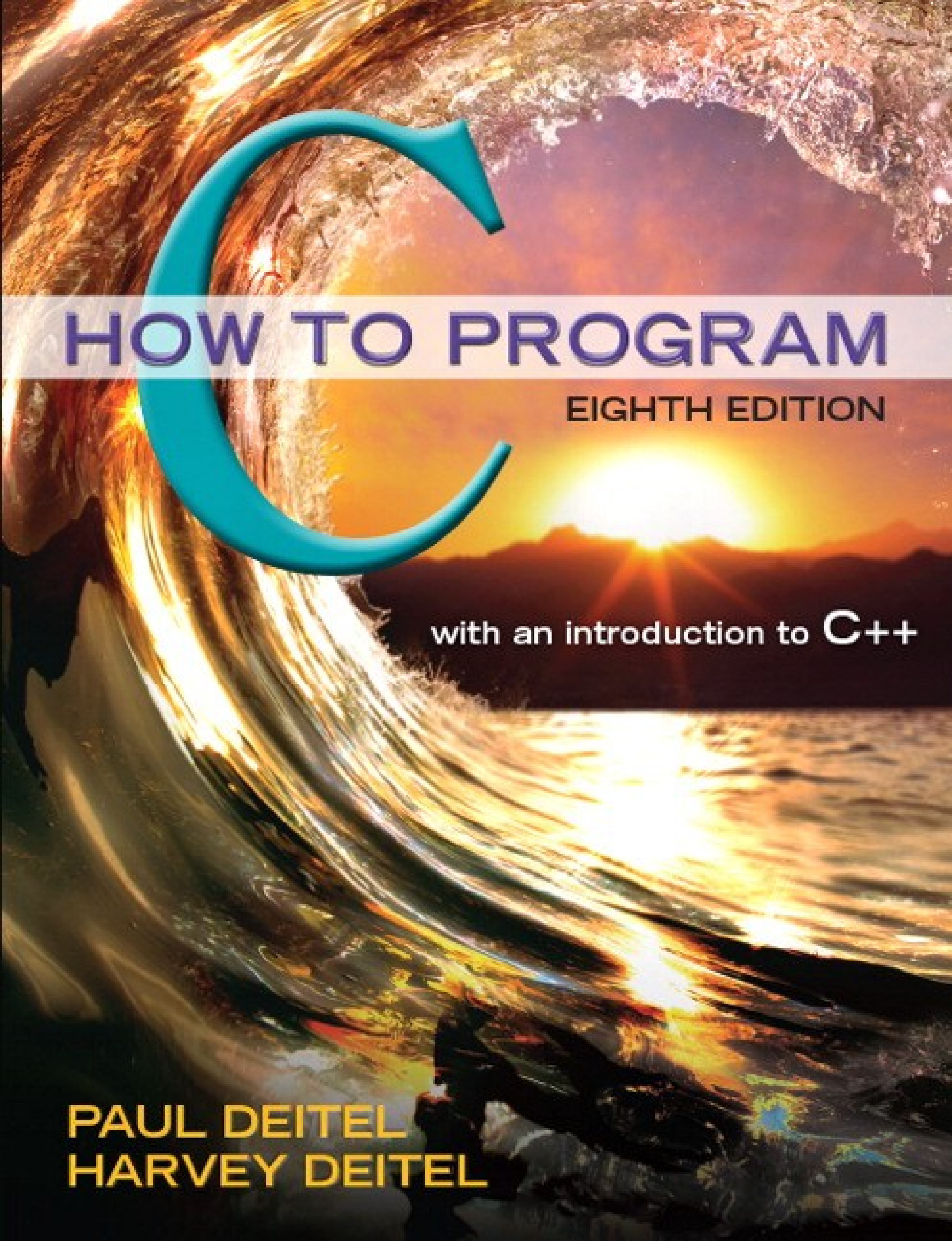
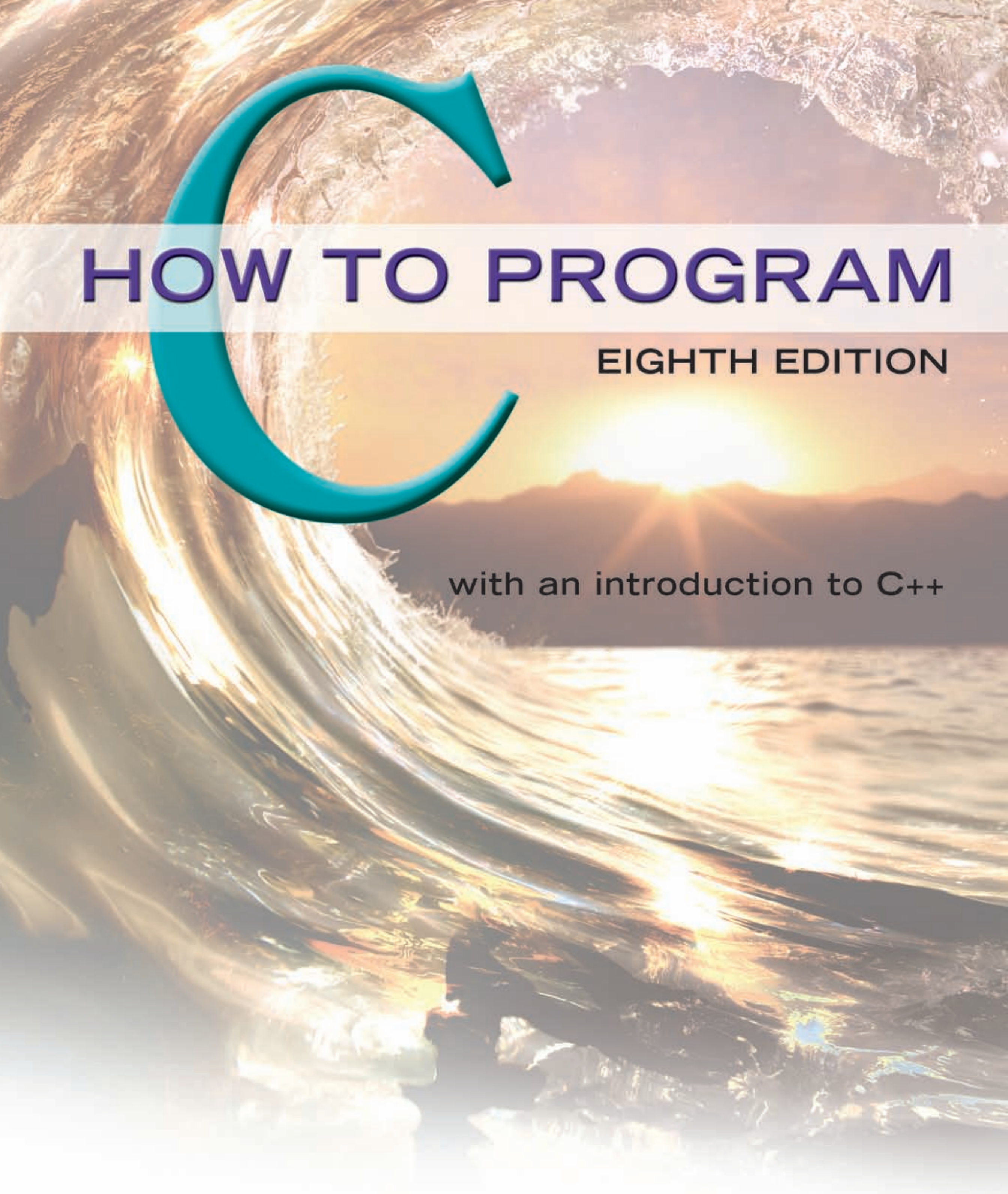


