



DEITEL®

HOW TO PROGRAM

NINTH
EDITION

with
Case Studies Introducing

**Applications
Programming** and

**Systems
Programming**

PAUL DEITEL
HARVEY DEITEL

This page intentionally left blank



DEITEL®

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>



DEITEL®

HOW TO PROGRAM

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



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.

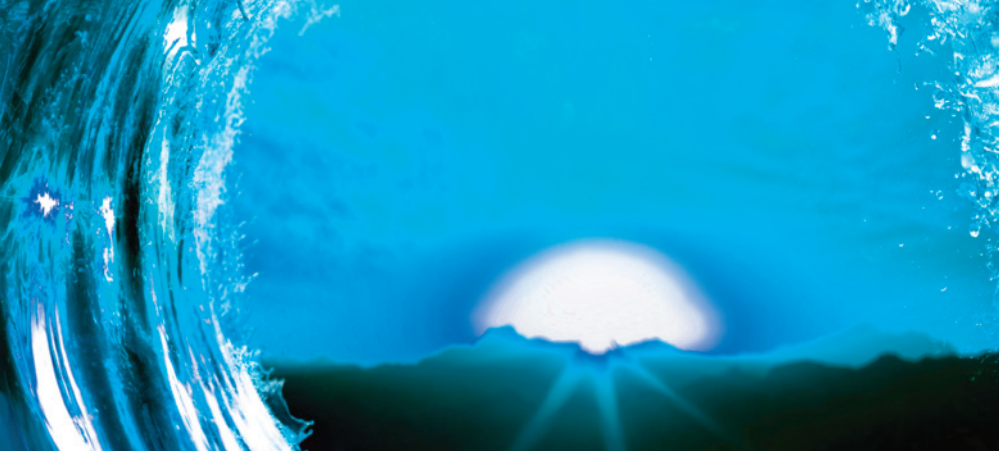
Java is a registered trademark of Oracle and/or its affiliates.

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**

I	Introduction to Computers and C	I
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 <code>if</code> Selection Statement	90
3.6	The <code>if...else</code> Selection Statement	92
3.7	The <code>while</code> 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

x **Contents**

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 <code>strtol</code>	397
8.4.3	Function <code>strtoul</code>	398
8.5	Standard Input/Output Library Functions	399
8.5.1	Functions <code>fgets</code> and <code>putchar</code>	399
8.5.2	Function <code>getchar</code>	401
8.5.3	Function <code>sprintf</code>	401
8.5.4	Function <code>sscanf</code>	402
8.6	String-Manipulation Functions of the String-Handling Library	403
8.6.1	Functions <code>strcpy</code> and <code>strncpy</code>	404
8.6.2	Functions <code>strcat</code> and <code>strncat</code>	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 <code>strchr</code>	409
8.8.2	Function <code>strcspn</code>	410
8.8.3	Function <code>strpbrk</code>	410
8.8.4	Function <code>strrchr</code>	411
8.8.5	Function <code>strspn</code>	411
8.8.6	Function <code>strstr</code>	412
8.8.7	Function <code>strtok</code>	413
8.9	Memory Functions of the String-Handling Library	414
8.9.1	Function <code>memcpy</code>	415
8.9.2	Function <code>memmove</code>	416
8.9.3	Function <code>memcmp</code>	416
8.9.4	Function <code>memchr</code>	417
8.9.5	Function <code>memset</code>	417
8.10	Other Functions of the String-Handling Library	419
8.10.1	Function <code>strerror</code>	419
8.10.2	Function <code>strlen</code>	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 <code>printf</code>	451
9.4	Printing Integers	452
9.5	Printing Floating-Point Numbers	453
9.5.1	Conversion Specifiers <code>e</code> , <code>E</code> and <code>f</code>	454
9.5.2	Conversion Specifiers <code>g</code> and <code>G</code>	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 unions Operations	493
10.8.3	Initializing unions in Declarations	493
10.8.4	Demonstrating unions	494

10.9	Bitwise Operators	495
10.9.1	Displaying an Unsigned Integer’s Bits	496
10.9.2	Making Function <code>displayBits</code> 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 <code>fopen</code> to Open a File	543
11.3.3	Using <code>feof</code> to Check for the End-of-File Indicator	543
11.3.4	Using <code>fprintf</code> to Write to a File	544
11.3.5	Using <code>fclose</code> 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 <code>fseek</code>	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	<code>#include</code> Preprocessor Directive	683
14.3	<code>#define</code> Preprocessor Directive: Symbolic Constants	683
14.4	<code>#define</code> 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	<code>#undef</code> 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	<code>#if...#endif</code> Preprocessor Directive	687
14.5.2	Commenting Out Blocks of Code with <code>#if...#endif</code>	688
14.5.3	Conditionally Compiling Debug Code	688
14.6	<code>#error</code> and <code>#pragma</code> Preprocessor Directives	689
14.7	<code>#</code> and <code>##</code> 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	<code>extern</code> Declarations for Global Variables in Other Files	704
15.4.2	Function Prototypes	705
15.4.3	Restricting Scope with <code>static</code>	705
15.5	Program Termination with <code>exit</code> and <code>atexit</code>	706
15.6	Suffixes for Integer and Floating-Point Literals	708

15.7	Signal Handling	708
15.8	Dynamic Memory Allocation Functions <code>calloc</code> and <code>realloc</code>	711
15.9	<code>goto</code> : 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 <code>bool</code>	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 <code>restrict</code> 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	<code>__func__</code> Predefined Identifier	732
C.7.8	<code>va_copy</code> Macro	733
C.8	C11/C18 Features	733
C.8.1	C11/C18 Headers	733
C.8.2	<code>quick_exit</code> Function	733
C.8.3	Unicode [®] Support	733
C.8.4	<code>_Noreturn</code> 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-**

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 **multithreading and multicore performance** case study. We also show a **performance icon in the margin** whenever we discuss a performance-related issue in the text.
- 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.
- 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.

PERF ERR SE 

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.

Whether you're a student getting a sense of the textbook you'll be using, an instructor planning your course syllabus or a professional software developer deciding which chapters to read as you prepare for a project, the following chapter overviews will help you make the best decisions.

Part I: Programming Fundamentals Quickstart

Chapter 1, Introduction to Computers and C, engages novice students with intriguing facts and figures to excite them about studying computers and computer programming. The chapter includes current technology trends, hardware and software concepts and the data hierarchy from bits to databases. It lays the groundwork for the C programming discussions in Chapters 2–15, the appendices and the integrated case studies.

We discuss the programming-language types and various technologies you're likely to use as you develop software. We introduce the C standard library—existing, reusable, top-quality, high-performance functions to help you avoid “reinventing the wheel.” You'll enhance your productivity by using libraries to perform significant tasks while writing only modest numbers of instructions. We also introduce the **Internet**, the **World Wide Web**, the “**Cloud**” and the **Internet of Things (IoT)**, laying the groundwork for modern applications development.

This chapter's **test-drives** demonstrate how to compile and execute C code with

- **Microsoft's Visual C++** in Visual Studio on Windows,
- **Apple's Xcode** on macOS, and
- **GNU's gcc** on Linux.

We've run the book's 147 code examples using each environment.⁴ Choose whichever program-development environment you prefer—the book works well with others, too.

We also demonstrate **GNU gcc in the GNU Compiler Collection Docker container**. This enables you to run the latest **GNU gcc** compiler on Windows, macOS or Linux—this is important because the GNU compilers generally implement all (or most) features in the latest language standards. See the **Before You Begin** section that follows this Preface for compiler installation instructions. See the **Docker** section later in this Preface for more information on this important developer tool. For Windows users, we point to Microsoft's step-by-step instructions that allow you to install Linux in Windows via the **Windows Subsystem for Linux (WSL)**. This is another way to be able to use the **GNU gcc** compiler on Windows.

You'll learn just how big “**big data**” is and how quickly it's getting even bigger. The chapter closes with an introduction to **artificial intelligence (AI)**—a key overlap between the computer-science and data-science fields. AI and data science are likely to play significant roles in your computing career.

Chapter 2, Intro to C Programming, presents C fundamentals and illustrates key language features, including input, output, fundamental data types, computer memory concepts, arithmetic operators and their precedence, and decision making.

4. We point out the few cases in which a compiler does not support a particular feature.

Chapter 3, Structured Program Development, is one of the most important chapters for programming novices. It focuses on **problem-solving and algorithm development** with C's **control statements**. You'll **develop algorithms through top-down, stepwise refinement**, using the `if` and `if...else` selection statements, the `while` iteration statement for counter-controlled and sentinel-controlled iteration, and the increment, decrement and assignment operators. The chapter presents three algorithm-development case studies.

