that popularized personal computing was .
- b) The computer that made personal computing legitimate in business and industry was the .
- c) Computers process data under the control of sets of instructions called .
- d) The key logical units of the computer are the , , , and .
- e) The three types of languages discussed in the chapter are , and .
- f) The programs that translate high-level language programs into machine language are called .
- g) is a smartphone operating system based on the Linux kernel and Java.
- h) software is generally feature complete and (supposedly) bug free and ready for use by the community.
- i) The Wii Remote, as well as many smartphones, uses a(n) which allows the device 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 functions 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 1.6):

- a) Objects have the property of encapsulation—although objects may know how to communicate 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 classes, 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 object-oriented analysis and design.
- d) With inheritance, new classes of objects are derived by absorbing characteristics of existing classes, then adding unique characteristics of their own.
- e) UML 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) Apple. b) IBM Personal Computer. c) programs. d) input unit, output unit, memory unit, central processing unit, arithmetic and logic unit, secondary storage unit. e) machine languages, assembly languages, high-level languages. f) compilers. g) Android. h) Release candidate. i) accelerometer.

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

1.3 a) encapsulation. b) classes. c) object-oriented analysis and design. d) inheritance. e) UML. f) The size, shape, color and weight of an object are considered of the object's class.

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 machine-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) is used to develop large-scale enterprise applications, to enhance the functionality of web servers, to provide applications for consumer devices and for many other purposes.
- b) initially became widely known as the development language of the Unix operating system.
- c) The Web 2.0 company is the fastest growing company ever.
- d) 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—`lex`, `parse`, `semantic`, `codegen`, `link`, and `load`.
- 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.
- c) The command `java` invokes the , which executes Java programs. d) A(n) is a software application that simulates a computer, but hides the underlying operating system and hardware from the programs that interact with it. e) The takes the `.class` files containing the program's bytecodes and transfers them to primary memory.
- f) The examines bytecodes to ensure that they're valid.

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:

Making a Difference 35

object, attributes, behaviors, class, inheritance (consider, for example, an alarm clock), abstraction, 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. 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) Many 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

36 Chapter 1 Introduction to Computers and C++

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 excessive 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 the projects developed by previous winners, visit www.imaginecup.com/about. You can also find several project ideas submitted by worldwide charitable organizations at www.imaginecup.com/students/imagine-cup-solve-this. 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 hunger, 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 Ventilation System, T.H.U.M.S. (Total Human Model for Safety) and Touch Tracer Display. You can participate 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

*What's in a name? that
which we call a rose
By any other name*

would smell as sweet.

—William Shakespeare

When faced with a decision, I always ask, "What would be the most fun?"

—Peggy Walker

High thoughts must have high language.

—Aristophanes

One person can make a difference and every person should try.

—John F. Kennedy

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.

Objectives

38 Chapter 2 Introduction to C++ Programming

Adding Integers

2.5 Memory Concepts

2.6 Arithmetic

2.7 Decision Making: Equality and Relational Operators

2.8 Wrap-Up

2.1 Introduction

2.2 First Program in C++: Printing a Line of Text

2.3 Modifying Our First C++

Program 2.4 Another C++ Program:

Summary | Self-Review Exercises | Answers to Self-Review Exercises | Exercises | Making a Difference

2.1 Introduction

We now introduce C++ programming, which facilitates a disciplined approach to program design. Most of the C++ programs you'll study in this book process information and display results. In this chapter, we present five examples that demonstrate how your programs can display messages and obtain information from the user for processing. The first three examples simply display messages on the screen. The next obtains two numbers from a user, calculates their sum and displays the result. The accompanying discussion shows you how to perform arithmetic calculations and save their results for later use. The fifth example demonstrates decision-making by showing you how to compare two numbers, then display messages based on the comparison results. We analyze each program one line at a time to help you ease your way into C++ programming.

2.2 First Program in C++: Printing a Line of Text

C++ uses notations that may appear strange to nonprogrammers. We now consider a simple program that prints a line of text (Fig. 2.1). This program illustrates several important features of the C++ language.

```
1 // Fig. 2.1: fig02_01.cpp
2 // Text-printing program.
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"; // display message 9
```

```
10 return 0; // indicate that program ended successfully 11 } // end
function main
```

Welcome to C++!

Fig. 2.1 | Text-printing program.

Comments

Lines 1 and 2

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

2.2 First Program in C++: Printing a Line of Text 39

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 terminates at the end of the current line. [Note: You also may use C’s style in which a comment—possibly containing many lines—begins with `/*` and ends with `*/`.]

Good Programming Practice 2.1

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

#include Preprocessor Directive

Line 3

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

is a **preprocessor 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 must be included for any program that outputs data to the screen or inputs 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 15.