C
HOW TO PROGRAM

EIGHTH EDITION

with an introduction to C++

PAUL DEITEL
HARVEY DEITEL



C HOW TO PROGRAM

EIGHTH EDITION

with an introduction to C++

Deitel® Series Page

How to Program Series

Android™ How to Program, 2/E
C++ How to Program, 9/E
C How to Program, 7/E
Java™ How to Program, Early Objects Version, 10/E
Java™ How to Program, Late Objects Version, 10/E
Internet & World Wide Web How to Program, 5/E
Visual Basic® 2012 How to Program, 6/E
Visual C#® 2012 How to Program, 5/E

Deitel® Developer Series

Android™ for Programmers: An App-Driven Approach, 2/E, Volume 1
C for Programmers with an Introduction to C11
C++11 for Programmers
C# 2012 for Programmers
iOS® 8 for Programmers: An App-Driven Approach with Swift™, Volume 1
Java™ for Programmers, 3/E
JavaScript for Programmers
Swift™ for Programmers

Simply Series

Simply C++: An App-Driven Tutorial Approach
Simply Java™ Programming: An App-Driven Tutorial Approach

(continued in next column)

(continued from previous column)

Simply C#: An App-Driven Tutorial Approach
Simply Visual Basic® 2010: An App-Driven Approach, 4/E

CourseSmart Web Books

`www.deitel.com/books/CourseSmart/`

C++ How to Program, 8/E and 9/E
Simply C++: An App-Driven Tutorial Approach
Java™ How to Program, 9/E and 10/E
Simply Visual Basic® 2010: An App-Driven Approach, 4/E
Visual Basic® 2012 How to Program, 6/E
Visual Basic® 2010 How to Program, 5/E
Visual C#® 2012 How to Program, 5/E
Visual C#® 2010 How to Program, 4/E

LiveLessons Video Learning Products

`www.deitel.com/books/LiveLessons/`

Android™ App Development Fundamentals, 2/e
C++ Fundamentals
Java™ Fundamentals, 2/e
C# 2012 Fundamentals
C# 2010 Fundamentals
iOS® 8 App Development Fundamentals, 3/e
JavaScript Fundamentals
Swift™ Fundamentals

To receive updates on Deitel publications, Resource Centers, training courses, partner offers and more, please join the Deitel communities on

- Facebook®—`facebook.com/DeitelFan`
- Twitter®—`@deitel`
- Google+™—`google.com/+DeitelFan`
- YouTube™—`youtube.com/DeitelTV`
- LinkedIn®—`linkedin.com/company/deitel-&-associates`

and register for the free *Deitel® Buzz Online* e-mail newsletter at:

`www.deitel.com/newsletter/subscribe.html`

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

`deitel@deitel.com`

For information on *Dive-Into® Series* on-site seminars offered by Deitel & Associates, Inc. worldwide, write to us at `deitel@deitel.com` or visit:

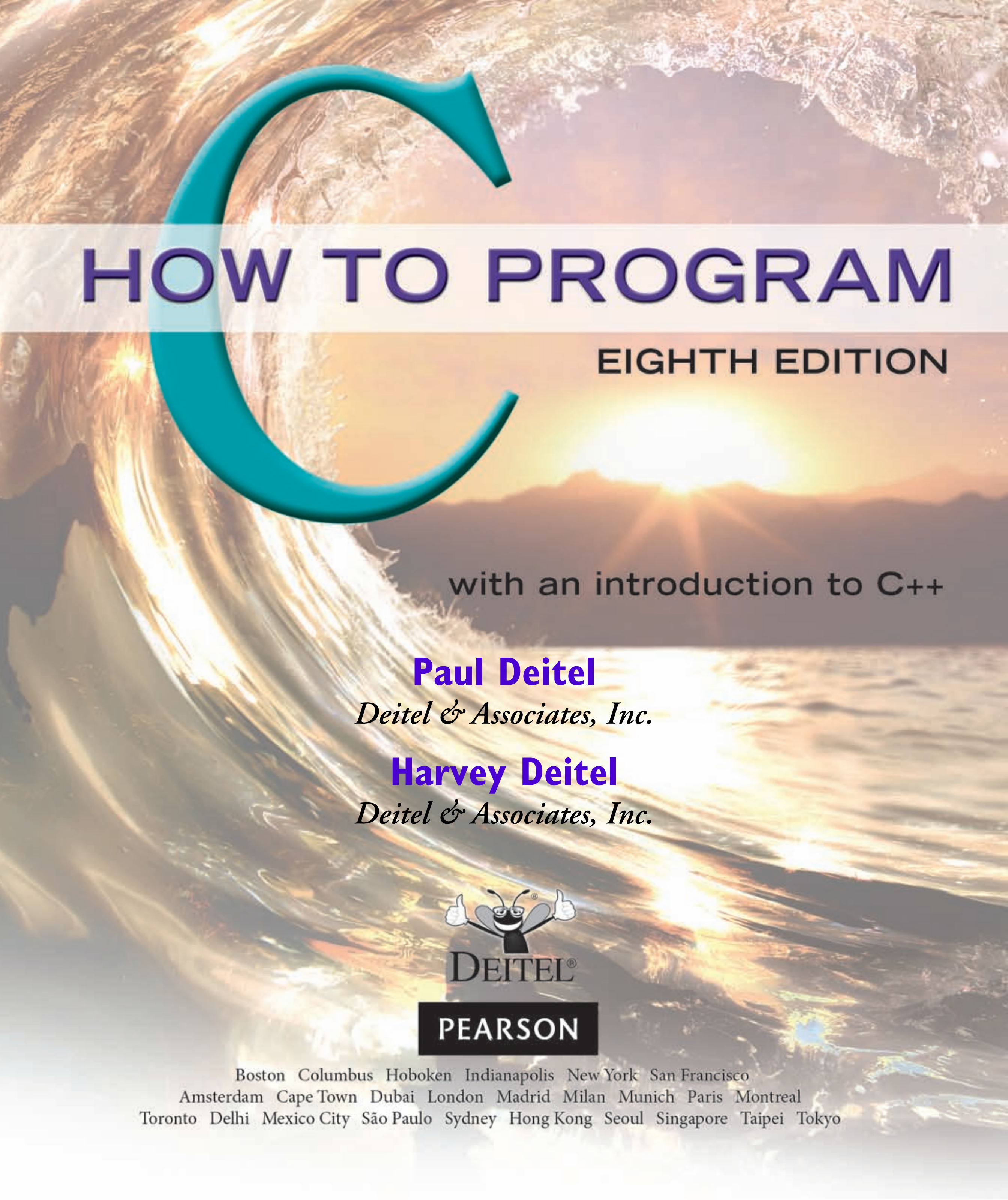
`www.deitel.com/training/`

For continuing updates on Pearson/Deitel publications visit:

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

Visit the Deitel Resource Centers that will help you master programming languages, software development, Android™ and iOS® app development, and Internet- and web-related topics:

`www.deitel.com/ResourceCenters.html`



C HOW TO PROGRAM

EIGHTH EDITION

with an introduction to C++

Paul Deitel
Deitel & Associates, Inc.