Chapter 4, Program Control, presents C's other **control statements**—`for`, `do...while`, `switch`, `break` and `continue`—and the logical operators. A key feature of this chapter is its **structured-programming summary**.

Chapter 5, Functions, introduces program construction using existing and custom functions as building blocks. We demonstrate **simulation techniques** with **random-number generation**. We also discuss passing information between functions and how the function-call stack and stack frames support the function call/return mechanism. We begin our treatment of recursion. This chapter also presents our first simulation case study—**Building a Casino Game**, which is enhanced by end-of-chapter exercises.

Part 2: Arrays, Pointers and Strings

Chapter 6, Arrays, presents C's built-in **array data structure** for representing lists and tables of values. You'll define and initialize arrays, and refer to their individual elements. We discuss passing arrays to functions, sorting and searching arrays, manipulating multidimensional arrays and creating variable-length arrays whose size is determined at execution time. **Chapter 13, Computer-Science Thinking: Sorting Algorithms and Big O**, discusses more sophisticated and higher-performance sorting algorithms and presents a friendly introduction to **analysis of algorithms** with computer science's **Big O** notation. Chapter 6 presents our first data-science case study—**Intro to Data Science: Survey Data Analysis**. In the exercises, we also present two **Game Programming with Graphics, Sound and Collision Detection** case studies and an **Embedded Systems Programming** case study (**Robotics with the Webots Simulator**).

Chapter 7, Pointers, presents what is arguably C's most powerful feature. Pointers enable programs to

- accomplish pass-by-reference,
- pass functions to other functions, and
- create and manipulate dynamic data structures, which you'll study in detail in **Chapter 12**.

The chapter explains pointer concepts, such as declaring pointers, initializing pointers, getting the memory address of a variable, dereferencing pointers, pointer arithmetic and the close relationship between arrays and pointers. This chapter presents our first systems software case-study exercise—**Building Your Own Computer with Simulation**. This case study introduces an essential modern computer-architecture topic—**virtual machines**.

Chapter 8, Characters and Strings, introduces the C standard library’s string, character and memory-block processing functions. You’ll use these powerful capabilities in **Chapter 11, File Processing**, as you work through a **natural language processing (NLP)** case study. You’ll see that strings are intimately related to pointers and arrays.

Part 3: Formatted Input/Output, Structures and File Processing

Chapter 9, Formatted Input/Output, discusses the powerful formatting features of functions `scanf` and `printf`. When properly used, these functions **securely** input data from the standard input stream and output data to the standard output stream, respectively.

Chapter 10, Structures, Unions, Bit Manipulation and Enumerations, introduces structures (`structs`) for aggregating related data items into custom types, unions for sharing memory among multiple variables, `typedefs` for creating aliases for previously defined data types, bitwise operators for manipulating the individual bits of integral operands and enumerations for defining sets of named integer constants. Many C programmers go on to study C++ and object-oriented programming. In C++, C’s `structs` evolve into `classes`, which are the “blueprints” C++ programmers use to create objects. C `structs` contain only data. C++ `classes` can contain data *and* functions.

Chapter 11, File Processing, introduces files for long-term data retention, even when the computer is powered off. Such data is said to be “persistent.” The chapter explains how plain-text files and binary files are created, updated and processed. We consider both sequential-access and random-access file processing. In one of our case-study exercises, you’ll read data from a comma-separated value (CSV) file. CSV is one of the most popular file formats in the data-science community. This chapter presents our next case study—**Building a Random-Access Transaction-Processing System**. We use random-access files to simulate the kind of high-speed direct-access capabilities that industrial-strength database-management systems have. This chapter also presents our first artificial-intelligence/data-science case study, which uses **Natural Language Processing (NLP)** techniques to begin investigating the controversial question, “Who really wrote the works of William Shakespeare?” A second artificial-intelligence/data-science case study—**Machine Learning with the GNU Scientific Library**—investigates Anscombe’s Quartet using simple linear regression.⁵ This is a collection of four dramatically different datasets that have identical or nearly identical basic descriptive statistics. It offers a valuable insight for students and developers learning some data-science basics in this computer-science textbook. The case study then asks you to run a simple linear regression on 126 years of New York City average January temperature data to determine if there is a cooling or warming trend.

5. “Anscombe’s Quartet.” Accessed November 13, 2020. https://en.wikipedia.org/wiki/Anscombe%27s_quartet.

Part 4: Algorithms and Data Structures

Chapter 12, Data Structures, uses structs to aggregate related data items into custom types, typedefs to create aliases for previously defined types, and **dynamically linked data structures** that can grow and shrink at execution time—**linked lists**, **stacks**, **queues** and **binary trees**. You can use the techniques you learn to implement other data structures. This chapter also presents our next systems-software case study exercise—**Building Your Own Compiler**. We'll define a simple yet powerful high-level language. You'll write some high-level-language programs that your compiler will compile into the machine language of the computer you built in Chapter 7. The compiler will place its machine-language output into a file. Your computer will **load** the machine language from the file into its memory, execute it and produce appropriate outputs.

Chapter 13, Computer-Science Thinking: Sorting Algorithms and Big O, introduces some classic computer-science topics. We consider several algorithms and compare their processor demands and memory consumption. We present a friendly introduction to computer science's **Big O notation**, which indicates how hard an algorithm may have to work to solve a problem, based on the number of items it must process. The chapter includes the case study **Visualizing the High-Performance Merge Sort**.

Our **recursion** (Chapter 5), **arrays** (Chapter 6), **searching** (Chapter 6), **data structures** (Chapter 12), **sorting** (Chapter 13) and **Big O** (Chapter 13) coverage provides nice content for a C data structures course.

Part 5: Preprocessor and Other Topics

Chapter 14, Preprocessor, discusses additional features of the C preprocessor, such as using `#include` to help manage files in large programs, using `#define` to create macros with and without arguments, using conditional compilation to specify portions of a program that should not always be compiled (e.g., extra code used only during program development), displaying error messages during conditional compilation, and using assertions to test whether expressions' values are correct.

Chapter 15, Other Topics, covers additional C topics, including multithreading support (available in GNU gcc, but not Xcode or Visual C++), variable-length argument lists, command-line arguments, compiling multiple-source-file programs, extern declarations for global variables in other files, function prototypes, restricting scope with `static`, makefiles, program termination with `exit` and `atexit`, suffixes for integer and floating-point literals, signal handling, dynamic memory-allocation functions `calloc` and `realloc` and unconditional branching with `goto`. This chapter presents our final case study—**Performance with Multithreading and Multicore Systems**. This case study demonstrates how to create multithreaded programs that will run faster (and often much faster) on today's multicore computer architectures. This is a nice capstone

case study for a book about C, for which writing high-performance programs is paramount.

Appendices

Appendix A, Operator Precedence Chart, lists C's operators in highest-to-lowest precedence order.

Appendix B, ASCII Character Set, shows characters and their corresponding numeric codes.

Appendix C, Multithreading/Multicore and Other C18/C11/C99 Topics, covers designated initializers, compound literals, type `bool`, complex numbers, additions to the preprocessor, the `restrict` keyword, reliable integer division, flexible array members, relaxed constraints on aggregate initialization, type generic math, inline functions, return without expression, `__func__` predefined identifier, `va_copy` macro, C11 headers, `_Generic` keyword (type generic expressions), `quick_exit` function, Unicode[®] support, `_noreturn` function specifier, type-generic expressions, Annex L: Analyzability and Undefined Behavior, memory-alignment control, static assertions, floating-point types and the `timespec_get` function.

Appendix D, Intro to Object-Oriented Programming Concepts, presents a friendly overview of object-oriented programming terminology and concepts. After learning C, you'll likely also learn one or more C-based object-oriented languages—such as C++, Java, C#, Objective-C or Swift—and use them side-by-side with C.

Online Appendices

Appendix E, Number Systems, introduces the binary, octal, decimal and hexadecimal number systems.

Appendices F–H, Using the Visual Studio Debugger, Using the GNU gdb Debugger and Using the Xcode Debugger, demonstrate how to use our three preferred compilers' basic debugging capabilities to locate and correct execution-time problems in your programs.

C How to Program, 9/e Key Features

C Programming Fundamentals

In our rich coverage of C fundamentals:

- We emphasize **problem-solving** and **algorithm development**.
- To help students prepare to work in industry, we use the terminology from the latest **C standard documents** in preference to general programming terms.
- We **avoid heavy math**, leaving it to upper-level courses. **Optional mathematical exercises** are included for science and engineering courses.

C11 and C18 Standards

C11 refined and expanded C’s capabilities. We’ve added more features from the C11 standard. Since C11, there has been only one new version, C18.⁶ It “addressed defects in C11 without introducing new language features.”⁷

Innovation: “Intro-to” Pedagogy with 350+ Integrated Self-Check Exercises

This book uses our new “Intro to” pedagogy with integrated Self Checks and answers. We introduced this pedagogy in our recent textbook, *Intro to Python for Computer Science and Data Science: Learning to Program with AI, Big Data and the Cloud*.