Common Programming Error 2.1

Forgetting to include the `<iostream>` header in a program that inputs data from the keyboard 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 in the program. The keyword `int` to

40 Chapter 2 Introduction to C++ Programming

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++ key words can be found in Fig. 4.3. We'll explain what it means for a function to “return 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 **string** of characters contained between the double quotation marks. A string is sometimes called a **character string** or a **string literal**. We refer to characters between double quotation marks simply as strings. White-space characters 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**. Every C++ statement must end with a semicolon (also known as the **statement terminator**). Preprocessor directives (like `#include`) do not end with a semicolon. 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 within the 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 program by the preprocessor directive `#include <iostream>`. The notation `std::cout` spec

2.2 First Program in C++: Printing a Line of Text 41

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 24, 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—in Fig. 2.13, we introduce 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

The `<<` operator is referred to as the **stream insertion operator**. When this program executes, the value to 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. The right operand'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.

<code>\n</code>	Newline. Position the screen cursor to the beginning of the next line.
<code>\t</code>	Horizontal tab. Move the screen cursor to the next tab stop.
<code>\r</code>	Carriage return. Position the screen cursor to the beginning of the current line; do not advance to the next line.
<code>\a</code>	Alert. Sound the system bell.
<code>\\</code>	Backslash. Used to print a backslash character.
<code>\'</code>	Single quote. Use to print a single quote character.
<code>\"</code>	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 successfully*. 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 `return` 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.

42 Chapter 2 Introduction to C++ Programming

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 yellow 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
9
10 } // end function main
```

```
std::cout << "Welcome ";
```

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

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> //
4 allows program to output data to the screen
5 // function main begins program execution
6 int main()
7 {
8 std::cout << "Welcome to C++!\n\n";
9 } // end
function main
```

Welcome
to

C++!





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

2.4 Another C++ Program: Adding Integers

Our next program uses the input stream object `std::cin` and the stream extraction operator, `>>`, to obtain 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.

```
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; // first integer to add
int number2; // second integer to add
int sum; // sum of number1 and number2
```

```
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
```

```
18
19
```

```
sum = number1 + number2; // add the numbers; store result in sum
```

```
20
21 std::cout << "Sum is " << sum << ; // display sum; end line
22 } // end function main
```

Enter first integer: 45 Enter
second integer: 72 Sum is 117

std::endl

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

The comments in lines 1 and 2 state the name of the file and the purpose of the program. The C++ preprocessor directive in line 3 includes the contents of the `<iostream>` header. 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.

Variable Declarations

Lines 9–11

```
int number1; // first integer to add
int number2; // second integer to add
int sum; // sum of number1 and number2
```

are **declarations**. The identifiers `number1`, `number2` and `sum` are the names of **variables**. A variable is a location in the computer's memory where a value can be stored for use by a program. These declarations specify that the variables `number1`, `number2` and `sum` are data of type `int`, meaning that these variables will hold **integer** values, i.e., whole numbers such

44 Chapter 2 Introduction to C++ Programming

as 7, -11, 0 and 31914. All variables *must* be declared with a *name* and a *data type before* they can be used in a program. Several variables of the same type may be declared in one declaration or in multiple declarations. We could have declared all three variables in one declaration by using a **comma-separated list** as follows:

```
int number1, number2, sum;
```

This makes the program less readable and prevents us from providing comments that describe each variable's purpose.

Good Programming Practice 2.4

Place a space after each comma (,) to make programs more readable.

Fundamental Types

We'll soon discuss the type `double` for specifying real numbers, and the type `char` for specifying character data. Real numbers are numbers with decimal points, such as 3.4, 0.0 and -11.19. A `char` variable may hold only a single lowercase letter, a single uppercase letter, a single digit or a single special character (e.g., \$ or *). Types such as `int`, `double` and `char` are called **fundamental types**. Fundamental-type names are *keywords* and therefore *must* appear in all lowercase letters. Appendix C contains the complete list of fundamental types.

Identifiers

A variable name (such as `number1`) is any valid **identifier** that is not a keyword. An identifier is a series of characters consisting of letters, digits and underscores (`_`) that does not begin with a digit. C++ is **case sensitive**—uppercase and lowercase letters are different, so `a1` and `A1` are different identifiers.

Portability Tip 2.1

C++ allows identifiers of any length, but your C++ implementation may restrict identifier lengths. Use identifiers of 31 characters or fewer to ensure portability.

Good Programming Practice 2.5

Choosing meaningful identifiers makes a program **self-documenting**—a person can understand the program simply by reading it rather than having to refer to manuals or comments.

Good Programming Practice 2.6

Avoid using abbreviations in identifiers. This improves program readability.