Harvey Deitel
Deitel & Associates, Inc.

DEITEL®

PEARSON

Boston Columbus Hoboken Indianapolis New York San Francisco
Amsterdam Cape Town Dubai London Madrid Milan Munich Paris Montreal
Toronto Delhi Mexico City São Paulo Sydney Hong Kong Seoul Singapore Taipei Tokyo

Vice President and Editorial Director, ECS: Marcia J. Horton
Executive Editor: Tracy Johnson (Dunkelberger)
Editorial Assistant: Kelsey Loanes
Program Manager: Carole Snyder
Project Manager: Robert Engelhardt
Media Team Lead: Steve Wright
R&P Manager: Rachel Youdelman
R&P Senior Project Manager: William Opaluch
Senior Operations Specialist: Maura Zaldivar-Garcia
Inventory Manager: Bruce Boundy
Marketing Manager: Demetrius Hall
Product Marketing Manager: Bram Van Kempen
Marketing Assistant: Jon Bryant
Cover Designer: Chuti Prasertsith / Michael Rutkowski / Marta Samsel
Cover Art: © Willyam Bradberry / Shutterstock

© 2016, 2013, 2010 Pearson Education, Inc., Hoboken, NJ 07030. All rights reserved. Manufactured in the United States of America. This publication is protected by Copyright and permissions 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 materials from this work, please submit a written request to Pearson Higher Education, Permissions Department, 221 River Street, Hoboken, NJ 07030.

Any of the designations by manufacturers and seller 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. Credits and acknowledgments borrowed from other sources and reproduced, with permission, in this textbook appears on page.

Microsoft® and Windows® are registered trademarks of the Microsoft Corporation in the U.S.A. and other countries. Screen shots and icons reprinted with permission from the Microsoft Corporation.

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 theories and programs to determine their effectiveness. The author and publisher make no warranty of any kind, expressed or implied, with regard to these programs or the documentation contained in this book. The author and publisher shall not be liable in any event for incidental or consequential damages with, or arising out of, the furnishing, performance, or use of these programs.

Pearson Education Ltd., London
Pearson Education Australia Ply. Ltd., Sydney
Pearson Education Singapore, Pte. Ltd.
Pearson Education North Asia Ltd., Hong Kong
Pearson Education Canada, Inc., Toronto
Pearson Education de Mexico, S.A. de C.V.
Pearson Education-Japan, Tokyo
Pearson Education Malaysia, Pte. Ltd.
Pearson Education, Inc., Hoboken, New Jersey

Library of Congress Cataloging-in-Publication Data
On file

10 9 8 7 6 5 4 3 2 1

www.pearsonhighered.com

ISBN-10: 0-13-397689-0
ISBN-13: 978-0-13-397689-2

PEARSON

In memory of Dennis Ritchie,
creator of the C programming language
and co-creator of the UNIX operating system.

Paul and Harvey Deitel

Trademarks

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

Apple, Xcode, Swift, Objective-C, iOS and OS X are trademarks or registered trademarks of Apple, Inc.

Java is a registered trademark of Oracle and/or its affiliates.

Microsoft and/or its respective suppliers make no representations about the suitability of the information contained in the documents and related graphics published as part of the services for any purpose. All such documents and related graphics are provided "as is" without warranty of any kind. Microsoft and/or its respective suppliers hereby disclaim all warranties and conditions with regard to this information, including all warranties and conditions of merchantability, whether express, implied or statutory, fitness for a particular purpose, title and non-infringement. In no event shall Microsoft and/or its respective suppliers be liable for any special, indirect or consequential damages or any damages whatsoever resulting from loss of use, data or profits, whether in an action of contract, negligence or other tortious action, arising out of or in connection with the use or performance of information available from the services.

The documents and related graphics contained herein could include technical inaccuracies or typographical errors. Changes are periodically added to the information herein. Microsoft and/or its respective suppliers may make improvements and/or changes in the product(s) and/or the program(s) described herein at any time. Partial screen shots may be viewed in full within the software version specified.

Other names may be trademarks of their respective owners.

Contents

Appendices F, G and H are PDF documents posted online at the book's Companion Website (located at `www.pearsonhighered.com/deitel`).

17 Classes: A Deeper Look; Throwing Exceptions 627

18 Operator Overloading; Class string 683

19 Object-Oriented Programming: Inheritance 732

Appendices F, G and H are PDF documents posted online at the book's Companion Website (located at www.pearsonhighered.com/deitel).

This page intentionally left blank

Preface

Welcome to the C programming language and to *C How to Program, Eighth Edition*! This book presents leading-edge computing technologies for college students, instructors and software-development professionals.

At the heart of the book is the Deitel signature "live-code approach"—we present concepts in the context of complete working programs, rather than in code snippets. Each code example is followed by one or more sample executions. Read the online Before You Begin section at

```
http://www.deitel.com/books/chtp8/chtp8_BYB.pdf
```

to learn how to set up your computer to run the hundreds of code examples. All the source code is available at

```
http://www.deitel.com/books/chtp8
```

and

```
http://www.pearsonhighered.com/deitel
```

Use the source code we provide to run every program as you study it.

We believe that this book and its support materials will give you an informative, challenging and entertaining introduction to C. As you read the book, if you have questions, send an e-mail to `deitel@deitel.com`—we'll respond promptly. For book updates, visit `www.deitel.com/books/chtp8/`, join our social media communities:

- Facebook®—`http://facebook.com/DeitelFan`
- Twitter®—`@deitel`
- LinkedIn®—`http://linkedin.com/company/deitel-&-associates`
- YouTube™—`http://youtube.com/DeitelTV`
- Google+™—`http://google.com/+DeitelFan`

and register for the *Deitel® Buzz Online* e-mail newsletter at:

```
http://www.deitel.com/newsletter/subscribe.html
```

New and Updated Features

Here are some key features of *C How to Program, 8/e*:

- ***Integrated More Capabilities of the C11 and C99 standards.*** Support for the C11 and C99 standards varies by compiler. Microsoft Visual C++ supports a subset of the features that were added to C in C99 and C11—primarily the features that are also required by the C++ standard. We incorporated several widely supported C11 and C99 features into the book's early chapters, as appropriate for introductory courses and for the compilers we used in this book. Appendix E, Multithreading and Other C11 and C99 Topics, presents more advanced features (such as multithreading for today's increasingly popular multi-core architectures) and various other features that are not widely supported by today's C compilers.