- Chapter sections are intentionally small. We use a “**read-a-little, do-a-little, test-a-little**” approach. You read about a new concept, study and execute the corresponding code examples, then test your understanding of the new concept via the integrated **Self-Check exercises immediately followed by their answers**. This will help you keep a brisk learning pace.
- **Fill-in-the-blank, true/false and discussion Self Checks** enable you to test your understanding of the concepts and terminology you’ve just studied.
- **Code-based Self Checks** give you a chance to use the terminology and reinforce the programming techniques you’ve just studied.
- The **Self-Checks** are particularly valuable for **flipped classroom** courses—we’ll soon say more about that popular educational phenomenon.

KIS (Keep It Simple), KIS (Keep it Small), KIT (Keep it Topical)

- **Keep it simple**—We strive for **simplicity and clarity**.
- **Keep it small**—Many of the book’s examples are small. We use more substantial code examples, exercises and projects when appropriate, particularly in the case studies that are a core feature of this textbook.
- **Keep it topical**—“*Who dares to teach must never cease to learn.*”⁸ (J. C. Dana)—In our research, we browsed, read or watched thousands of current articles, research papers, white papers, books, videos, webinars, blog posts, forum posts, documentation pieces and more.

6. ISO/IEC 9899:2018, Information technology — Programming languages — C, <https://www.iso.org/standard/74528.html>.

7. [https://en.wikipedia.org/wiki/C18_\(C_standard_revision\)](https://en.wikipedia.org/wiki/C18_(C_standard_revision)). Also http://www.iso-9899.info/wiki/The_Standard.

8. John Cotton Dana. From <https://www.bartleby.com/73/1799.html>: “In 1912 Dana, a Newark, New Jersey, librarian, was asked to supply a Latin quotation suitable for inscription on a new building at Newark State College (now Kean University), Union, New Jersey. Unable to find an appropriate quotation, Dana composed what became the college motto.—*The New York Times Book Review*, March 5, 1967, p. 55.”

Hundreds of Contemporary Examples, Exercises and Projects (EEPs)

You'll use a **hands-on applied approach** to learn from a broad selection of real-world **examples, exercises and projects (EEPs)** drawn from computer science, data science and other fields:

- You'll attack exciting and entertaining challenges in our larger case studies, such as building a casino game, building a survey-data-analysis program, building a transaction-processing system, building your own computer (using simulation to build a **virtual machine**), using **AI/data-science** technologies such as **natural language processing** and **machine learning**, building your own compiler, programming computer games, programming **robotics simulations** with **Webots**, and writing multithreaded code to take advantage of today's multicore computer architectures to get the best performance from your computer.
- **Research and project exercises** encourage you to go deeper into what you've learned and explore other technologies. We encourage you to use computers and the Internet to solve significant problems. Projects are often more extensive in scope than the exercises—some might require days or weeks of implementation effort. Many of these are appropriate for **class projects, term projects, directed-study projects, capstone-course projects** and **thesis research**. **We do not provide solutions to the projects.**
- Instructors can tailor their courses to their audience's unique requirements and vary labs and exam questions each semester.

Working with Open-Source Software

*In those days [batch processing] programmers never even documented their programs, because it was assumed that nobody else would ever use them. Now, however, time-sharing had made exchanging software trivial: you just stored one copy in the public repository and thereby effectively gave it to the world. Immediately people began to document their programs and to think of them as being usable by others. They started to build on each other's work.*⁹

—Robert Fano, Founding Director of MIT's Project MAC in the 1960s, which evolved into today's Computer Science and Artificial Intelligence Laboratory (CSAIL)¹⁰

Open source is software with source code that anyone can inspect, modify, and enhance."¹¹ We encourage you to try lots of **demos** and view free, **open-source** code examples (available on sites such as **GitHub**) for inspiration. We say more about GitHub in the section "Thinking Like a Developer—GitHub, StackOverflow and More."

9. Robert Fano, quoted in *Dream Machine: J.C.R. Licklider and the Revolution That Made Computing Personal* by Mitchell Waldrop. Penguin Putnam, 2002. p. 232.

10. "MIT Computer Science and Artificial Intelligence Laboratory." Accessed November 9, 2020. https://en.wikipedia.org/wiki/MIT_Computer_Science_and_Artificial_Intelligence_Laboratory.

11. "What is open source?" Accessed November 14, 2020. <https://opensource.com/resources/what-open-source>.

Visualizations

We include high-level **visualizations** produced with the **gnuplot** open-source visualization package to reinforce your understanding of the concepts:

- We use **visualizations** as a pedagogic tool. For instance, one example makes the **law of large numbers** “come alive” in a **dice-rolling simulation** (see **Chapter 10—Raylib Game Programming Case Studies** later in this Preface). As this program performs increasing numbers of die rolls, you’ll see each of the six faces’ (1, 2, 3, 4, 5, 6) percentage of the total rolls gradually approach 16.667% (1/6th), and the lengths of the bars representing the percentages equalize.
- You should experiment with the code to implement your own visualizations.

Data Experiences

In the book’s examples, exercises and projects—especially in the file-processing chapter—you’ll work with **real-world data** such as Shakespeare’s play *Romeo and Juliet*. You’ll download and analyze text from Project Gutenberg—a great source of free downloadable texts for analysis. The site contains nearly 63,000 e-books in various formats, including plain-text files—these are out of copyright in the United States. You’ll also work with real-world temperature data. In particular, you’ll analyze 126 years of New York City average January temperature data and determine whether there is a cooling or warming trend. You’ll get this data from National Oceanic and Atmospheric Administration (NOAA) website `noaa.gov`.

Thinking Like a Developer—GitHub, StackOverflow and More

*The best way to prepare [to be a programmer] is to write programs, and to study great programs that other people have written. In my case, I went to the garbage cans at the Computer Science Center and fished out listings of their operating systems.*¹²

—William Gates

- To help prepare for your career, you’ll work with such popular developer websites as **GitHub** and **StackOverflow**, and you’ll do Internet research.
- **StackOverflow** is one of the most popular developer-oriented, question-and-answer sites.
- There is a massive C open-source community. For example, on **GitHub**, there are over 32,000¹³ C code repositories! You can check out other people’s C code on GitHub and even build upon it if you like. This is a great way to learn and is a natural extension of our live-code teaching philosophy.¹⁴
- **GitHub** is an excellent venue for **finding free, open-source code** to incorporate into your projects—and for you to contribute your code to the **open-**

12. William Gates, quoted in *Programmers at Work: Interviews With 19 Programmers Who Shaped the Computer Industry* by Susan Lammers. Microsoft Press, 1986, p. 83.

13. “C.” Accessed January 4, 2021. <https://github.com/topics/c>.

14. Students will need to become familiar with the variety of open-source licenses for software on GitHub.

source community if you like. Fifty million developers use GitHub.¹⁵ The site currently hosts over 100 million repositories for code written in an enormous number of languages¹⁶—developers contributed to 44+ million repositories in 2019 alone.¹⁷ **GitHub** is a crucial element of the professional software developer’s arsenal with **version control tools** that help teams of developers manage public open-source projects and private projects.

- In 2018, Microsoft purchased **GitHub** for \$7.5 billion. If you become a software developer, you’ll almost certainly use GitHub regularly. According to Microsoft’s CEO, Satya Nadella, they bought GitHub to “empower every developer to build, innovate and solve the world’s most pressing challenges.”¹⁸
- We encourage you to study and execute lots of developers’ open-source C code on GitHub.

Privacy

The ACM/IEEE’s curricula recommendations for Computer Science, Information Technology and Cybersecurity **mention privacy over 200 times**. Every programming student and professional needs to be acutely aware of privacy issues and concerns. Students research privacy in four exercises in Chapters 1, 3 and 10.

In Chapter 1’s exercises, you’ll start thinking about these issues by researching ever-stricter privacy laws such as **HIPAA (Health Insurance Portability and Accountability Act)** and the **California Consumer Privacy Act (CCPA)** in the United States and **GDPR (General Data Protection Regulation)** for the European Union.

Ethics

The ACM’s curricula recommendations for Computer Science, Information Technology and Cybersecurity mention ethics more than 100 times. In several Chapter 1 exercises, you’ll focus on ethics issues via Internet research. You’ll investigate privacy and ethical issues surrounding **intelligent assistants**, such as **IBM Watson**, **Amazon Alexa**, **Apple Siri**, **Google Assistant** and **Microsoft Cortana**. For example, a judge ordered Amazon to turn over Alexa recordings for use in a criminal case.¹⁹

Performance

Programmers prefer C (and C++) for performance-intensive operating systems, real-time systems, embedded systems, game systems and communications systems, so we **focus on performance issues**. We use **timing operations** in our multithreading exam-

15. “GitHub.” Accessed November 14, 2020. <https://github.com/>.