Good Programming Practice 2.7

Do not use identifiers that begin with underscores and double underscores, because C++ compilers may use names like that for their own purposes internally. This will prevent the names you choose from being confused with names the compilers choose.

Placement of Variable Declarations

Declarations of variables can be placed almost anywhere in a program, but they must appear *before* their corresponding variables are used in the program. For example, in the program of Fig. 2.5, the declaration in line 9

2.4 Another C++ Program: Adding Integers 45

```
int number1; // first integer to add
```

could have been placed immediately before line 14

```
std::cin >> number1; // read first integer from user into number1 the
```

declaration in line 10

```
int number2; // second integer to add
```

could have been placed immediately before line 17

```
std::cin >> number2; // read second integer from user into number2 and the
```

declaration in line 11

```
int sum; // sum of number1 and number2
```

could have been placed immediately before line 19

```
sum = number1+number2; // add the numbers; store result in sum
```

Good Programming Practice 2.8

Always place a blank line between a declaration and adjacent executable statements. This makes the declarations stand out and contributes to program clarity.

Obtaining the First Value from the User

Line 13

```
std::cout << "Enter first integer: "; // prompt user for data
```

displays Enter first integer: followed by a space. This message is called a **prompt** because it directs the user to take a specific action. We like to pronounce the preceding statement as “std::cout gets the character string “Enter first integer: .” Line 14

```
std::cin >> number1; // read first integer from user into number1
```

uses the **standard input stream object cin** (of namespace std) and the **stream extraction operator, >>**, to obtain a value from the keyboard. Using the stream

extraction operator with `std::cin` takes character input from the standard input stream, which is usually the keyboard. We like to pronounce the preceding statement as, “`std::cin` gives a value to `number1`” or simply “`std::cin` gives `number1`.”

When the computer executes the preceding statement, it waits for the user to enter a value for variable `number1`. The user responds by typing an integer (as characters), then pressing the *Enter* key (sometimes called the *Return* key) to send the characters to the computer. The computer converts the character representation of the number to an integer and assigns (i.e., copies) this number (or **value**) to the variable `number1`. Any subsequent references to `number1` in this program will use this same value.

The `std::cout` and `std::cin` stream objects facilitate interaction between the user and the computer. Because this interaction resembles a dialog, it’s often called **interactive computing**.

Obtaining the Second Value from the User

Line 16

```
std::cout << "Enter second integer: "; // prompt user for data
```

46 Chapter 2 Introduction to C++ Programming

prints Enter second integer: on the screen, prompting the user to take action. Line 17

```
std::cin >> number2; // read second integer from user into number2 obtains a
```

value for variable `number2` from the user.

Calculating the Sum of the Values Input by the User

The assignment statement in line 19

```
sum = number1+number2; // add the numbers; store result in sum
```

adds the values of variables `number1` and `number2` and assigns the result to variable `sum` using the **assignment operator** `=`. The statement is read as, “`sum` gets the value of `number1 + number2`.” Most calculations are performed in assignment statements. The `=` operator and the `+` operator are called **binary operators** because each has two operands. In the case of the `+` operator, the two operands are `number1` and `number2`. In the case of the preceding `=` operator, the two operands are `sum` and the value of the expression `number1 + number2`.

Good Programming Practice 2.9

Place spaces on either side of a binary operator. This makes the operator stand out and makes the program more readable.

Displaying the Result

Line 21

```
std::cout << "Sum is " << sum << std::endl; // display sum; end line
```

displays the character string `Sum is` followed by the numerical value of variable `sum` followed by `std::endl`—a so-called **stream manipulator**. The name `endl` is an abbreviation for “end line” and belongs to namespace `std`. The `std::endl` stream manipulator outputs a newline, then “flushes the output buffer.” This simply means that, on some systems where outputs accumulate in the machine until there are enough to “make it worthwhile” to display them on the screen, `std::endl` forces any accumulated outputs to be displayed at that moment. This can be important when the outputs are prompting the user for an action, such as entering data.

The preceding statement outputs multiple values of different types. The stream insertion operator “knows” how to output each type of data. Using multiple stream insertion operators (<<) in a single statement is referred to as **concatenating**, **chaining** or **cascading stream insertion operations**. It’s unnecessary to have multiple statements to output multiple pieces of data.

Calculations can also be performed in output statements. We could have combined the statements in lines 19 and 21 into the statement

```
std::cout << "Sum is " << number1 + number2 << std::endl;
```

thus eliminating the need for the variable `sum`.

A powerful feature of C++ is that you can create your own data types called classes (we introduce this capability in Chapter 3 and explore it in depth in Chapters 9 and 10). You can then “teach” C++ how to input and output values of these new data types using the >> and << operators (this is called **operator overloading**—a topic we explore in Chapter 11).