C How to Program

DEITEL®

NINTH EDITION

with

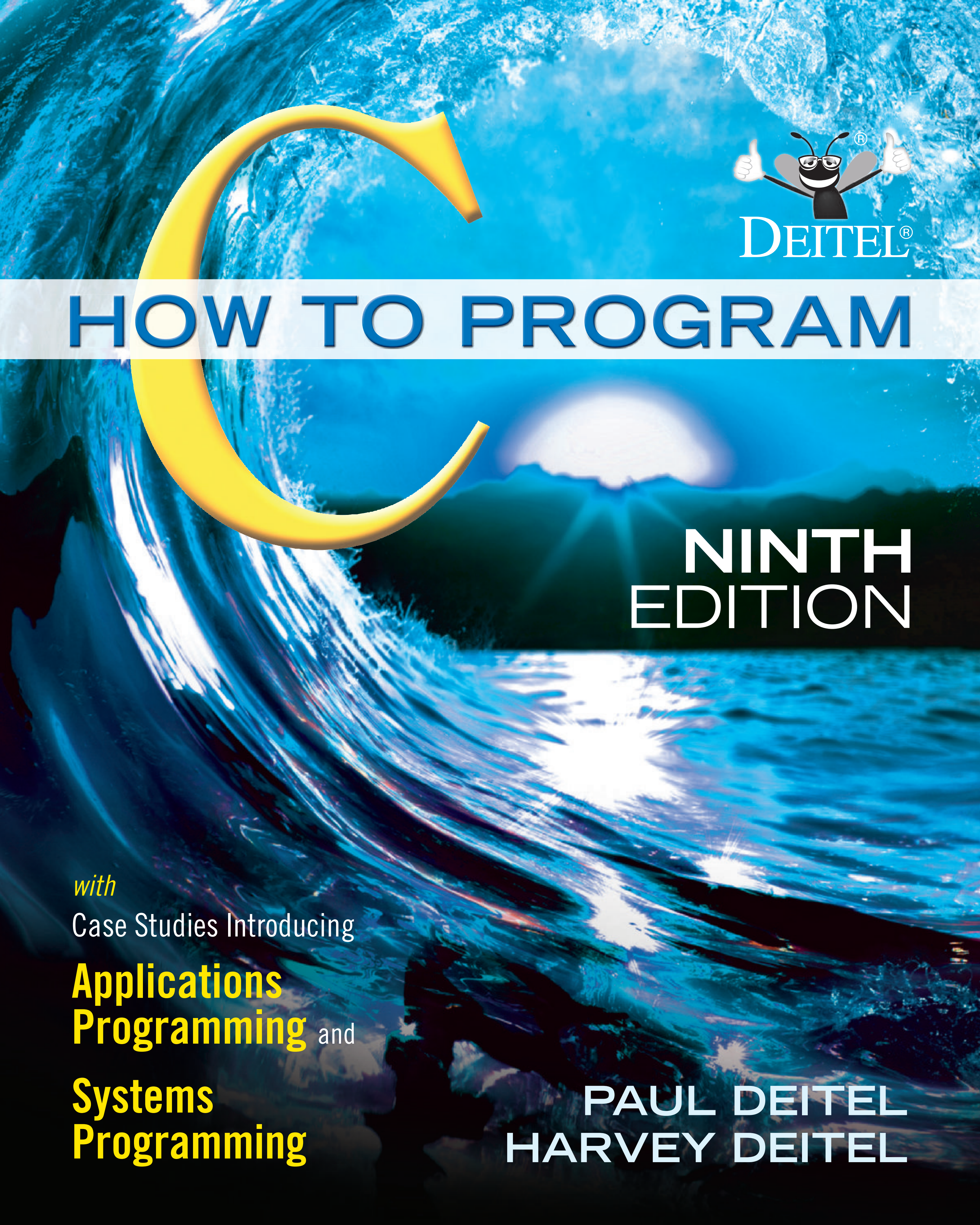
Case Studies Introducing

Applications Programming and

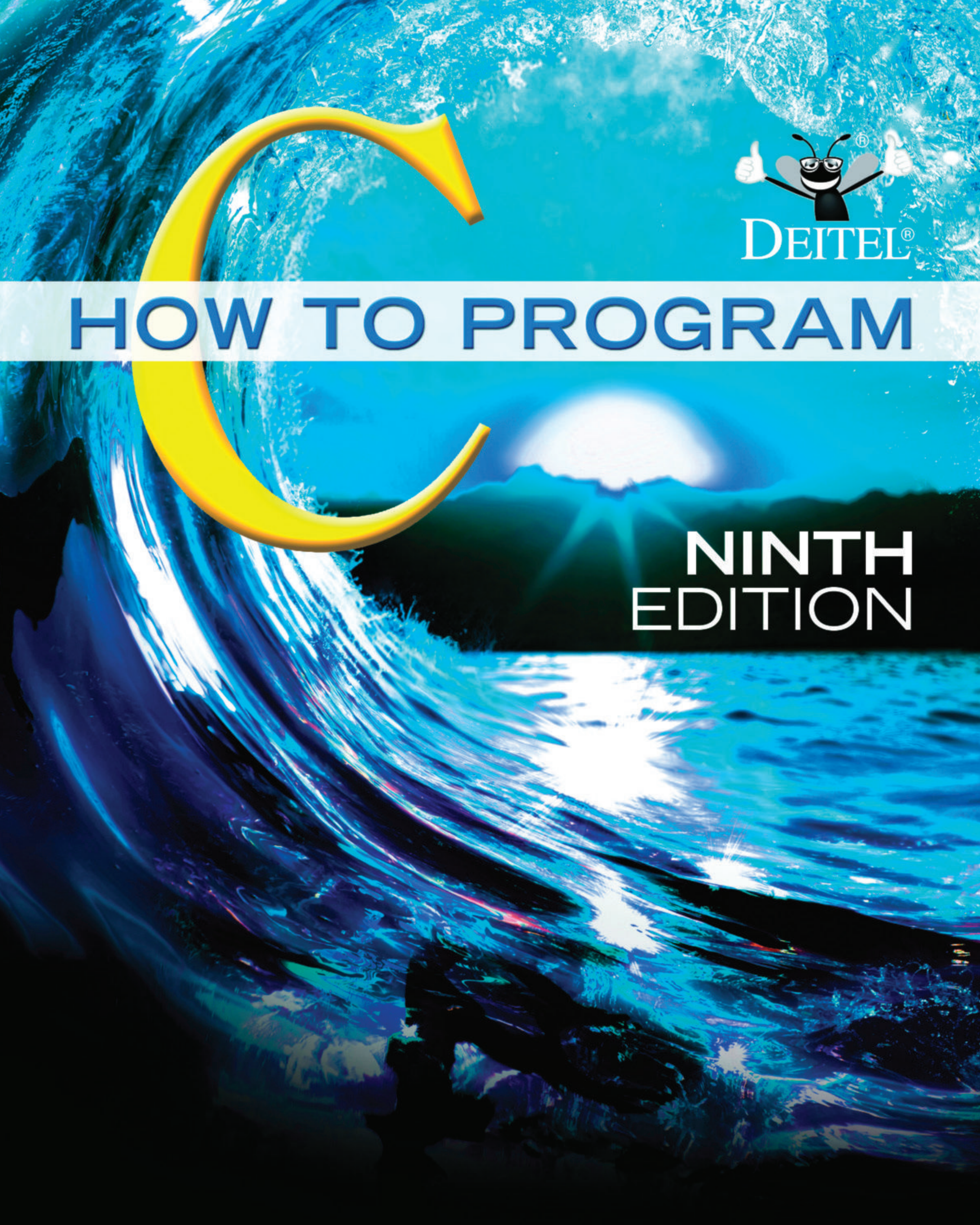
Systems Programming

PAUL DEITEL

HARVEY DEITEL



This page intentionally left blank



DEITEL®

C HOW TO PROGRAM

NINTH EDITION

Deitel® Series Page

Intro to Series

Intro to Python® for Computer Science and Data Science: Learning to Program with AI, Big Data and the Cloud

How To Program Series

C How to Program, 9/E
Java™ How to Program, Early Objects Version, 11/E
Java™ How to Program, Late Objects Version, 11/E
C++ How to Program, 10/E
Android™ How to Program, 3/E
Internet & World Wide Web How to Program, 5/E
Visual Basic® 2012 How to Program, 6/E
Visual C#® How to Program, 6/E

LiveLessons Video Training

`https://deitel.com/LiveLessons/`

Python® Fundamentals
Java™ Fundamentals
C++20 Fundamentals
C11/C18 Fundamentals
C# 6 Fundamentals
Android™ 6 Fundamentals, 3/E
C# 2012 Fundamentals
JavaScript Fundamentals
Swift™ Fundamentals

REVEL™ Interactive Multimedia

REVEL™ for Deitel Java™
REVEL™ for Deitel Python®

E-Books

`https://VitalSource.com`
`https://RedShelf.com`
`https://Chegg.com`

Intro to Python® for Computer Science and Data Science: Learning to Program with AI, Big Data and the Cloud
Java™ How to Program, 10/E and 11/E
C++ How to Program, 9/E and 10/E
C How to Program, 8/E and 9/E
Android™ How to Program, 2/E and 3/E
Visual Basic® 2012 How to Program, 6/E
Visual C#® How to Program, 6/E

Deitel® Developer Series

Python® for Programmers
Java™ for Programmers, 4/E
C++20 for Programmers
Android™ 6 for Programmers: An App-Driven Approach, 3/E
C for Programmers with an Introduction to C11
C# 6 for Programmers
JavaScript for Programmers
Swift™ for Programmers