16. “GitHub is how people build software.” Accessed November 14, 2020. <https://github.com/about>.

17. “The State of the Octoverse.” Accessed November 14, 2020. <https://octoverse.github.com>.

18. “Microsoft to acquire GitHub for \$7.5 billion.” Accessed November 14, 2020. <https://news.microsoft.com/2018/06/04/microsoft-to-acquire-github-for-7-5-billion/>.

19. “Judge orders Amazon to turn over Echo recordings in double murder case.” Accessed November 14, 2020. <https://techcrunch.com/2018/11/14/amazon-echo-recordings-judge-murder-case/>.

ples to measure the performance improvement we get on today's popular multicore systems, as we employ an increasing number of cores.

Static Code Analysis Tools

Static code analysis tools let you quickly check your code for common errors and security problems and provide insights for improving your code. We checked all our C code using the `clang-tidy` tool (<https://clang.llvm.org/extra/clang-tidy/>). We also used the compiler flag `-Wall` in the GNU `gcc` and Clang compilers to enable all compiler warnings. With a few exceptions for warnings beyond this book's scope, we ensure that our programs compile without warning messages.

How We're Handling C11's Annex K and `printf_s/scanf_s`

The C11 standard's Annex K introduced more secure versions of `printf` (for output) and `scanf` (for input) called `printf_s` and `scanf_s`. We discuss these functions and the corresponding security issues in Sections 6.13 and 7.13:

- Annex K is optional, so not every C vendor implements it. In particular, GNU C++ and Clang C++ do not implement Annex K, so using `scanf_s` and `printf_s` might compromise your code's portability among compilers.
- Microsoft implemented its own Visual C++ versions of `printf_s` and `scanf_s` before the C11 standard. Its compiler immediately began warning on every `scanf` call that `scanf` was deprecated—i.e., it should no longer be used—and that you should consider using `scanf_s` instead. Microsoft now treats what used to be a warning about `scanf` as an error. By default, a program with `scanf` will not compile on Visual C++. Chapter 1's Visual C++ test-drive shows how to handle this issue and compile our programs.
- **Many organizations have coding standards that require code to compile without warning messages.** There are two ways to eliminate Visual C++'s `scanf` warnings—use `scanf_s` instead of `scanf` or disable these warnings.
- There is some discussion of removing Annex K from the C standard. For this reason, we use `printf/scanf` throughout this book and show Visual C++ users how to disable Microsoft's `printf/scanf` errors. **Windows users who prefer not to do that can use the `gcc` compiler in the GNU GCC Docker container, which we discuss in this Preface's "Docker" section. See the Before You Begin section that follows this Preface, and see Section 1.10 for details.**

New Appendix: Intro to Object-Oriented Programming Appendix

C's programming model is called **procedural programming**. We teach it as **structured procedural programming**. After learning C, you'll likely also learn one or more C-based object-oriented languages—such as Java, C++, C#, Objective-C or Swift—and use them side-by-side with C. **Many of these languages support several programming paradigms among procedural programming, object-oriented programming, generic programming and functional-style programming.** In Appendix D, we present a friendly overview of object-oriented programming fundamentals.

A Case Studies Tour

We include many case studies as more substantial chapter examples, exercises and projects (EEPs). These are at an **appropriate level for introductory programming courses**. We anticipate that instructors will select subsets of the case studies appropriate for their particular courses.

Chapter 5—Random-Number Simulation: Building a Casino Game

In this case study, you'll use **random-number generation** and **simulation techniques** to implement the popular casino dice game called craps.

Chapter 5—Random-Number Simulation Case Study: The Tortoise and the Hare Race

In this case study exercise, you'll use **random-number generation** and **simulation techniques** to implement the famous race between the tortoise and the hare.

Chapter 6—Visualizing Binary Search

In this case study, you'll learn the high-speed binary-search algorithm and see a **visualization** that shows the **algorithm's halving effect** that achieves **high performance**.

Chapter 6—Intro to Data Science: Survey Data Analysis

In this case study, you'll learn various **basic descriptive statistics** (mean, median and mode) that are commonly used to “get to know your data.” You'll then build a nice array-manipulation application that calculates these statistics for a batch of survey data.

Chapter 7—Random-Number Simulation—Card Shuffling and Dealing

In this case study, you'll use arrays of strings, random-number generation and simulation techniques to implement a text-based card-shuffling-and-dealing program.

Chapter 7—Embedded Systems Programming: Robotics with the Webots Simulator

Webots (<https://cyberbotics.com/>) is a wonderful open-source, 3D, robotics simulator that runs on Windows, macOS and Linux. It comes bundled with simulations for **dozens of robots** that walk, fly, roll, drive and more:

<https://cyberbotics.com/doc/guide/robots>

You'll use the **free tier** of the Webot robotics simulator to **explore their dozens of simulated robots**. You'll **execute various full-color 3D robotics simulations** written in C and study the provided code. Webots is a **self-contained development environment** that provides a C code editor and compiler. You'll use these tools to **program your own simulations** using Webot's robots.

Webots provides lots of fully coded C programs. A great way for you to learn C is to study existing programs, modify them to work a bit differently and observe the results. Many prominent robotics companies use Webots simulators to prototype new products.

Chapter 7—Systems Software Case Study: Building Your Own Computer (Virtual Machine) with Simulation

In the context of several exercises, you’ll “peel open” a *hypothetical* computer and look at its internal structure. We introduce simple **machine-language programming** and write several small machine-language programs for this computer, which we call the **Simpletron**. As its name implies, it’s a simple machine, but as you’ll see, a powerful one as well. The Simpletron runs programs written in the only language it directly understands—that is, **Simpletron Machine Language**, or **SML** for short. To make this an especially valuable experience, you’ll then **build a computer** (through the technique of **software-based simulation**) on which you can actually run your machine-language programs! The Simpletron experience will give you a basic introduction to the notion of **virtual machines**—one of the most important systems-architecture concepts in modern computing.

Chapter 8—Pqyoaf X Nylfomigrob Qwbbfmh Mndogvk: Rboqlrut yua Boklnxhmywex

This case study exercise’s title looks like gibberish. This is not a mistake! In this exercise, we introduce **cryptography**, which is critically important in today’s connected world. Every day, cryptography is used behind the scenes to **ensure that your Internet-based communications are private and secure**. This case study exercise continues our security emphasis by having readers study the **Vigenère secret-key cipher** algorithm and implement it using array-processing techniques.²⁰ They’ll then use it to encrypt and decrypt messages and to decrypt this section’s title.

Chapter 8—RSA Public-Key Cryptography

Secret key encryption and decryption have a weakness—an encrypted message can be decrypted by anyone who discovers or steals the secret key. We explore public-key cryptography with the RSA algorithm. This technique performs encryption with a public key known to every sender who might want to send a secret message to a particular receiver. The public key can be used to encrypt messages but not decrypt them. Messages can be decrypted only with a paired private key known only to the receiver, so it’s much more secure than the secret key in secret-key cryptography. RSA is among the world’s most widely used public-key cryptography technologies. You’ll build a working, small-scale, classroom version of the RSA cryptosystem.

Chapter 10—Raylib Game Programming Case Studies

In this series of **five case study exercises** and **10 additional exercises**, you’ll use the open-source, cross-platform **raylib**²¹ game programming library, which supports Windows, macOS, Linux and other platforms. The **raylib** development team provides many **C demos** to help you learn key library features and techniques. You’ll study two completely coded games and a dynamic animated visualization that we created:

20. “Vigenère Cipher.” Accessed November 22, 2020. https://en.wikipedia.org/wiki/Vigenère_cipher.

21. “raylib.” Accessed November 14, 2020. <https://www.raylib.com>.

- The Spot-On game tests your reflexes by requiring you to click moving spots before they disappear. With each new game level the spots move faster, making them harder to click.
- The Cannon game challenges you to repeatedly aim and fire a cannon to destroy nine moving targets before a time limit expires. A moving blocker makes the game more difficult.
- The Law of Large Numbers dynamic animated visualization repeatedly rolls a six-sided die and creates an **animated bar chart**. **Visualizations** give you a powerful way to understand data that goes beyond simply looking at raw data. This case study exercise allows students to see the “law of large numbers” at work. When repeatedly rolling a die, we expect each die face to appear approximately 1/6th (16.667%) of the time. For small numbers of rolls (e.g., 60 or 600), you’ll see that the frequencies typically are not evenly distributed. As you simulate larger numbers of die rolls (e.g., 60,000), you’ll see the die frequencies become more balanced. When you simulate significant numbers of die rolls (e.g., 60,000,000), the bars will appear to be the same size.