- *All Code Tested on Linux, Windows and OS X.* We retested all the example and exercise code using GNU gcc on Linux, Visual C++ on Windows (in Visual Studio 2013 Community Edition) and LLVM in Xcode on OS X.
- *Updated Chapter 1.* The new Chapter 1 engages students with updated intriguing facts and figures to get them excited about studying computers and computer programming. The chapter includes current technology trends and hardware discussions, the data hierarchy, social networking and a table of business and technology publications and websites that will help you stay up to date with the latest technology news and trends. We've included updated test-drives that show how to run a command-line C program on Linux, Microsoft Windows and OS X. We also updated the discussions of the Internet and web, and the introduction to object technology.
- *Updated Coverage of C++ and Object-Oriented Programming.* We updated Chapters 15–23 on object-oriented programming in C++ with material from our textbook *C++ How to Program, 9/e*, which is up-to-date with the C++11 standard.
- *Updated Code Style.* We removed the spacing inside parentheses and square brackets, and toned down our use of comments a bit. We also added parentheses to certain compound conditions for clarity.
- *Variable Declarations.* Because of improved compiler support, we were able to move variable declarations closer to where they're first used and define `for`-loop counter-control variables in each `for`'s initialization section.
- *Summary Bullets.* We removed the end-of-chapter terminology lists and updated the detailed section-by-section, bullet-list summaries with **bolded** key terms and, for most, page references to their defining occurrences.
- *Use of Standard Terminology.* To help students prepare to work in industry worldwide, we audited the book against the C standard and upgraded our terminology to use C standard terms in preference to general programming terms.
- *Online Debugger Appendices.* We've updated the online GNU `gdb` and Visual C++® debugging appendices, and added an Xcode® debugging appendix.
- *Additional Exercises.* We updated various exercises and added some new ones, including one for the Fisher-Yates unbiased shuffling algorithm in Chapter 10.

Other Features

Other features of *C How to Program, 8/e* include:

- *Secure C Programming Sections.* Many of the C chapters end with a Secure C Programming Section. We've also posted a Secure C Programming Resource Center at `www.deitel.com/SecureC/`. For more details, see the section "A Note About Secure C Programming" on the next page.

- *Focus on Performance Issues.* C (and C++) are favored by designers of performance-intensive systems such as operating systems, real-time systems, embedded systems and communications systems, so we focus intensively on performance issues.
- *"Making a Difference" Contemporary Exercises.* We encourage you to use computers and the Internet to research and solve significant problems. These exercises are meant to increase awareness of important issues the world is facing. We hope you'll approach them with your own values, politics and beliefs.
- *Sorting: A Deeper Look.* Sorting places data in order, based on one or more sort keys. We begin our sorting presentation in Chapter 6 with a simple algorithm—in Appendix D, we present a deeper look. We consider several algorithms and compare them with regard to their memory consumption and processor demands. For this purpose, we present a friendly introduction to Big O notation, which indicates how hard an algorithm may have to work to solve a problem. Through examples and exercises, Appendix D discusses the selection sort, insertion sort, recursive merge sort, recursive selection sort, bucket sort and recursive Quicksort. Sorting is an intriguing problem because different sorting techniques achieve the same final result but they can vary hugely in their consumption of memory, CPU time and other system resources.
- *Titled Programming Exercises.* Most of the programming exercises are titled to help instructors conveniently choose assignments appropriate for their students.
- *Order of Evaluation.* We caution the reader about subtle order of evaluation issues.
- *C++-Style // Comments.* We use the newer, more concise C++-style `//` comments in preference to C's older style `/*...*/` comments.

A Note About Secure C Programming

Throughout this book, we focus on C programming *fundamentals*. When we write each *How to Program* book, we search the corresponding language's standards document for the features that we feel novices need to learn in a first programming course, and features that professional programmers need to know to begin working in that language. We also cover computer-science and software-engineering fundamentals for novices—our core audience.

Industrial-strength coding techniques in any programming language are beyond the scope of an introductory textbook. For that reason, our Secure C Programming sections present some key issues and techniques, and provide links and references so you can continue learning.

Experience has shown that it's difficult to build industrial-strength systems that stand up to attacks from viruses, worms, etc. Today, via the Internet, such attacks can be instantaneous and global in scope. Software vulnerabilities often come from simple programming issues. Building security into software from the start of the development cycle can greatly reduce costs and vulnerabilities.

The CERT® Coordination Center (`www.cert.org`) was created to analyze and respond promptly to attacks. CERT—the Computer Emergency Response Team—publishes and promotes secure coding standards to help C programmers and others implement industrial-strength systems that avoid the programming practices that leave systems vulnerable to attacks. The CERT standards evolve as new security issues arise.

We've upgraded our code (as appropriate for an introductory book) to conform to various CERT recommendations. If you'll be building C systems in industry, consider reading *The CERT C Secure Coding Standard, 2/e* (Robert Seacord, Addison-Wesley Professional, 2014) and *Secure Coding in C and C++, 2/e* (Robert Seacord, Addison-Wesley Professional, 2013). The CERT guidelines are available free online at

```
    https://www.securecoding.cert.org/confluence/display/seccode/
      CERT+C+Coding+Standard
```

Mr. Seacord, a technical reviewer for the C portion of the last edition of this book, provided specific recommendations on each of our Secure C Programming sections. Mr. Seacord is the Secure Coding Manager at CERT at Carnegie Mellon University's Software Engineering Institute (SEI) and an adjunct professor in the Carnegie Mellon University School of Computer Science.

The Secure C Programming sections at the ends of Chapters 2–13 discuss many important topics, including:

- testing for arithmetic overflows
- using unsigned integer types
- the more secure functions in the C standard's Annex K
- the importance of checking the status information returned by standard-library functions
- range checking
- secure random-number generation
- array bounds checking
- preventing buffer overflows
- input validation
- avoiding undefined behaviors
- choosing functions that return status information vs. using similar functions that do not
- ensuring that pointers are always NULL or contain valid addresses
- using C functions vs. using preprocessor macros, and more.

Web-Based Materials

The book's open access Companion Website (`http://www.pearsonhighered.com/deitel`) contains source code for all the code examples and the following appendices in PDF format:

- Appendix F, Using the Visual Studio Debugger
- Appendix G, Using the GNU gdb Debugger
- Appendix H, Using the Xcode Debugger

Dependency Charts

Figures 1 and 2 on the next two pages show the dependencies among the chapters to help instructors plan their syllabi. *C How to Program, 8/e* is appropriate for CS1 and many CS2 courses, and for intermediate-level C and C++ programming courses. The C++ part of the book assumes that you've studied C Chapters 1–10.