To receive updates on Deitel publications, please join the Deitel communities on

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

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

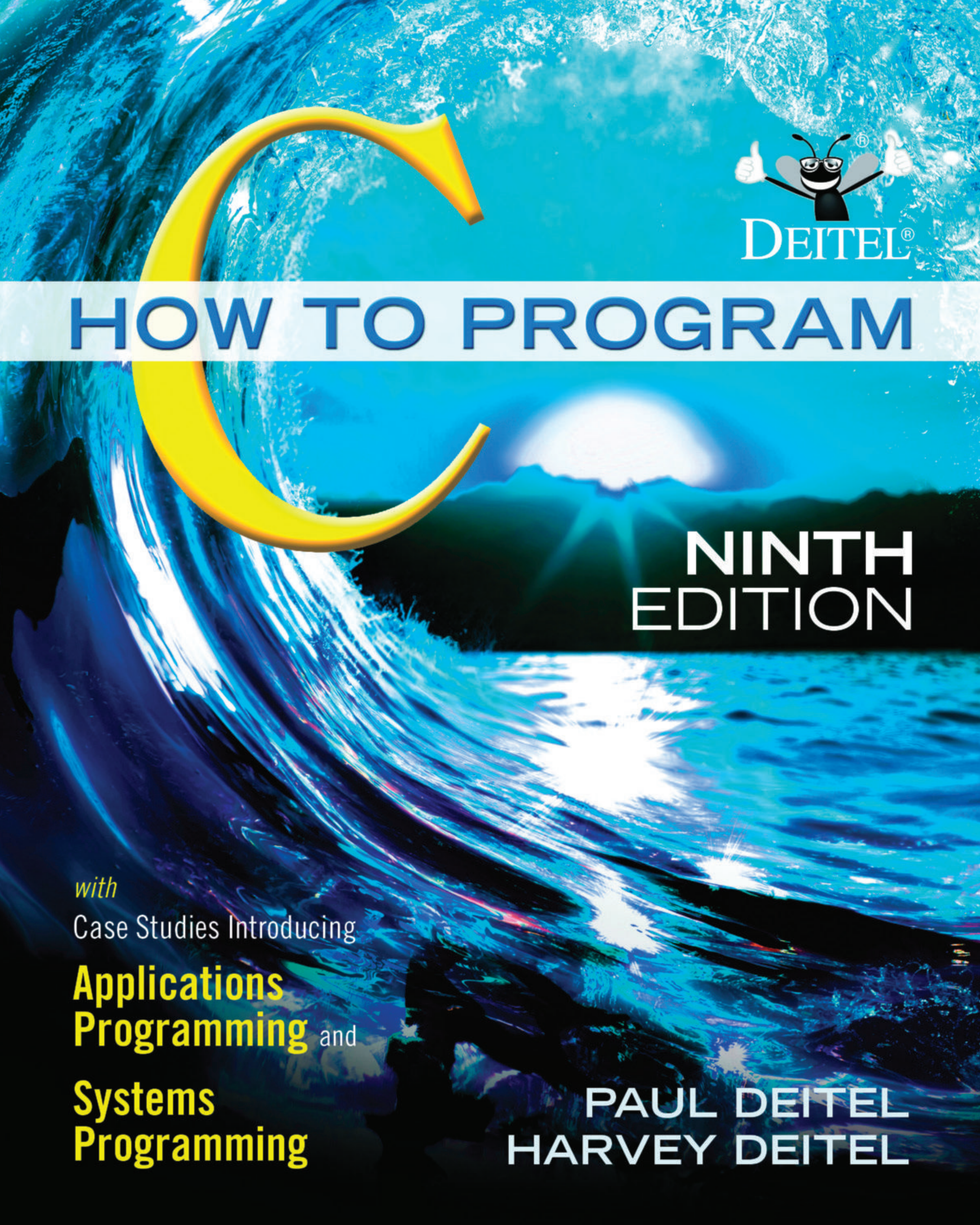
`deitel@deitel.com`

For information on Deitel programming-languages corporate training offered online and on-site worldwide, write to `deitel@deitel.com` or visit:

`https://deitel.com/training/`

For continuing updates on Pearson/Deitel publications visit:

`https://deitel.com`
`https://pearson.com/deitel`



C HOW TO PROGRAM

DEITEL®

NINTH EDITION

with
Case Studies Introducing
Applications Programming and
Systems Programming

PAUL DEITEL
HARVEY DEITEL

Content Development: **Tracy Johnson**
Content Management: **Dawn Murrin**, **Tracy Johnson**
Content Production: **Carole Snyder**
Product Management: **Holly Stark**
Product Marketing: **Wayne Stevens**
Rights and Permissions: **Anjali Singh**

Please contact `https://support.pearson.com/getsupport/s/` with any queries on this content.

Copyright © 2022 by Pearson Education, Inc. or its affiliates, 221 River Street, Hoboken, NJ 07030. 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 otherwise. For information regarding permissions, request forms, and the appropriate contacts within the Pearson Education Global Rights and Permissions department, please visit `https://www.pearsoned.com/permissions/`.

PEARSON, ALWAYS LEARNING, and REVEL are exclusive trademarks owned by Pearson Education, Inc. or its affiliates in the U.S. and/or other countries. Unless otherwise indicated herein, any third-party trademarks, logos, or icons that may appear in this work are the property of their respective owners, and any references to third-party trademarks, logos, icons, or other trade dress are for demonstrative or descriptive purposes only. Such references are not intended to imply any sponsorship, endorsement, authorization, or promotion of Pearson's products by the owners of such marks, or any relationship between the owner and Pearson Education, Inc., or its affiliates, authors, licensees, or distributors. Library of Congress Cataloging-in-Publication Data

Library of Congress Cataloging-in-Publication Data
On file

ScoutAutomatedPrintCode

Pearson

```
ISBN-10:     0-13-540467-3
ISBN-13: 978-0-13-739839-3
```

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

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

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

Linux is a registered trademark of Linus Torvalds.

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 D–G are PDF documents posted online at the book's Companion Website (located at `https://www.pearson.com/deitel`).

Preface xix

Before You Begin li

1 Introduction to Computers and C 1

1.1 Introduction 2
1.2 Hardware and Software 4
 1.2.1 Moore's Law 4
 1.2.2 Computer Organization 5
1.3 Data Hierarchy 8
1.4 Machine Languages, Assembly Languages and High-Level Languages 11
1.5 Operating Systems 13
1.6 The C Programming Language 16
1.7 The C Standard Library and Open-Source Libraries 18
1.8 Other Popular Programming Languages 19
1.9 Typical C Program-Development Environment 21
 1.9.1 Phase 1: Creating a Program 21
 1.9.2 Phases 2 and 3: Preprocessing and Compiling a C Program 21
 1.9.3 Phase 4: Linking 22
 1.9.4 Phase 5: Loading 23
 1.9.5 Phase 6: Execution 23
 1.9.6 Problems That May Occur at Execution Time 23
 1.9.7 Standard Input, Standard Output and Standard Error Streams 24
1.10 Test-Driving a C Application in Windows, Linux and macOS 24
 1.10.1 Compiling and Running a C Application with Visual Studio 2019 Community Edition on Windows 10 25
 1.10.2 Compiling and Running a C Application with Xcode on macOS 29

1.10.3 Compiling and Running a C Application with GNU `gcc` on Linux 32
1.10.4 Compiling and Running a C Application in a GCC Docker Container Running Natively over Windows 10, macOS or Linux 34
1.11 Internet, World Wide Web, the Cloud and IoT 35
1.11.1 The Internet: A Network of Networks 36
1.11.2 The World Wide Web: Making the Internet User-Friendly 37
1.11.3 The Cloud 37
1.11.4 The Internet of Things 38
1.12 Software Technologies 39
1.13 How Big Is Big Data? 39
1.13.1 Big-Data Analytics 45
1.13.2 Data Science and Big Data Are Making a Difference: Use Cases 46
1.14 Case Study—A Big-Data Mobile Application 47
1.15 AI—at the Intersection of Computer Science and Data Science 48