The games and simulation use various **raylib** capabilities, including **shapes**, **colors**, **sounds**, **animation**, **collision detection** and **user-input events** (such as **mouse clicks**).

After studying our code, you’ll use the raylib graphics, animation and sound features you learn to enhance your implementation of Chapter 5’s **Tortoise and the Hare Race**. You’ll incorporate a traditional horse race’s sounds, and multiple tortoise and hare images to create a fun, animated **multimedia “extravaganza.”** Then, you’ll use a raylib to enhance this chapter’s high-performance card-shuffling-and-dealing simulation to display card images. Finally, you can select from **10 additional raylib game-programming and simulation exercises**. Get creative—have some fun designing and building your own games, too!

Chapter 11—Case Study: Building a Random-Access Transaction-Processing System

In this case study, you’ll use random-access file processing to implement a simple **transaction-processing system** that simulates the kind of high-speed **direct-access** capabilities that industrial-strength database-management systems have. This case study gives you both application-programming and some “under-the-hood” systems-programming experience.

Chapter 11—Artificial Intelligence Case Study: Natural Language Processing (NLP)

Natural Language Processing (NLP) helps computers understand, analyze and process text. One of its most common uses is **sentiment analysis**—determining whether text has positive, neutral or negative sentiment. Another interesting use of NLP is assessing text readability, which is affected by the vocabulary used, word lengths, sentence structure, sentence lengths, topic and more. While writing this book, we used

the paid (NPL) tool Grammarly²² to help tune the writing and ensure the text’s readability for a wide audience. Instructors who use the “**flipped classroom**” format prefer textbooks that students can understand on their own.

Some people believe that the works of William Shakespeare actually might have been written by Christopher Marlowe or Sir Francis Bacon among others.^{23,24} In the NLP case study exercise, you’ll use array-, string- and file-processing techniques to perform **simple similarity detection** on Shakespeare’s *Romeo and Juliet* and Marlowe’s *Edward the Second* to determine how alike they are. You may be surprised by the results.

Chapter 11—Artificial Intelligence Case Study: Machine Learning with the GNU Scientific Library

Statistics can be deceiving. Dramatically different datasets can have identical or nearly identical descriptive statistics. You’ll consider a famous example of this phenomenon—**Anscombe’s Quartet**²⁵—which consists of four datasets of x - y coordinate pairs that differ significantly, yet have nearly identical descriptive statistics. You’ll then study a **completely coded example** that uses the **machine-learning** technique called **simple linear regression** to calculate the equation of a straight line ($y = mx + b$) that, given a collection of points (x - y coordinate pairs) representing an *independent variable* (x) and a *dependent variable* (y), describes the relationship between these variables with a straight line, known as the regression line. As you’ll see, the regression lines for Anscombe’s Quartet are visually identical for all four quite different datasets. The program you’ll study then passes commands to the **open-source gnuplot package** to create several attractive **visualizations**. gnuplot uses its own plotting language different from C, so in our code, we provide extensive comments that explain its commands. Finally, the case study asks you to **run a simple linear regression on 126 years of New York City average January temperature data to determine if there is a cooling or warming trend**. As part of this case study, you’ll also read **comma-separated values (CSV) text files** containing the datasets.

Chapter 11—Web Services and the Cloud: Getting a Weather Report Using libcurl and the OpenWeatherMap Web Services; Introducing Mashups

More and more computing today is done “in the cloud,” using software and data distributed across the Internet worldwide. The apps we use daily are heavily dependent on various cloud-based services. A service that provides access to itself over the Internet is known as a **web service**. In this case study exercise, you’ll work through a completely coded application that uses the **open-source C library libcurl** to invoke an

22. Grammarly has free and paid versions (<https://www.grammarly.com>). They provide free plugins you can use in several popular web browsers.

23. “Did Shakespeare Really Write His Own Plays?” Accessed November 13, 2020. <https://www.history.com/news/did-shakespeare-really-write-his-own-plays>.

24. “Shakespeare authorship question.” Accessed November 13, 2020. https://en.wikipedia.org/wiki/Shakespeare_authorship_question.

25. “Anscombe’s quartet.” Accessed November 13, 2020. https://en.wikipedia.org/wiki/Anscombe%27s_quartet.

OpenWeatherMap (free tier) web service that returns the current weather for a specified city. The web service returns results in JSON (JavaScript Object Notation) format, which we process using the open-source cJSON library.

This exercise opens a world of possibilities. You can explore nearly 24,000 web services listed in the ProgrammableWeb²⁶ web services directory. Many are free or provide free tiers that you can use to create fun and interesting mashups that combine complementary web services.

Chapter 12—Systems Software Case Study: Building Your Own Compiler

In the context of several exercises, you'll build a simple compiler that translates programs written in a simple high-level programming language to our Simpletron Machine Language (SML). You'll write programs in this small new high-level language, compile them on the compiler you build and run them on your Simpletron simulator. And with Chapter 11, File Processing, your compiler can write the generated machine-language code into a file from which your Simpletron computer can then read your SML program, load it into the Simpletron's memory and execute it! This is a nice end-to-end exercise sequence for novice computing students.

Chapter 13—Visualizing the High-Performance Merge Sort

A centerpiece of our sorting treatment is our implementation of the high-performance merge sort algorithm. In that case study, you'll use outputs to visualize the algorithm's partition and merge steps, which will help a user understand how the merge sort works.

Appendix C—Systems Architecture Case Study: Performance with Multithreading and Multicore Systems

Multithreading—which allows you to break a program into separate “threads” that can be executed in parallel—has been around for many decades, but interest in it is higher today due to the availability of multicore processors in computers and devices, including smartphones and tablets. These processors economically implement multiple processors on one integrated circuit chip. They put multiple cores to work executing different parts of your program in parallel, thereby enabling the individual tasks and the program as a whole to complete faster. Four and eight cores are common in many of today's devices, and the number of cores will continue to grow. We wrote and tested the code for this book using an eight-core MacBook Pro. Multithreaded applications enable you to execute separate threads simultaneously on multiple cores, so that you can take the fullest advantage of multicore architecture.

For a convincing demonstration of the power of multithreading on a multicore system, we present a case study with two programs. One performs two compute-intensive calculations in sequence. The other executes the same compute-intensive calculations in parallel threads. We time each calculation and determine the total execution time in each program. The program outputs show the dramatic time improvement when the multithreaded version executes on a multicore system.

26. “ProgrammableWeb.” Accessed November 22, 2020. <https://programmableweb.com/>.

Secure C Programming

The people responsible for the ACM/IEEE curricula guidelines emphasize the importance of security—it's **mentioned 395 times in the Computer Science Curricula document** and **235 times in the Information Technology Curricula document**. In 2017, the ACM/IEEE published its **Cybersecurity Curricula**, which focuses on security courses and security throughout the other computing curricula. That document **mentions security 865 times**.

Chapters 2–12 and 14 each end with a **Secure C Programming** section. These are designed to raise awareness among novice programming students of security issues that could cause breaches. These sections present some key issues and techniques and provide links and references so you can continue learning. Our goal is to encourage you to start thinking about security issues, even if this is your first programming course.

Experience has shown that it's challenging to build industrial-strength systems that stand up to attacks. 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 significantly reduce vulnerabilities.

The CERT Division of **Carnegie Mellon's Software Engineering Institute**

<https://www.sei.cmu.edu/about/divisions/cert/index.cfm>

was created to analyze and respond promptly to attacks. They publish and promote secure coding standards to help C programmers and others implement industrial-strength systems that avoid the programming practices that leave systems vulnerable to attacks. Their standards evolve as new security issues arise.

We explain how to upgrade your code (as appropriate for an introductory book) to conform to the latest secure C coding recommendations. If you're building C systems in industry, consider reading the *SEI CERT C Coding Standard* rules at

<https://wiki.sei.cmu.edu/confluence/display/c>

Also, consider reading *Secure Coding in C and C++, 2/e* by Robert Seacord (Addison-Wesley Professional, 2013). Mr. Seacord, a technical reviewer for an earlier edition of this book, provided specific recommendations on each of our Secure C Programming sections. At the time, he was the Secure Coding Manager at CERT and an adjunct professor at Carnegie Mellon's School of Computer Science. He is now a Technical Director at NCC Group (an IT Security company).

Our Secure C Programming sections discuss many important topics, including:

- Testing for Arithmetic Overflows
- 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.