Teaching Approach

C How to Program, 8/e, contains a rich collection of examples. We focus on good software engineering, program clarity, preventing common errors, program portability and performance issues.

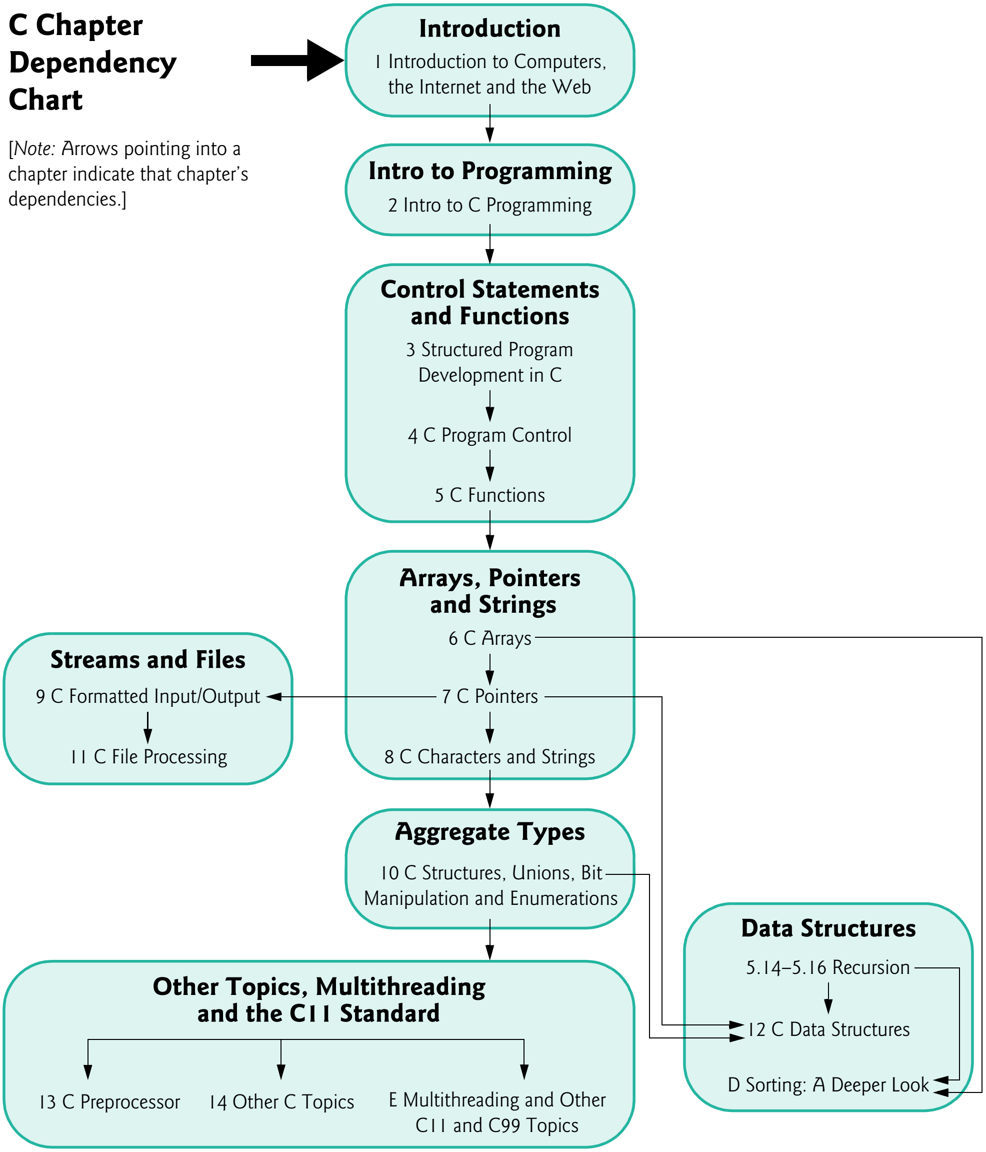

C Chapter Dependency Chart
[Note: Arrows pointing into a chapter indicate that chapter's dependencies.]
Introduction
1 Introduction to Computers, the Internet and the Web
Intro to Programming
2 Intro to C Programming
Control Statements and Functions
3 Structured Program Development in C
4 C Program Control
5 C Functions
Arrays, Pointers and Strings
6 C Arrays
7 C Pointers
8 C Characters and Strings
Streams and Files
9 C Formatted Input/Output
11 C File Processing
Aggregate Types
10 C Structures, Unions, Bit Manipulation and Enumerations
Data Structures
5.14–5.16 Recursion
12 C Data Structures
D Sorting: A Deeper Look
Other Topics, Multithreading and the C11 Standard
13 C Preprocessor
14 Other C Topics
E Multithreading and Other C11 and C99 Topics

Fig. 1 | C chapter dependency chart.

Syntax Shading. For readability, we syntax shade the code, similar to the way most IDEs and code editors syntax color code. Our syntax-shading conventions are:

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

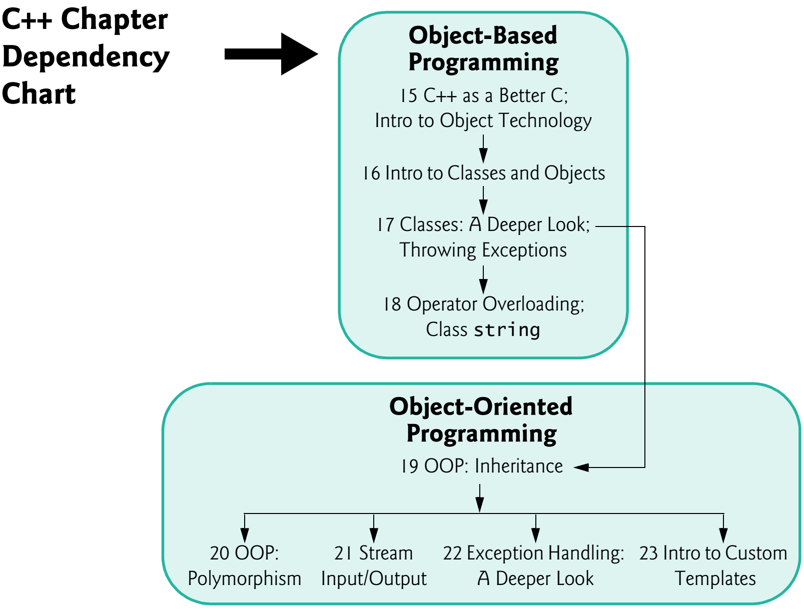


Fig. 2 | C++ chapter dependency chart.

Code Highlighting. We place gray rectangles around the key code in each program.

Using Fonts for Emphasis. We place the key terms and the index's page reference for each defining occurrence in **bold colored** text for easy reference. We emphasize C program text in the `Lucida` font (for example, `int x = 5;`).