2 Intro to C Programming 55

2.1 Introduction 56
2.2 A Simple C Program: Printing a Line of Text 56
2.3 Another Simple C Program: Adding Two Integers 60
2.4 Memory Concepts 64
2.5 Arithmetic in C 65
2.6 Decision Making: Equality and Relational Operators 69
2.7 Secure C Programming 73

3 Structured Program Development 85

3.1 Introduction 86
3.2 Algorithms 86
3.3 Pseudocode 87
3.4 Control Structures 88
3.5 The `if` Selection Statement 90
3.6 The `if...else` Selection Statement 92
3.7 The `while` Iteration Statement 96
3.8 Formulating Algorithms Case Study 1: Counter-Controlled Iteration 97
3.9 Formulating Algorithms with Top-Down, Stepwise Refinement Case Study 2: Sentinel-Controlled Iteration 99
3.10 Formulating Algorithms with Top-Down, Stepwise Refinement Case Study 3: Nested Control Statements 106
3.11 Assignment Operators 110
3.12 Increment and Decrement Operators 111
3.13 Secure C Programming 114

4 Program Control **133**
4.1 Introduction 134
4.2 Iteration Essentials 134
4.3 Counter-Controlled Iteration 135
4.4 `for` Iteration Statement 136
4.5 Examples Using the `for` Statement 140
4.6 `switch` Multiple-Selection Statement 144
4.7 `do…while` Iteration Statement 150
4.8 `break` and `continue` Statements 151
4.9 Logical Operators 153
4.10 Confusing Equality (==) and Assignment (=) Operators 157
4.11 Structured-Programming Summary 158
4.12 Secure C Programming 163

5 Functions **179**
5.1 Introduction 180
5.2 Modularizing Programs in C 180
5.3 Math Library Functions 182
5.4 Functions 183
5.5 Function Definitions 184
5.5.1 `square` Function 184
5.5.2 `maximum` Function 187
5.6 Function Prototypes: A Deeper Look 188
5.7 Function-Call Stack and Stack Frames 191
5.8 Headers 195
5.9 Passing Arguments by Value and by Reference 197
5.10 Random-Number Generation 197
5.11 Random-Number Simulation Case Study: Building a Casino Game 202
5.12 Storage Classes 207
5.13 Scope Rules 209
5.14 Recursion 212
5.15 Example Using Recursion: Fibonacci Series 216
5.16 Recursion vs. Iteration 219
5.17 Secure C Programming—Secure Random-Number Generation 222
Random-Number Simulation Case Study: The Tortoise and the Hare 241

6 Arrays **243**
6.1 Introduction 244
6.2 Arrays 244
6.3 Defining Arrays 246
6.4 Array Examples 246

6.4.1 Defining an Array and Using a Loop to Set the Array's Element Values 247
6.4.2 Initializing an Array in a Definition with an Initializer List 248
6.4.3 Specifying an Array's Size with a Symbolic Constant and Initializing Array Elements with Calculations 249
6.4.4 Summing the Elements of an Array 250
6.4.5 Using Arrays to Summarize Survey Results 250
6.4.6 Graphing Array Element Values with Bar Charts 252
6.4.7 Rolling a Die 60,000,000 Times and Summarizing the Results in an Array 253
6.5 Using Character Arrays to Store and Manipulate Strings 255
6.5.1 Initializing a Character Array with a String 255
6.5.2 Initializing a Character Array with an Initializer List of Characters 255
6.5.3 Accessing the Characters in a String 255
6.5.4 Inputting into a Character Array 255
6.5.5 Outputting a Character Array That Represents a String 256
6.5.6 Demonstrating Character Arrays 256
6.6 Static Local Arrays and Automatic Local Arrays 258
6.7 Passing Arrays to Functions 260
6.8 Sorting Arrays 264
6.9 Intro to Data Science Case Study: Survey Data Analysis 267
6.10 Searching Arrays 272
6.10.1 Searching an Array with Linear Search 272
6.10.2 Searching an Array with Binary Search 274
6.11 Multidimensional Arrays 278
6.11.1 Illustrating a Two-Dimensional Array 278
6.11.2 Initializing a Double-Subscripted Array 279
6.11.3 Setting the Elements in One Row 281
6.11.4 Totaling the Elements in a Two-Dimensional Array 281
6.11.5 Two-Dimensional Array Manipulations 281
6.12 Variable-Length Arrays 285
6.13 Secure C Programming 289

7 Pointers 309
7.1 Introduction 310
7.2 Pointer Variable Definitions and Initialization 311
7.3 Pointer Operators 312
7.4 Passing Arguments to Functions by Reference 315
7.5 Using the `const` Qualifier with Pointers 319
7.5.1 Converting a String to Uppercase Using a Non-Constant Pointer to Non-Constant Data 320

7.5.2 Printing a String One Character at a Time Using a Non-Constant Pointer to Constant Data 320
7.5.3 Attempting to Modify a Constant Pointer to Non-Constant Data 322
7.5.4 Attempting to Modify a Constant Pointer to Constant Data 323
7.6 Bubble Sort Using Pass-By-Reference 324
7.7 `sizeof` Operator 328
7.8 Pointer Expressions and Pointer Arithmetic 330
7.8.1 Pointer Arithmetic Operators 331
7.8.2 Aiming a Pointer at an Array 331
7.8.3 Adding an Integer to a Pointer 331
7.8.4 Subtracting an Integer from a Pointer 332
7.8.5 Incrementing and Decrementing a Pointer 332
7.8.6 Subtracting One Pointer from Another 332
7.8.7 Assigning Pointers to One Another 332
7.8.8 Pointer to `void` 332
7.8.9 Comparing Pointers 333
7.9 Relationship between Pointers and Arrays 333
7.9.1 Pointer/Offset Notation 333
7.9.2 Pointer/Subscript Notation 334
7.9.3 Cannot Modify an Array Name with Pointer Arithmetic 334
7.9.4 Demonstrating Pointer Subscripting and Offsets 334
7.9.5 String Copying with Arrays and Pointers 336
7.10 Arrays of Pointers 338
7.11 Random-Number Simulation Case Study: Card Shuffling and Dealing 339
7.12 Function Pointers 344
7.12.1 Sorting in Ascending or Descending Order 344
7.12.2 Using Function Pointers to Create a Menu-Driven System 347
7.13 Secure C Programming 349
Special Section: Building Your Own Computer as a Virtual Machine 362
Special Section—Embedded Systems Programming Case Study: Robotics with the Webots Simulator 369