Computing and Data Science Curricula

This book is designed for courses that adhere to one or more of the following ACM/IEEE CS-and-related curriculum documents:

- CC2020: Paradigms for Future Computing Curricula (cc2020.net),²⁷
- Computer Science Curricula 2013,²⁸
- Information Technology Curricula 2017,²⁹
- Cybersecurity Curricula 2017.³⁰

Computing Curricula

- According to “CC2020: A Vision on Computing Curricula,”³¹ the curriculum “needs to be reviewed and updated to include the new and emerging areas of computing such as **cybersecurity** and **data science**.”³² (See “Data Science Overlaps with Computer Science” below and this Preface’s earlier “**Secure C Programming**” section).
- Data science includes key topics (besides statistics and general-purpose programming) such as **machine learning**, **deep learning**, **natural language processing**, **speech synthesis and recognition**, and others that are **classic artificial intelligence (AI) topics—and hence CS topics as well**. We cover **machine learning** and **natural language processing** in the case studies.

27. “Computing Curricula 2020.” Accessed November 22, 2020. <https://cc2020.nsparc.ms-state.edu/wp-content/uploads/2020/11/Computing-Curricula-Report.pdf>.

28. ACM/IEEE (Assoc. Comput. Mach./Inst. Electr. Electron. Eng.). 2013. *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science* (New York: ACM), <http://ai.stanford.edu/users/sahami/CS2013/final-draft/CS2013-final-report.pdf>.

29. *Information Technology Curricula 2017*, <http://www.acm.org/binaries/content/assets/education/it2017.pdf>.

30. *Cybersecurity Curricula 2017*, https://cybered.hosting.acm.org/wp-content/uploads/2018/02/newcover_csec2017.pdf.

31. A. Clear, A. Parrish, G. van der Veer and M. Zhang, “CC2020: A Vision on Computing Curricula,” <https://dl.acm.org/citation.cfm?id=3017690>.

32. <http://delivery.acm.org/10.1145/3020000/3017690/p647-clear.pdf>.

Data Science Overlaps with Computer Science³³

The undergraduate data science curriculum proposal³⁴ includes algorithm development, programming, computational thinking, data structures, database, mathematics, statistical thinking, machine learning, data science and more—a significant overlap with computer science, especially given that the **data science courses include some key AI topics**. Even though ours is a C programming textbook, we **work data science topics into various examples, exercises, projects and case studies**.

Key Points from the Data Science Curriculum Proposal

This section calls out some key points from the data science undergraduate curriculum proposal and its detailed course descriptions appendix.³⁵ Each of the following items is covered in *C How to Program, 9/e*:

- Learn **programming fundamentals** commonly presented in **computer science** courses, including working with **data structures**.
- Be able to **solve problems** by **creating algorithms**.
- Work with **procedural programming**.
- **Explore concepts via simulations**.
- Use **development environments** (we tested all our code on **Microsoft Visual C++**, **Apple Xcode**, the **GNU command-line gcc compiler on Linux** and in the **GNU Compiler Collection Docker container**).
- Work with **real-world data** in **practical case studies** and **projects**—such as William Shakespeare’s *Romeo and Juliet* and Christopher Marlowe’s *Edward the Second* from Project Gutenberg (<https://www.gutenberg.org/>), and 126 years of New York City average January temperatures.
- Create **data visualizations**.
- Communicate **reproducible results**. (**Docker** plays an important role in that—see the next page.)
- Work with **existing software** and **cloud-based tools**.
- Work with **high-performance tools**, such as **C’s multithreading libraries**.
- Focus on **data’s ethics, security, privacy and reproducibility** issues.

33. This section is intended primarily for data science instructors. Given that the emerging 2020 Computing Curricula for computer science and related disciplines is likely to include some key data science topics, this section includes important information for computer science instructors as well.

34. “Curriculum Guidelines for Undergraduate Programs in Data Science,” <http://www.anualreviews.org/doi/full/10.1146/annurev-statistics-060116-053930>.

35. “Appendix—Detailed Courses for a Proposed Data Science Major,” http://www.anualreviews.org/doi/suppl/10.1146/annurev-statistics-060116-053930/suppl_file/st04_de_veaux_supmat.pdf.

Get the Code Examples and Install the Software

For your convenience, we provide the book's examples in C source-code (.c) files for use with integrated development environments (IDEs) and command-line compilers. See the Before You Begin section that follows the Preface for software installation details. See the **Chapter 1 test-drives** for information on running the book's code examples. If you encounter a problem, you can reach us at deitel@deitel.com or via the contact form at <https://deitel.com/contact-us>.

Docker

We introduce **Docker**—a tool for packaging software into **containers** that bundle everything required to execute that software conveniently, **reproducibly** and **portably** across platforms. Some software packages you'll use require complicated setup and configuration. For many of these, you can download free preexisting **Docker containers** that help you avoid complex installation issues. You can simply execute software locally on your desktop or notebook computers, making Docker a great way to help you get started with new technologies quickly, conveniently and economically. For your convenience, we show how to install and execute a Docker container that's **pre-configured** with the **GNU Compiler Collection (GCC)**, which includes the **gcc compiler**. This can run in Docker on **Windows, macOS** and **Linux**. It's particularly useful for people using Visual C++, which can compile C code but is not 100% compliant with the latest C standard.

Docker also helps with **reproducibility**. Custom Docker containers can be configured with every piece of software and every library you use. This would enable others to recreate the environment you used, then reproduce your work, and will help you reproduce your own results. **Reproducibility is especially important in the sciences and medicine—for example, when researchers want to prove and extend the work in published articles.**

Flipped Classrooms

Many instructors use “**flipped classrooms.**”^{36,37} Students learn the content on their own before coming to class, and class time is used for tasks such as hands-on coding, working in groups and discussions. Our book and supplements also are appropriate for flipped classrooms:

- We use **Grammarly** to control the book's reading level to help ensure it's appropriate for students learning on their own.
- In parallel with reading the text, students should **execute** the **147 live-code C examples** and do the **350+** integrated **Self Check exercises, which are imme-**

36. https://en.wikipedia.org/wiki/Flipped_classroom.

37. <https://www.edsurge.com/news/2018-05-24-a-case-for-flipping-learning-without-videos>.

diately followed by their answers. These encourage active participation by the student. They learn the content in small pieces using a “**read-a-little, do-a-little, test-a-little**” approach—appropriate for a flipped classroom’s active, self-paced, hands-on learning. Students are encouraged to modify the code and see the effects of their changes.

- We provide **445 exercises and projects**, which students can work on at home and/or in class. Many of the exercises are at an elementary or intermediate level that students should be able to do independently. And many are appropriate for **group projects** on which students can collaborate in class.
- **Section-by-section detailed chapter summaries** with **bolded** key terms help students quickly review the material.
- In the book’s extensive index, the **defining occurrences** of key terms are highlighted with a **bold** page number, making it easy for students to find the introductions to the topics they’re studying. This facilitates the outside-the-classroom learning experience of the **flipped classroom**.

A key aspect of flipped classrooms is getting your questions answered when you’re working on your own. See the “Getting Your Questions Answered” section later in this Preface for details. And you can always reach us at deitel@deitel.com.

Teaching Approach

C How to Program, 9/e contains a rich collection of examples, exercises, projects and case studies drawn from many fields. Students solve interesting, **real-world problems** working with **real-world data**. The book concentrates on the principles of good **software engineering** and stresses **program clarity**.

Using Fonts for Emphasis

We place the key terms and the index’s page reference for each defining occurrence in **bold text** for easier reference. C code uses a fixed-width font (e.g., `x = 5`). We place on-screen components in the **bold Helvetica** font (e.g., the **File** menu).

Syntax Coloring

For readability, we syntax color all the code. In our full-color books and e-books, our syntax-coloring conventions are as follows:

```

comments appear in green
keywords appear in dark blue
constants and literal values appear in light blue
errors appear in red
all other code appears in black

```

Objectives and Outline

Each chapter begins with objectives that tell you what to expect and give you an opportunity, after reading the chapter, to determine whether it has met the intended goals. The chapter outline enables students to approach the material in a top-down fashion.

Examples







The book's 147 live-code examples contain thousands of lines of proven code.

Tables and Illustrations

Abundant tables and line drawings are included.

Programming Wisdom

We integrate into the text discussions programming wisdom and mistakes we've accumulated from our combined nine decades of programming and teaching experience, and from the scores of academics and industry professionals who have reviewed the nine editions of this book over the past 29 years, including:

- ERR 
 • **Good programming practices** and preferred C idioms that help you produce clearer, more understandable and more maintainable programs.
- ERR 
 • **Common programming errors** to reduce the likelihood that you'll make them.
- ERR 
 • **Error-prevention tips** with suggestions for exposing bugs and removing them from your programs. Many of these tips describe techniques for preventing bugs from getting into your programs in the first place.
- PERF 
 • **Performance tips** highlighting opportunities to make your programs run faster or minimize the amount of memory they occupy.
- SE 
 • **Software engineering observations** highlighting architectural and design issues for proper software construction, especially for larger systems.
- SEC 
 • **Security best practices** that will help you strengthen your programs against attacks.