Objectives. Each chapter begins with a list of objectives.

Illustrations/Figures. Abundant flowcharts, tables, line drawings, UML diagrams (in the C++ chapters), 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 eight 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 and for avoiding bugs 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 detailed section-by-section, bullet-list summary of each chapter with **bolded** key terms. For easy reference, most of the key terms are followed by the page number of their defining occurrences.

Self-Review Exercises and Answers. Extensive self-review exercises *and* answers are included for self-study.

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

- simple recall of important terminology and concepts
- identifying the errors in code samples
- writing individual program statements
- writing small portions of C functions (and C++ member functions and classes)
- writing complete programs
- implementing major projects

Index. We've included an extensive index, which is especially helpful when you use the book as a reference. Defining occurrences of key terms are highlighted in the index with a **bold colored** page number.

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

We tested the programs in *C How to Program, 8/e* using the following free compilers:

- GNU C and C++ (`http://gcc.gnu.org/install/binaries.html`), which are already installed on most Linux systems and can be installed on OS X and Windows systems.
- Microsoft's Visual C++ in Visual Studio 2013 Community edition, which you can download from `http://go.microsoft.com/?linkid=9863608`
- LLVM in Apple's Xcode IDE, which OS X users can download from the Mac App Store.

For other free C and C++ compilers, visit:

```
http://www.thefreecountry.com/compilers/cpp.shtml
http://www.compilers.net/Dir/Compilers/CCpp.htm
http://www.freebyte.com/programming/cpp/#cppcompilers
http://en.wikipedia.org/wiki/List_of_compilers#C.2B.2B_compilers
```

CourseSmart Web Books

Today's students and instructors have increasing demands on their time and money. Pearson has responded to that need by offering various digital texts and course materials online through CourseSmart. Faculty can review course materials online, saving time and costs. It offers students a high-quality digital version of the text for less than the cost of a print copy. Students receive the same content offered in the print textbook enhanced by search, note-taking and printing tools. For more information, visit `http://www.coursesmart.com`.

Instructor Resources

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

- ***PowerPoint*® *slides*** containing all the code and figures in the text, plus bulleted items that summarize key points.
- ***Test Item File*** of multiple-choice questions (approximately two per top-level book section)
- ***Solutions Manual*** with solutions to *most* (but not all) of the end-of-chapter exercises. Please check the Instructor Resource Center to determine which exercises have solutions.

Please do not write to us requesting access to the Instructor Resource Center. Access is restricted to college instructors teaching from the book. Instructors may obtain access only through their Pearson representatives. If you're not a registered faculty member, contact your Pearson representative or visit `http://www.pearsonhighered.com/replocator/`.

Solutions are *not* provided for "project" exercises. Check out our Programming Projects Resource Center for lots of additional exercise and project possibilities (`http://www.deitel.com/ProgrammingProjects/`).

Acknowledgments

We'd like to thank Abbey Deitel and Barbara Deitel for long hours devoted to this project. Abbey co-authored Chapter 1. We're fortunate to have worked with the dedicated team of publishing professionals at Pearson. We appreciate the guidance, savvy and energy of Tracy Johnson, Executive Editor, Computer Science. Kelsey Loanes and Bob Engelhardt did a marvelous job managing the review and production processes, respectively.

C How to Program, 8/e *Reviewers*

We wish to acknowledge the efforts of our reviewers. Under tight deadlines, they scrutinized the text and the programs and provided countless suggestions for improving the presentation: Dr. Brandon Invergo (GNU/European Bioinformatics Institute), Danny Kalev (A Certified System Analyst, C Expert and Former Member of the C++ Standards Committee), Jim Hogg (Program Manager, C/C++ Compiler Team, Microsoft Corporation), José Antonio González Seco (Parliament of Andalusia), Sebnem Onsay (Special Instructor, Oakland University School of Engineering and Computer Science), Alan Bunning (Purdue University), Paul Clingan (Ohio State University), Michael Geiger (University of Massachusetts, Lowell), Jeonghwa Lee (Shippensburg University), Susan Mengel (Texas Tech University), Judith O'Rourke (SUNY at Albany) and Chen-Chi Shin (Radford University).

Other Recent Editions Reviewers

William Albrecht (University of South Florida), Ian Barland (Radford University), Ed James Beckham (Altera), John Benito (Blue Pilot Consulting, Inc. and Convener of ISO WG14—the Working Group responsible for the C Programming Language Standard), Dr. John F. Doyle (Indiana University Southeast), Alireza Fazelpour (Palm Beach Community College), Mahesh Hariharan (Microsoft), Hemanth H.M. (Software Engineer at SonicWALL), Kevin Mark Jones (Hewlett Packard), Lawrence Jones, (UGS Corp.), Don Kostuch (Independent Consultant), Vytautus Leonavicius (Microsoft), Xiaolong Li (Indiana State University), William Mike Miller (Edison Design Group, Inc.), Tom Rethard (The University of Texas at Arlington), Robert Seacord (Secure Coding Manager at SEI/CERT, author of *The CERT C Secure Coding Standard* and technical expert for the international standardization working group for the programming language C), José Antonio González Seco (Parliament of Andalusia), Benjamin Seyfarth (University of Southern Mississippi), Gary Sibbitts (St. Louis Community College at Meramec), William Smith (Tulsa Community College) and Douglas Walls (Senior Staff Engineer, C compiler, Sun Microsystems—now part of Oracle).

A Special Thank You to Brandon Invergo and Jim Hogg

We were privileged to have Brandon Invergo (GNU/European Bioinformatics Institute) and Jim Hogg (Program Manager, C/C++ Compiler Team, Microsoft Corporation) do full-book reviews. They scrutinized the C portion of the book, providing numerous insights and constructive comments. The largest part of our audience uses either the GNU gcc compiler or Microsoft's Visual C++ compiler (which also compiles C). Brandon and Jim helped us ensure that our content was accurate for the GNU and Microsoft compilers, respectively. Their comments conveyed a love of software engineering, computer science and education that we share.

Well, there you have it! C is a powerful programming language that will help you write high-performance, portable programs quickly and effectively. It scales nicely into the realm of enterprise systems development to help organizations build their business-critical and mission-critical information systems. As you read the book, we would sincerely appreciate your comments, criticisms, corrections and suggestions for improving the text. Please address all correspondence—including questions—to:

```
deitel@deitel.com
```

We'll respond promptly, and post corrections and clarifications on:

```
www.deitel.com/books/chtp8/
```

We hope you enjoy working with *C How to Program, Eighth Edition* as much as we enjoyed writing it!

Paul Deitel
Harvey Deitel

About the Authors

Paul Deitel, CEO and Chief Technical Officer of Deitel & Associates, Inc., is a graduate of MIT, where he studied Information Technology. Through Deitel & Associates, Inc., he has delivered hundreds of programming courses to industry clients, including Cisco, IBM, Siemens, Sun Microsystems, Dell, Lucent Technologies, Fidelity, NASA at the Kennedy Space Center, the National Severe Storm Laboratory, White Sands Missile Range, Hospital Sisters Health System, 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/professional book/video authors.