8 Characters and Strings 387
8.1 Introduction 388
8.2 Fundamentals of Strings and Characters 388
8.3 Character-Handling Library 390
8.3.1 Functions `isdigit`, `isalpha`, `isalnum` and `isxdigit` 391
8.3.2 Functions `islower`, `isupper`, `tolower` and `toupper` 393
8.3.3 Functions `isspace`, `iscntrl`, `ispunct`, `isprint` and `isgraph` 394
8.4 String-Conversion Functions 396
8.4.1 Function `strtod` 396

8.4.2 Function `strtol` 397
8.4.3 Function `strtoul` 398
8.5 Standard Input/Output Library Functions 399
8.5.1 Functions `fgets` and `putchar` 399
8.5.2 Function `getchar` 401
8.5.3 Function `sprintf` 401
8.5.4 Function `sscanf` 402
8.6 String-Manipulation Functions of the String-Handling Library 403
8.6.1 Functions `strcpy` and `strncpy` 404
8.6.2 Functions `strcat` and `strncat` 405
8.7 Comparison Functions of the String-Handling Library 406
8.8 Search Functions of the String-Handling Library 408
8.8.1 Function `strchr` 409
8.8.2 Function `strcspn` 410
8.8.3 Function `strpbrk` 410
8.8.4 Function `strrchr` 411
8.8.5 Function `strspn` 411
8.8.6 Function `strstr` 412
8.8.7 Function `strtok` 413
8.9 Memory Functions of the String-Handling Library 414
8.9.1 Function `memcpy` 415
8.9.2 Function `memmove` 416
8.9.3 Function `memcmp` 416
8.9.4 Function `memchr` 417
8.9.5 Function `memset` 417
8.10 Other Functions of the String-Handling Library 419
8.10.1 Function `strerror` 419
8.10.2 Function `strlen` 419
8.11 Secure C Programming 420
Pqyoaf X Nylfomigrob Qwbbfmh Mndogvk: Rboqlrut yua Boklnxhmywex **434**
Secure C Programming Case Study: Public-Key Cryptography **440**

9 Formatted Input/Output 449
9.1 Introduction 450
9.2 Streams 450
9.3 Formatting Output with `printf` 451
9.4 Printing Integers 452
9.5 Printing Floating-Point Numbers 453
9.5.1 Conversion Specifiers `e`, `E` and `f` 454
9.5.2 Conversion Specifiers `g` and `G` 454
9.5.3 Demonstrating Floating-Point Conversion Specifiers 455
9.6 Printing Strings and Characters 456

9.7 Other Conversion Specifiers 457
9.8 Printing with Field Widths and Precision 458
9.8.1 Field Widths for Integers 458
9.8.2 Precisions for Integers, Floating-Point Numbers and Strings 459
9.8.3 Combining Field Widths and Precisions 460
9.9 `printf` Format Flags 461
9.9.1 Right- and Left-Alignment 461
9.9.2 Printing Positive and Negative Numbers with and without the + Flag 462
9.9.3 Using the Space Flag 462
9.9.4 Using the `#` Flag 463
9.9.5 Using the `0` Flag 463
9.10 Printing Literals and Escape Sequences 464
9.11 Formatted Input with `scanf` 465
9.11.1 `scanf` Syntax 466
9.11.2 `scanf` Conversion Specifiers 466
9.11.3 Reading Integers 467
9.11.4 Reading Floating-Point Numbers 468
9.11.5 Reading Characters and Strings 468
9.11.6 Using Scan Sets 469
9.11.7 Using Field Widths 470
9.11.8 Skipping Characters in an Input Stream 471
9.12 Secure C Programming 472

10 Structures, Unions, Bit Manipulation and Enumerations 481

10.1 Introduction 482
10.2 Structure Definitions 483
10.2.1 Self-Referential Structures 483
10.2.2 Defining Variables of Structure Types 484
10.2.3 Structure Tag Names 484
10.2.4 Operations That Can Be Performed on Structures 484
10.3 Initializing Structures 486
10.4 Accessing Structure Members with `.` and `->` 486
10.5 Using Structures with Functions 488
10.6 `typedef` 488
10.7 Random-Number Simulation Case Study: High-Performance Card Shuffling and Dealing 489
10.8 Unions 492
10.8.1 `union` Declarations 493
10.8.2 Allowed `union`s Operations 493
10.8.3 Initializing `union`s in Declarations 493
10.8.4 Demonstrating `union`s 494

10.9 Bitwise Operators 495
- 10.9.1 Displaying an Unsigned Integer's Bits 496
- 10.9.2 Making Function `displayBits` More Generic and Portable 497
- 10.9.3 Using the Bitwise AND, Inclusive OR, Exclusive OR and Complement Operators 498
- 10.9.4 Using the Bitwise Left- and Right-Shift Operators 501
- 10.9.5 Bitwise Assignment Operators 503

10.10 Bit Fields 504
- 10.10.1 Defining Bit Fields 504
- 10.10.2 Using Bit Fields to Represent a Card's Face, Suit and Color 505
- 10.10.3 Unnamed Bit Fields 507

10.11 Enumeration Constants 507

10.12 Anonymous Structures and Unions 509

10.13 Secure C Programming 510

Special Section: Raylib Game-Programming Case Studies 520

Game-Programming Case Study Exercise: SpotOn Game 526

Game-Programming Case Study: Cannon Game 527

Visualization with raylib—Law of Large Numbers Animation 529

Case Study: The Tortoise and the Hare with raylib—a Multimedia "Extravaganza" 531