Section-By-Section Chapter Summaries

To help students quickly review the material, each chapter ends with a detailed bullet-list summary with **bolded** key terms and, for most, **bold** page references to their defining occurrences.

Free Software Used in the Book

The **Before You Begin** section following this Preface discusses installing the software you'll need to work with our examples. We tested *C How to Program, 9/e*'s examples using the following popular *free* compilers:

- **GNU gcc** on Linux—which is already installed on most Linux systems and can be installed on macOS and Windows systems.
- **Microsoft's Visual Studio Community Edition** on Windows.
- **Apple's Clang compiler in Xcode** on macOS.

GNU gcc in Docker

We also demonstrate **GNU gcc** in a **Docker container**—ideal for instructors who want all their students to use GNU gcc, regardless of their operating system. This

gives Visual C++ users a true C compiler option, since Visual C++ is not 100% compliant with the latest C standard.

Windows Subsystem for Linux

The **Windows Subsystem for Linux (WSL)** enables Windows users to install Linux and run it inside Windows. We provide a link to Microsoft's step-by-step instructions for setting up WSL and installing a Linux distribution. This provides yet another option for Windows users to access the GNU gcc compiler.

C Documentation

You'll find the following documentation helpful as you work through the book:

- The GNU C Standard Library Reference Manual:
<https://www.gnu.org/software/libc/manual/pdf/libc.pdf>
- C Language Reference at [cppreference.com](https://en.cppreference.com)
<https://en.cppreference.com/w/c>
- C Standard Library Headers at [cppreference.com](https://en.cppreference.com)
<https://en.cppreference.com/w/c/header>
- Microsoft's C Language Reference:
<https://docs.microsoft.com/en-us/cpp/c-language/c-language-reference>

Getting Your Questions Answered

Online forums enable you to interact with other C programmers worldwide and get your questions answered. Popular C and general programming online forums include:

- <https://stackoverflow.com>
- https://www.reddit.com/r/C_Programming/
- <https://groups.google.com/forum/#!forum/comp.lang.C>
- <https://cboard.cprogramming.com/c-programming/>
- <https://www.dreamincode.net/forums/forum/15-c-and-c/>

For a list of other sites, see

<https://www.geeksforgeeks.org/stuck-in-programming-get-the-solution-from-these-10-best-websites/>

Also, vendors often provide forums for their tools and libraries. Many libraries are managed and maintained at **github.com**. Some library maintainers provide support through the **Issues** tab on a given library's GitHub page.

Student and Instructor Supplements

The following supplements are available to students and instructors.

Web-Based Materials on deitel.com

To get the most out of your *C How to Program, 9/e* learning experience, you should execute each code example in parallel with reading the corresponding discussion. On the book's web page at

<https://deitel.com/c-how-to-program-9-e>

we provide the following resources:

- Links to the **downloadable C source code (.c files)** for the book's code examples and exercises that include code in the exercise description. You can also get this from the book's Pearson companion website at <https://pearson.com/deitel>
- Links to our **Getting Started videos** showing how to use the compilers and code examples. We also introduce these tools in **Chapter 1**.
- **Blog posts**—<https://deitel.com/blog>.
- **Book updates**—<https://deitel.com/c-how-to-program-9-e>.
- “Using the Debugger” appendices for the Visual Studio, GNU **gdb** and Xcode debuggers.

For more information about downloading the examples and setting up your C development environment, see the **Before You Begin** section that follows this Preface.

Instructor Supplements

Pearson Education's IRC (Instructor Resource Center)

<http://www.pearsonhighered.com/irc>

provides qualified instructors access to the following supplements for this book:

- **PowerPoint Slides.**
- **Instructor Solutions Manual** with solutions to most of the exercises. Solutions are not provided for “project” and “research” exercises. **Before assigning a particular exercise for homework, instructors should check the IRC to ensure that the solution is available.**
- **Test Item File** with four-part multiple-choice, short-answer questions and answers. You may also request from your Pearson representative (<https://pearson.com/relocator>) versions of the **Test Item File** for use with popular automated assessment tools.

Please do not write to us requesting access to the Pearson Instructor's Resource Center (IRC). Access to the instructor supplements and exercise solutions on the IRC is strictly limited by our publisher to college instructors who adopt the book for their classes. Instructors may obtain access through their Pearson representatives. If you're not a registered faculty member, contact your Pearson representative or visit

<https://pearson.com/relocator>

Instructors can request **examination copies** of Deitel books from their Pearson representatives.

Communicating with the Authors

For questions, instructor syllabus assistance or to report an error, we're easy to reach at deitel@deitel.com

or via the contact form at

<https://deitel.com/contact-us>

Interact with us via **social media** on

- **Facebook**[®]—<https://facebook.com/DeitelFan>
- **Twitter**[®]—[@deitel](https://twitter.com/deitel) or <https://twitter.com/deitel>
- **LinkedIn**[®]—<https://linkedin.com/company/deitel-&-associates>
- **YouTube**[®]—<https://youtube.com/DeitelTV>

Deitel Pearson Products on O'Reilly Online Learning

O'Reilly Online Learning subscribers have access to many Deitel Pearson textbooks, professional books, LiveLessons videos and Full Throttle one-day webinars. Sign up for a 10-day free trial at

<https://deitel.com/LearnWithDeitel>

Textbooks and Professional Books

Each Deitel e-book on O'Reilly Online Learning is presented in full color and extensively indexed.

Asynchronous LiveLessons Video Products

Learn hands-on with Paul Deitel as he presents compelling, leading-edge computing technologies in Python, Python Data Science/AI and Java. C++20 and C are coming in 2021.

Live Full Throttle Webinars

Paul Deitel offers **Full Throttle webinars** at O'Reilly Online Learning. These are one-full-day, fast-paced, code-intensive introductions to Python, Python Data Science/AI and Java, with C++20 and C coming in 2021. Paul's Full Throttle webinars are for experienced developers and software project managers preparing for projects using other languages. After taking a Full Throttle course, participants often take the corresponding LiveLessons video course which has many more hours of classroom-paced learning.

Acknowledgments

We'd like to thank Barbara Deitel for long hours devoted to Internet research on this project. We're fortunate to have worked with the dedicated team of publishing professionals at Pearson. We appreciate the guidance, wisdom and energy of Tracy Johnson (Pearson Education, Global Content Manager, Computer Science)—on all our

academic publications, both print and digital. She challenges us at every step of the process to “get it right” and make the best books. Carole Snyder managed the book’s production and interacted with Pearson’s permissions team, promptly clearing our graphics and citations to keep the book on schedule. Erin Sullivan recruited and managed the book’s review team. We selected the cover art, and Chuti Prasertsith designed the cover, adding his special touch of graphics magic.

We wish to acknowledge the efforts of our academic and professional reviewers. Adhering to a tight schedule, the reviewers scrutinized the manuscript, providing countless suggestions for improving the presentation’s accuracy, completeness and timeliness. They helped us make a better book.

Reviewers

C How to Program, 9/e Reviewers

Dr. Danny Kalev (Ben-Gurion University of the Negev, A Certified System Analyst, C Expert and Former Member of the C++ Standards Committee)
José Antonio González Seco (Parliament of Andalusia)

C How to Program, 8/e Reviewers

Dr. Brandon Invergo (GNU/European Bioinformatics Institute)
Jim Hogg (Program Manager, C/C++ Compiler Team, Microsoft Corporation)
José Antonio González Seco (Parliament of Andalusia)
Alan Bunning (Purdue University)
Paul Clingan (Ohio State University)
Michael Geiger (University of Massachusetts, Lowell)
Dr. Danny Kalev (Ben-Gurion University of the Negev, A Certified System Analyst, C Expert and Former Member of the C++ Standards Committee)
Jeonghwa Lee (Shippensburg University)
Susan Mengel (Texas Tech University)
Judith O'Rourke (SUNY at Albany)
Chen-Chi Shin (Radford University)

Other Recent Editions Reviewers (and their affiliations at the time)

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)
Vytautas 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)
Benjamin Seyfarth (University of Southern Mississippi)
Gary Sibbitts (St. Louis Community College at Meramec)
William Smith (Tulsa Community College)
Douglas Walls (Senior Staff Engineer, C compiler, Sun Microsystems—now Oracle).