Dr. Harvey M. Deitel, Chairman and Chief Strategy Officer of Deitel & Associates, Inc., has 54 years of experience in the computer field. Dr. Deitel earned B.S. and M.S. degrees in electrical engineering from MIT and a Ph.D. in mathematics from Boston University (all with a focus on computing). 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 in 1991 with his son, Paul Deitel. The Deitels' publications have earned international recognition, with translations published in Chinese, Korean, Japanese, German, Russian, Spanish, French, Polish, Italian, Portuguese, Greek, Urdu and Turkish. Dr. Deitel has delivered hundreds of programming courses to academic institutions, major corporations, government organizations and the military.

About Deitel & Associates, Inc.

Deitel & Associates, Inc., founded by Paul Deitel and Harvey Deitel, is an internationally recognized authoring and corporate training organization, specializing in computer programming languages, object technology, mobile app development and Internet and web software technology. The company's training clients include many of the world's largest companies, government agencies, branches of the military, and academic institutions. The company offers instructor-led training courses delivered at client sites worldwide on major programming languages and platforms, including C, C++, Java™, Android app development, Swift™ and iOS® app development, Visual C#®, Visual Basic®, Visual C++®, Python®, object technology, Internet and web programming and a growing list of additional programming and software development courses.

Through its 40-year publishing partnership with Pearson/Prentice Hall, Deitel & Associates, Inc., publishes leading-edge programming textbooks and professional books in print and popular e-book formats, and *LiveLessons* video courses (available on Safari Books Online and other video platforms). Deitel & Associates, Inc. and the authors can be reached at:

```
deitel@deitel.com
```

To learn more about Deitel's *Dive-Into® Series* Corporate Training curriculum delivered to groups of software engineers at client sites worldwide, visit:

```
http://www.deitel.com/training
```

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

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

```
http://www.informit.com/store/sales.aspx
```

Introduction to Computers, the Internet and the Web

1

Objectives

In this chapter, you'll learn:

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

Outline

1.1 Introduction

Welcome to C and C++! C is a concise yet powerful computer programming language that's appropriate for technically oriented people with little or no programming experience and for experienced programmers to use in building substantial software systems. *C How to Program, Eighth Edition*, is an effective learning tool for each of these audiences.

The core of the book emphasizes software engineering through the proven methodologies of *structured programming* in C and *object-oriented programming* in C++. The book presents hundreds of complete working programs and shows the outputs produced when those programs are run on a computer. We call this the "live-code approach." All of these example programs may be downloaded from our website `www.deitel.com/books/chtp8/`.

Most people are familiar with the exciting tasks that computers perform. Using this textbook, you'll learn how to command computers to perform those tasks. It's **software** (i.e., the instructions you write to command computers to perform **actions** and make **decisions**) that controls computers (often referred to as **hardware**).

1.2 Hardware and Software

Computers can perform calculations and make logical decisions phenomenally faster than human beings can. Many of today's personal computers can perform billions of calculations in one second—more than a human can perform in a lifetime. *Supercomputers* are already performing *thousands of trillions (quadrillions)* of instructions per second! China's National University of Defense Technology's Tianhe-2 supercomputer can perform over 33 quadrillion calculations per second (33.86 *petaflops*)![1] To put that in perspective, *the Tianhe-2 supercomputer can perform in one second about 3 million calculations for every person on the planet!* And supercomputing "upper limits" are growing quickly.

Computers process data under the control of sequences of instructions called **computer programs**. These software programs guide the computer through ordered actions specified by people called computer **programmers**.

A computer consists of various devices referred to as hardware (e.g., the keyboard, screen, mouse, hard disks, memory, DVD drives and processing units). Computing costs are dropping dramatically, owing to rapid developments in hardware and software technologies. Computers that might have filled large rooms and cost millions of dollars decades ago are now inscribed on silicon chips smaller than a fingernail, costing perhaps a few dollars each. Ironically, silicon is one of the most abundant materials on Earth—it's an ingredient in common sand. Silicon-chip technology has made computing so economical that computers have become a commodity.

1.2.1 Moore's Law

Every year, you probably expect to pay at least a little more for most products and services. The opposite has been the case in the computer and communications fields, especially with regard to the hardware supporting these technologies. For many decades, hardware costs have fallen rapidly.

Every year or two, the capacities of computers have approximately *doubled* inexpensively. This remarkable trend often is called **Moore's Law**, named for the person who identified it in the 1960s, Gordon Moore, co-founder of Intel—the leading manufacturer of the processors in today's computers and embedded systems. Moore's Law and *related* observations apply especially to the amount of memory that computers have for programs, the amount of secondary storage (such as disk storage) they have to hold programs and data over longer periods of time, and their processor speeds—the speeds at which they *execute* their programs (i.e., do their work).

Similar growth has occurred in the communications field—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] `http://www.top500.org`.

1.2.2 Computer Organization

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

Logical unit	Description
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 user input is entered into computers through keyboards, touch screens and mouse devices. Other forms of input include receiving voice commands, scanning images and barcodes, reading from secondary storage devices (such as hard drives, DVD drives, Blu-ray Disc™ drives and USB flash drives—also called "thumb drives" or "memory sticks"), receiving video from a webcam and having your computer receive information from the Internet (such as when you stream videos from YouTube® or download e-books from Amazon). Newer forms of input include position data from a GPS device, and motion and orientation information from an *accelerometer* (a device that responds to up/down, left/right and forward/backward acceleration) in a smartphone or game controller (such as Microsoft® Kinect® for Xbox®, Wii™ Remote and Sony® PlayStation® Move).
Output unit	This "shipping" section takes information the computer has processed and places it on various **output devices** to make it available for use outside the computer. Most information that's output from computers today is displayed on screens (including touch screens), printed on paper ("going green" discourages this), played as audio or video on PCs and media players (such as Apple's iPods) and giant screens in sports stadiums, transmitted over the Internet or used to control other devices, such as robots and "intelligent" appliances. Information is also commonly output to secondary storage devices, such as hard drives, DVD drives and USB flash drives. Popular recent forms of output are smartphone and game controller vibration, and virtual reality devices like Oculus Rift.
Memory unit	This rapid-access, relatively low-capacity "warehouse" section retains information that has been entered through the input unit, making it immediately available for processing when needed. The memory unit also retains processed information until it can be placed on output devices by the output unit. Information in the memory unit is *volatile*—it's typically lost when the computer's power is turned off. The memory unit is often called either **memory**, **primary memory** or **RAM** (Random Access Memory). Main memories on desktop and notebook computers contain as much as 128 GB of RAM, though 2 to 16 GB is most common. GB stands for gigabytes; a gigabyte is approximately one billion bytes. A **byte** is eight bits. A bit is either a `0` or a `1`.
Arithmetic and logic unit (ALU)	This "manufacturing" section performs *calculations*, such as addition, subtraction, multiplication and division. It also contains the *decision* mechanisms that allow the computer, for example, to compare two items from the memory unit to determine whether they're equal. In today's systems, the ALU is implemented as part of the next logical unit, the CPU.