Random-Number Simulation Case Study: High-Performance Card Shuffling and Dealing with Card Images and raylib 533

11 File Processing 539

11.1 Introduction 540

11.2 Files and Streams 540

11.3 Creating a Sequential-Access File 542
- 11.3.1 Pointer to a `FILE` 543
- 11.3.2 Using `fopen` to Open a File 543
- 11.3.3 Using `feof` to Check for the End-of-File Indicator 543
- 11.3.4 Using `fprintf` to Write to a File 544
- 11.3.5 Using `fclose` to Close a File 544
- 11.3.6 File-Open Modes 545

11.4 Reading Data from a Sequential-Access File 547
- 11.4.1 Resetting the File Position Pointer 548
- 11.4.2 Credit Inquiry Program 548

11.5 Random-Access Files 552

11.6 Creating a Random-Access File 553

11.7 Writing Data Randomly to a Random-Access File 555
- 11.7.1 Positioning the File Position Pointer with `fseek` 557
- 11.7.2 Error Checking 558

11.8 Reading Data from a Random-Access File 558

11.9 Case Study: Transaction-Processing System 560
11.10 Secure C Programming 566
AI Case Study: Intro to NLP—Who Wrote Shakespeare's Works? 576
AI/Data-Science Case Study—Machine Learning with GNU Scientific Library 582
AI/Data-Science Case Study: Time Series and Simple Linear Regression 588
Web Services and the Cloud Case Study—libcurl and OpenWeatherMap 589

12 Data Structures 595

12.1 Introduction 596
12.2 Self-Referential Structures 597
12.3 Dynamic Memory Management 598
12.4 Linked Lists 599
12.4.1 Function `insert` 603
12.4.2 Function `delete` 605
12.4.3 Functions `isEmpty` and `printList` 607
12.5 Stacks 608
12.5.1 Function `push` 612
12.5.2 Function `pop` 613
12.5.3 Applications of Stacks 613
12.6 Queues 614
12.6.1 Function `enqueue` 619
12.6.2 Function `dequeue` 620
12.7 Trees 621
12.7.1 Function `insertNode` 624
12.7.2 Traversals: Functions `inOrder`, `preOrder` and `postOrder` 625
12.7.3 Duplicate Elimination 626
12.7.4 Binary Tree Search 626
12.7.5 Other Binary Tree Operations 626
12.8 Secure C Programming 627
Special Section: Systems Software Case Study—Building Your Own Compiler 636

13 Computer-Science Thinking: Sorting Algorithms and Big O 657

13.1 Introduction 658
13.2 Efficiency of Algorithms: Big O 659
13.2.1 $O(1)$ Algorithms 659
13.2.2 $O(n)$ Algorithms 659
13.2.3 $O(n^2)$ Algorithms 659

13.3 Selection Sort 660
13.3.1 Selection Sort Implementation 661
13.3.2 Efficiency of Selection Sort 664
13.4 Insertion Sort 665
13.4.1 Insertion Sort Implementation 665
13.4.2 Efficiency of Insertion Sort 668
13.5 Case Study: Visualizing the High-Performance Merge Sort 668
13.5.1 Merge Sort Implementation 669
13.5.2 Efficiency of Merge Sort 673
13.5.3 Summarizing Various Algorithms' Big O Notations 674

14 Preprocessor 681
14.1 Introduction 682
14.2 `#include` Preprocessor Directive 683
14.3 `#define` Preprocessor Directive: Symbolic Constants 683
14.4 `#define` Preprocessor Directive: Macros 684
14.4.1 Macro with One Argument 685
14.4.2 Macro with Two Arguments 686
14.4.3 Macro Continuation Character 686
14.4.4 `#undef` Preprocessor Directive 686
14.4.5 Standard-Library Macros 686
14.4.6 Do Not Place Expressions with Side Effects in Macros 687
14.5 Conditional Compilation 687
14.5.1 `#if...#endif` Preprocessor Directive 687
14.5.2 Commenting Out Blocks of Code with `#if...#endif` 688
14.5.3 Conditionally Compiling Debug Code 688
14.6 `#error` and `#pragma` Preprocessor Directives 689
14.7 `#` and `##` Operators 690
14.8 Line Numbers 690
14.9 Predefined Symbolic Constants 691
14.10 Assertions 691
14.11 Secure C Programming 692

15 Other Topics 699
15.1 Introduction 700
15.2 Variable-Length Argument Lists 700
15.3 Using Command-Line Arguments 702
15.4 Compiling Multiple-Source-File Programs 704
15.4.1 `extern` Declarations for Global Variables in Other Files 704
15.4.2 Function Prototypes 705
15.4.3 Restricting Scope with `static` 705
15.5 Program Termination with `exit` and `atexit` 706
15.6 Suffixes for Integer and Floating-Point Literals 708

15.7 Signal Handling 708
15.8 Dynamic Memory Allocation Functions `calloc` and `realloc` 711
15.9 `goto`: Unconditional Branching 713

A Operator Precedence Chart 719

B ASCII Character Set 721

C Multithreading/Multicore and Other C18/C11/C99 Topics 723

C.1 Introduction 724
C.2 Headers Added in C99 725
C.3 Designated Initializers and Compound Literals 725
C.4 Type `bool` 727
C.5 Complex Numbers 728
C.6 Macros with Variable-Length Argument Lists 730
C.7 Other C99 Features 730
- C.7.1 Compiler Minimum Resource Limits 730
- C.7.2 The `restrict` Keyword 730
- C.7.3 Reliable Integer Division 731
- C.7.4 Flexible Array Members 731
- C.7.5 Type-Generic Math 732
- C.7.6 Inline Functions 732
- C.7.7 `__func__` Predefined Identifier 732
- C.7.8 `va_copy` Macro 733

C.8 C11/C18 Features 733
- C.8.1 C11/C18 Headers 733
- C.8.2 `quick_exit` Function 733
- C.8.3 Unicode® Support 733
- C.8.4 `_Noreturn` Function Specifier 734
- C.8.5 Type-Generic Expressions 734
- C.8.6 Annex L: Analyzability and Undefined Behavior 734
- C.8.7 Memory Alignment Control 735
- C.8.8 Static Assertions 735
- C.8.9 Floating-Point Types 735

C.9 Case Study: Performance with Multithreading and Multicore Systems 736
- C.9.1 Example: Sequential Execution of Two Compute-Intensive Tasks 739
- C.9.2 Example: Multithreaded Execution of Two Compute-Intensive Tasks 741
- C.9.3 Other Multithreading Features 745

D Intro to Object-Oriented Programming Concepts 747

D.1 Introduction 747
D.2 Object-Oriented Programming Languages 747
D.3 Automobile as an Object 748
D.4 Methods and Classes 748
D.5 Instantiation 748
D.6 Reuse 748
D.7 Messages and Method Calls 749
D.8 Attributes and Instance Variables 749
D.9 Inheritance 749
D.10 Object-Oriented Analysis and Design (OOAD) 750

Index 751

Online Appendices

D Number Systems

E Using the Visual Studio Debugger

F Using the GNU gdb Debugger

G Using the Xcode Debugger

Preface

An Innovative C Programming Textbook for the 2020s

Good programmers write code that humans can understand.[1]
—Martin Fowler

I think that it's extraordinarily important that we in computer science keep fun in computing.[2]
—Alan Perlis

Welcome to *C How to Program, Ninth Edition*. We present a friendly, contemporary, code-intensive, case-study-oriented introduction to C—which is among the world's most popular programming languages.[3] Whether you're a student, an instructor or a professional programmer, this book has much to offer you. In this Preface, we present the "soul of the book."

At the heart of the book is the Deitel signature **live-code approach**—we generally present concepts in the context of **147 complete, working, real-world C programs**, rather than in code snippets. We follow each code example with one or more live program input/output dialogs. All the code is provided free for download at

```
https://deitel.com/c-how-to-program-9-e
https://pearson.com/deitel
```

You should execute each program in parallel with reading the text, making your learning experience "come alive."

For many decades:

- computer hardware has rapidly been getting faster, cheaper and smaller,
- Internet bandwidth (that is, its information-carrying capacity) has rapidly been getting larger and cheaper, and
- quality computer software has become ever more abundant and often free or nearly free through the **open-source movement**.

1. Martin Fowler (with contributions by Kent Beck). *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, 1999. p. 15.
2. Alan Perlis, Quoted in the book dedication of *The Structure and Interpretation of Computer Programs, 2/e* by Hal Abelson, Gerald Jay Sussman and Julie Sussman. McGraw-Hill. 1996.
3. Tiobe Index for November 2020. Accessed November 9, 2020. `https://www.tiobe.com/tiobe-index/`.

We'll say lots more about these important trends. The **Internet of Things** (**IoT**) is already connecting tens of billions of computerized devices of every imaginable type. These generate enormous volumes of data (one form of "**big data**") at rapidly increasing speeds and quantities. And most computing will eventually be performed online in "the **Cloud**"—that is, by using computing services accessible over the Internet.

For the novice, the book's early chapters establish a solid foundation in programming fundamentals. The mid-range to high-end chapters and the 20+ case studies ease novices into the world of professional software-development challenges and practices.

Given the extraordinary performance demands that today's applications place on computer hardware, software and the Internet, professionals often choose C to build the most performance-intensive portions of these applications. Throughout the book, we emphasize performance issues to help you prepare for industry.

The book's modular architecture (see the chart on the inside front cover) makes it appropriate for several audiences:

- **Introductory and intermediate college programming courses** in Computer Science, Computer Engineering, Information Systems, Information Technology, Software Engineering and related disciplines.
- **Science, technology, engineering and math** (STEM) **college courses** with a programming component.
- **Professional industry training courses**.
- **Experienced professionals** learning C to prepare for upcoming software-development projects.

We've enjoyed writing nine editions of this book over the last 29 years. We hope you'll find *C How to Program, 9/e* informative, challenging and entertaining as you prepare to develop leading-edge, high-performance applications and systems in your career.

New and Updated Features in This Ninth Edition

Here, we briefly overview some of this edition's new and updated features. There are many more. The sections that follow provide more details:

- We added a **one-page color Table of Contents chart** on the inside front cover, making it easy for you to see the entire book from "40,000 feet." This chart emphasizes the book's **modular architecture** and lists most of the case studies.
- Some of the case studies are book sections that walk through the complete source code—these are supported by end-of-chapter exercises that might ask you to modify the code presented in the text or take on related challenges. Some are exercises with detailed specifications from which you should be able to develop the code solution on your own. Some are exercises that ask you to visit websites that contain nice tutorials. And some are exercises that ask you to visit developer websites where there may be code to study, but no tutorials—and the code may not be well commented. Instructors will decide which of the case studies are appropriate for their particular audiences.