Fig. 1.1 \| Logical units of a computer. (Part 1 of 2.)

Logical unit	Description
Central processing unit (CPU)	This "administrative" section coordinates and supervises the operation of the other sections. The CPU tells the input unit when information should be read into the memory unit, tells the ALU when information from the memory unit should be used in calculations and tells the output unit when to send information from the memory unit to certain output devices. Many of today's computers have multiple CPUs and, hence, can perform many operations simultaneously. A **multi-core processor** implements multiple processors on a single integrated-circuit chip—a *dual-core processor* has two CPUs and a *quad-core processor* has four CPUs. Today's desktop computers have processors that can execute billions of instructions per second.
Secondary storage unit	This is the long-term, high-capacity "warehousing" section. Programs or data not actively being used by the other units normally are placed on secondary storage devices (e.g., your *hard drive*) until they're again needed, possibly hours, days, months or even years later. Information on secondary storage devices is *persistent*—it's preserved even when the computer's power is turned off. Secondary storage information takes much longer to access than information in primary memory, but its cost per unit is much less. Examples of secondary storage devices include hard drives, DVD drives and USB flash drives, some of which can hold over 2 TB (TB stands for terabytes; a terabyte is approximately one trillion bytes). Typical hard drives on desktop and notebook computers hold up to 2 TB, and some desktop hard drives can hold up to 6 TB.

Fig. 1.1 | Logical units of a computer. (Part 2 of 2.)

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 the simplest data items (called "bits") to richer ones, such as characters and fields. Figure 1.2 illustrates a portion of the data hierarchy.

Bits

The smallest data item in a computer can assume the value `0` or the value `1`. It's called a **bit** (short for "binary digit"—a digit that can assume one of *two* values). Remarkably, the impressive functions performed by computers involve only the simplest manipulations of `0`s and `1`s—*examining a bit's value*, *setting a bit's value* and *reversing a bit's value* (from `1` to `0` or from `0` to `1`).

Characters

It's tedious for people to work with data in the low-level form of bits. Instead, they prefer to work with *decimal digits* (0–9), *letters* (A–Z and a–z), and *special symbols* (e.g., $, @, %, &, *, (,), –, +, ", :, ? and /). Digits, letters and special symbols are known as **characters**. The computer's **character set** is the set of all the characters used to write programs and represent data items. Computers process only `1`s and `0`s, so a computer's character set represents every character as a pattern of `1`s and `0`s. C supports various character sets (including **Unicode**®) that are composed of characters containing one, two or four bytes (8, 16 or 32 bits). Unicode contains characters for many of the world's languages. See Appendix B for more information on the **ASCII (American Standard Code for Information Interchange)** character set—the popular subset of Unicode that represents uppercase and lowercase letters, digits and some common special characters.

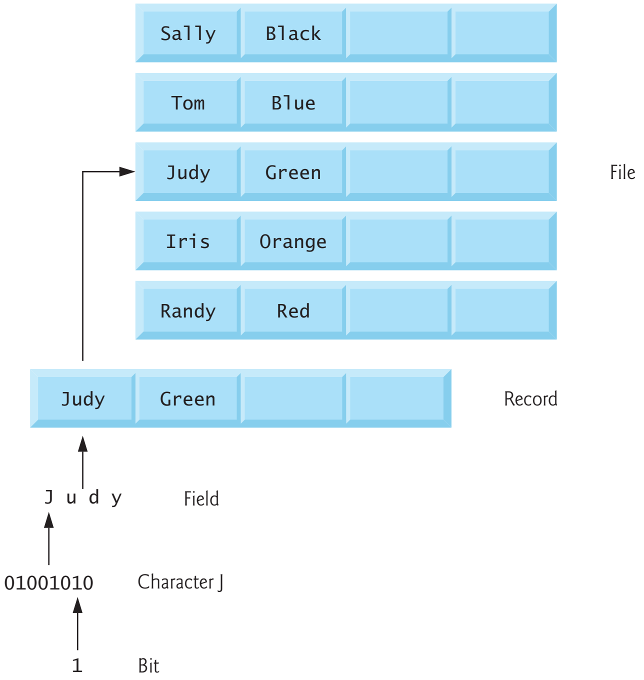


Fig. 1.2 | Data hierarchy.

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**. 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.

Files

A **file** is a group of related records. [*Note:* More generally, a file contains arbitrary data in arbitrary formats. In some operating systems, a file is viewed simply as a *sequence of bytes*—any organization of the bytes in a file, such as organizing the data into records, is a view created by the application programmer.] It's not unusual for an organization to have many files, some containing billions, or even trillions, of characters of information.

Database

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

Big Data

The amount of data being produced worldwide is enormous and growing quickly. According to IBM, approximately 2.5 quintillion bytes (2.5 *exabytes*) of data are created daily and 90% of the world's data was created in just the past two years![2] According to an IDC study, the global data supply will reach 40 *zettabytes* (equal to 40 trillion gigabytes) annually by 2020.[3] Figure 1.3 shows some common byte measurements. **Big data** applications deal with massive amounts of data and this field is growing quickly, creating lots of opportunity for software developers. According to a study by Gartner Group, over 4 million IT jobs globally will support big data by 2015.[4]

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

Fig. 1.3 | Byte measurements.

[2] `http://www.ibm.com/smarterplanet/us/en/business_analytics/article/it_business_intelligence.html`.

[3] `http://recode.net/2014/01/10/stuffed-why-data-storage-is-hot-again-really/`.

[4] `http://tech.fortune.cnn.com/2013/09/04/big-data-employment-boom/`.

1.4 Machine Languages, Assembly Languages and High-Level Languages

Programmers write instructions in various programming languages, some directly understandable by computers and others requiring intermediate *translation* steps. 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

Machine 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 payroll program that adds overtime pay to base pay and stores the result in gross pay:

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

Assembly Languages and Assemblers

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 payroll program also adds overtime pay to base pay and stores the result in gross pay:

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

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

High-Level Languages and Compilers

With the advent of assembly languages, computer usage increased rapidly, but programmers still had to use numerous 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 is one of the most widely used high-level programming languages.

Interpreters

Compiling a large high-level language program into machine language can take considerable computer time. *Interpreter* programs, developed to execute high-level language programs directly, avoid the delay of compilation, although they run slower than compiled programs.

1.5 The C Programming Language

C evolved from two previous languages, BCPL and B. BCPL was developed in 1967 by Martin Richards as a language for writing operating systems and compilers. Ken Thompson modeled many features in his B language after their counterparts in BCPL, and in 1970 he used B to create early versions of the UNIX operating system at Bell Laboratories.

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

Built for Performance

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

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

Fig. 1.4 | Some popular performance-oriented C applications. (Part 1 of 2.)