- We adhere to the **C11/C18 standards**.
- We tested all the code for correctness on the **Windows**, **macOS** and **Linux** operating systems using the latest versions of the **Visual C++**, **Xcode** and **GNU `gcc` compilers**, respectively, noting differences among the platforms. See the **Before You Begin** section that follows this Preface for software installation instructions.
- We used the **`clang-tidy` static code analysis tool** to check all the code in the book's **code examples** for improvement suggestions, from simple items like **ensuring variables are initialized** to **warnings about potential security flaws**. We also ran this tool on the code solutions that we make available to instructors for hundreds of the book's exercises. The complete list of code checks can be found at `https://clang.llvm.org/extra/clang-tidy/checks/list.html`.
- GNU `gcc` tends to be the most compliant C compiler. **To enable macOS and Windows users to use `gcc` if they wish, Chapter 1 includes a test-drive demonstrating how to compile programs and run them using `gcc` in the cross-platform GNU Compiler Collection Docker container.**
- We've added **350+ integrated Self-Check exercises**, each followed immediately by its answer. These are ideal for **self study** and for use in "**flipped classrooms**" (see the "Flipped Classrooms" section later in this Preface).
- To ensure that book content is **topical**, we did extensive Internet research on C specifically and the world of computing in general, which influenced our choice of case studies. We show C as it's intended to be used with a rich collection of applications programming and systems programming case studies, focusing on **computer-science**, **artificial intelligence**, **data science** and other fields. See the "Case Studies" section later in this Preface for more details.
- In the text, code examples, exercises and case studies, we familiarize students with **current topics of interest to developers**, including open-source software, virtualization, simulation, web services, multithreading, multicore hardware architecture, systems programming, game programming, animation, visualization, 2D and 3D graphics, artificial intelligence, natural language processing, machine learning, robotics, data science, secure programming, cryptography, Docker, GitHub, StackOverflow, forums and more.
- We adhere to the latest **ACM/IEEE computing curricula recommendations**, which call for covering security, data science, ethics, privacy and performance concepts and using real-world data throughout the curriculum. See the "Computing and Data Science Curricula" section for more details.
- Most chapters in this book's recent editions end with **Secure C programming sections** that focus on the SEI CERT C Coding Standard from the CERT group of Carnegie Mellon University's Software Engineering Institute (SEI). For this edition, we tuned the SEI CERT-based sections. We also added **security icons in the page margin** whenever we discuss a security-related issue in the text. All of this is consistent with the **ACM/IEEE computing curricula docu-**

SEC

ments' enhanced emphasis on security. See the "Computing and Data Science Curricula" section later in this Preface for a list of the key curricula documents.

- Consistent with our richer treatment of security, we've added case studies on secret-key and public-key cryptography. The latter includes a detailed walk-through of the enormously popular RSA algorithm's steps, providing hints to help you build a working, simple, small-scale implementation.
- We've enhanced existing case studies and added new ones focusing on AI and data science, including simulations with random-number generation, survey data analysis, natural language processing (NLP) and artificial intelligence (machine-learning with simple linear regression). Data science is emphasized in the latest ACM/IEEE computing curricula documents.
- We've added exercises in which students use the Internet to research **ethics** and **privacy** issues in computing.
- We tuned our **mutltithreading and multicore performance** case study. We also show a **performance icon in the margin** whenever we discuss a performance-related issue in the text. PERF
- We integrated the previous edition's hundreds of software-development tips directly into the text for a smoother reading experience. We call out **common errors** and **good software engineering practices** with new margin icons. ERR SE
- We upgraded our appendix on additional sorting algorithms and analysis of algorithms with Big O to full-chapter status (Chapter 13).
- C programmers often subsequently learn one or more C-based object-oriented languages. We added an appendix that presents a friendly intro to object-oriented programming concepts and terminology. C is a procedural programming language, so this appendix will help students appreciate differences in thinking between C developers and the folks who program in languages like C++, Java, C#, Objective-C, Swift and other object-oriented languages. We do lots of things like this in the book to prepare students for industry.
- Several case studies now have you use free open-source libraries and tools.
- We added a case study that performs visualization with gnuplot.
- We removed the previous edition's introduction to C++ programming to make room for the hundreds of integrated self-check exercises and our new applications programming and systems programming case studies.
- This new edition is published in a larger font size and page size for enhanced readability.

A Tour of the Book

The **Table of Contents** graphic on the inside front cover shows the book's **modular architecture**. Instructors can conveniently adapt the content to a variety of courses and audiences. Here we present a brief chapter-by-chapter walkthrough and indicate where the book's case studies are located. Some are in-chapter examples and some are end-of-chapter exercises. Some are fully coded. For others, you'll develop the solution.

Chapters 1–5 are traditional introductory C programming topics. Chapters 6–11 are intermediate topics forming the high end of Computer Science 1 and related courses. Chapters 12–15 are advanced topics for late CS1 or early CS2 courses. Here's a list of the topical, challenging and often entertaining hands-on case studies.

Systems Programming Case Studies

- **Systems Software**—Building Your Own Computer (as a virtual machine)
- **Systems Software**—Building Your Own Compiler
- **Embedded Systems Programming**—Robotics, 3D graphics and animation with the Webots Simulator
- **Performance with Multithreading and Multicore Systems**

Application Programming Case Studies

- **Algorithm Development**—Counter-Controlled Iteration
- **Algorithm Development**—Sentinel-Controlled Iteration
- **Algorithm Development**—Nested Control Statements
- **Random-Number Simulation**—Building a Casino Game
- **Random-Number Simulation**—Card Shuffling and Dealing
- **Random-Number Simulation**—The Tortoise and the Hare Race
- **Intro to Data Science**—Survey Data Analysis
- **Direct-Access File Processing**—Building a Transaction-Processing System
- **Visualizing Searching and Sorting Algorithms**—Binary Search and Merge Sort.
- **Artificial Intelligence/Data Science**—Natural Language Processing ("Who Really Wrote the Works of William Shakespeare?")
- **Artificial Intelligence/Data Science**—Machine Learning with the GNU Scientific Library ("Statistics Can Be Deceiving" and "Have Average January Temperatures in New York City Been Rising Over the Last Century?")
- **Game Programming**—Cannon Game with the raylib Library
- **Game Programming**—SpotOn Game with the raylib Library
- **Multimedia: Audio and Animation**—The Tortoise and the Hare Race with the raylib Library
- **Security and Cryptography**—Implementing a Vigenère Secret-Key Cipher and RSA Public-Key Cryptography
- **Animated Visualization with raylib**—The Law of Large Numbers
- **Web Services and the Cloud**—Getting a Weather Report Using libcurl and the OpenWeatherMap Web Services, and An Introduction to Building Mashups with Web Services.