A special thanks to Prof. Alison Clear, an Associate Professor in the School of Computing at Eastern Institute of Technology (EIT) in New Zealand and co-chair of the **Computing Curricula 2020 (CC2020) Task Force**, which recently released new computing curricula recommendations:

Computing Curricula 2020: Paradigms for Future Computing Curricula
[https://cc2020.nsparc.msstate.edu/wp-content/uploads/2020/11/
Computing-Curricula-Report.pdf](https://cc2020.nsparc.msstate.edu/wp-content/uploads/2020/11/Computing-Curricula-Report.pdf)

Prof. Clear graciously answered our questions.

And finally, a special note of thanks to the enormous numbers of technically oriented people worldwide who contribute to the open-source movement and write about their work online, to their organizations that encourage the proliferation of such open software and information, and to Google, whose search engine answers our constant stream of questions, each in a fraction of a second, at any time day or night—and at no charge.

Well, there you have it! As you read the book, we'd appreciate your comments, criticisms, corrections and suggestions for improvement. Please send all correspondence, including questions, to

`deitel@deitel.com`

We'll respond promptly. Welcome to the exciting world of C programming for the 2020s. We hope you have an informative, entertaining and challenging learning experience with *C How to Program, 9/e* and enjoy this look at leading-edge software development with C. We wish you great success!

Paul Deitel

Harvey Deitel

About the Authors

Paul J. Deitel, CEO and Chief Technical Officer of Deitel & Associates, Inc., is an MIT graduate with 41 years of experience in computing. Paul is one of the world's most experienced programming-languages trainers, having taught professional courses to software developers since 1992. He has delivered hundreds of programming courses to academic, industry, government and military clients internationally, including UCLA, Cisco, IBM, Siemens, Sun Microsystems (now Oracle), Dell, Fidelity, NASA at the Kennedy Space Center, the National Severe Storm Laboratory, White Sands Missile Range, Rogue Wave Software, Boeing, Nortel Networks, Puma, iRobot and many more. He and his co-author, Dr. Harvey M. Deitel, are among the world's best-selling programming-language textbook, professional book, video and interactive multimedia e-learning authors, and virtual- and live-training presenters.

Dr. Harvey M. Deitel, Chairman and Chief Strategy Officer of Deitel & Associates, Inc., has 59 years of experience in computing. Dr. Deitel earned B.S. and M.S. degrees in Electrical Engineering from MIT and a Ph.D. in Mathematics from Boston University—he studied computing in each of these programs before they spun off Computer Science programs. He has extensive college teaching experience, including earning tenure and serving as the Chairman of the Computer Science Department at Boston College before founding Deitel & Associates, Inc., in 1991 with his son, Paul. The Deitels' publications have earned international recognition,

with more than 100 translations published in Japanese, German, Russian, Spanish, French, Polish, Italian, Simplified Chinese, Traditional Chinese, Korean, Portuguese, Greek, Urdu and Turkish. Dr. Deitel has delivered hundreds of programming courses to academic, corporate, government and military clients.

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 some of the world's largest companies, government agencies, branches of the military, and academic institutions. The company offers instructor-led training courses delivered virtually and live at client sites worldwide, and for Pearson Education on O'Reilly Online Learning.

Through its 45-year publishing partnership with Pearson/Prentice Hall, Deitel & Associates, Inc., publishes leading-edge programming textbooks and professional books in **print** and **e-book** formats, **LiveLessons** video courses, **O'Reilly Online Learning live webinars** and **Revel™** interactive multimedia college courses.

To contact Deitel & Associates, Inc. and the authors, or to request a proposal for virtual or on-site, instructor-led training worldwide, write to

deitel@deitel.com

To learn more about Deitel on-site corporate training, visit

<https://deitel.com/training>

Individuals wishing to purchase Deitel books can do so at

<https://amazon.com>

Bulk orders by corporations, the government, the military and academic institutions should be placed directly with Pearson. For corporate and government sales, send an email to

corpsales@pearsoned.com

For textbook orders visit

<https://pearson.com>

Deitel e-books are available in various formats from

<https://www.amazon.com/>

<https://www.vitalsource.com/>

<https://www.bn.com/>

<https://www.redshelf.com/>

<https://www.informit.com/>

<https://www.chegg.com/>

To register for a free 10-day trial to O'Reilly Online Learning, visit

<https://deitel.com/LearnWithDeitel>

which will forward you to our O'Reilly Online Learning landing page. On that page, click the **Begin a free trial** link.



Before You Begin

Before using this book, please read this section to understand our conventions and ensure that your computer can compile and run our example programs.

Font and Naming Conventions

We use fonts to distinguish application elements and C++ code elements from regular text. For on-screen application elements, we use a **sans-serif bold font**, as in the **File** menu. For C code elements, we use a **sans-serif font**, as in `sqrt(9)`.

Obtaining the Code Examples

We maintain the code examples for *C How to Program, 9/e* in a GitHub repository. The book's web page

<https://deitel.com/c-how-to-program-9-e>

includes a link to the repository and a link to a ZIP file containing the code. If you download the ZIP file, be sure to extract its contents once the download completes. In our instructions, we assume the examples reside in your user account's Documents folder in a subfolder named `examples`.

If you're not familiar with Git and GitHub but are interested in learning about these essential developer tools, check out their guides at

<https://guides.github.com/activities/hello-world/>

Compilers We Use in *C How to Program, 9/e*

We tested *C How to Program, 9/e*'s examples using the following free compilers:

- For Microsoft Windows, we used Microsoft Visual Studio Community edition¹, which includes the Visual C++ compiler and other Microsoft development tools. Visual C++ can compile C code.
- For macOS, we used Apple Xcode, which includes the Clang C compiler. Command-line Clang also can be installed on Linux and Windows Systems.
- For Linux, we used the GNU gcc compiler—part of the GNU Compiler Collection (GCC). GNU gcc is already installed on most Linux systems and can be installed on macOS and Windows systems.

1. At the time of this writing, the current version was Visual Studio 2019 Community edition.

This Before You Begin section describes installing the compilers. Section 1.10's test-drives demonstrate how to compile and run C programs using these compilers.

Before You Begin Videos

To help you get started with each of our preferred compilers, we provide Before You Begin videos at:

<https://deitel.com/c-how-to-program-9-e>

We also provide a Before You Begin video demonstrating how to install the GNU GCC Docker container. This enables you to use the gcc compiler on any Docker-enabled computer.² See the section, “Docker and the GNU Compiler Collection (GCC) Docker Container” later in this Before You Begin section.

Installing Visual Studio Community Edition on Windows

If you use Windows, first ensure that your system meets the requirements for Microsoft Visual Studio Community edition at:

<https://docs.microsoft.com/en-us/visualstudio/releases/2019/system-requirements>

Next, go to:

<https://visualstudio.microsoft.com/downloads/>

then perform the following installation steps:

1. Click **Free download** under **Community**.
2. Depending on your web browser, you may see a pop-up at the bottom of your screen in which you can click **Run** to start the installation process. If not, double-click the installer file in your **Downloads** folder.
3. In the **User Account Control** dialog, click **Yes** to allow the installer to make changes to your system.
4. In the **Visual Studio Installer** dialog, click **Continue** to allow the installer to download the components it needs for you to configure your installation.
5. For this book's examples, select the option **Desktop Development with C++**, which includes the Visual C++ compiler and the C and C++ standard libraries.
6. Click **Install**. Depending on your Internet connection speed, the installation process can take a significant amount of time.

Installing Xcode on macOS

On macOS, perform the following steps to install Xcode:

1. Click the Apple menu and select **App Store...**, or click the **App Store** icon in the dock at the bottom of your Mac screen.

2. “Docker Frequently Asked Questions (FAQ).” Accessed January 3, 2021. <https://docs.docker.com/engine/faq/>.

2. In the App Store's Search field, type Xcode.
3. Click the Get button to install Xcode.

Installing GNU gcc on Linux

Most Linux users already have a recent version of GNU gcc installed. To check, open a shell or Terminal window on your Linux system, then enter the command

```
gcc --version
```

If it does not recognize the command, you must install GNU gcc. We use the Ubuntu Linux distribution. On that distribution, you must be logged in as an administrator or have the administrator password to execute the following commands:

1. `sudo apt update`
2. `sudo apt install build-essential gdb`

Linux distributions often use different software installation and upgrade techniques. If you are not using Ubuntu Linux, search online for “Install GCC on *MyLinuxDistribution*” and replace *MyLinuxDistribution* with your Linux version. You can download the GNU Compiler Collection for various platforms at:

```
https://gcc.gnu.org/install/binaries.html
```

Installing GNU GCC in Ubuntu Linux Running on the Windows Subsystem for Linux

Another way to install GNU gcc on Windows is via the **Windows Subsystem for Linux (WSL)**, which enables you to run Linux on Windows. Ubuntu Linux provides an easy-to-use installer in the Windows Store, but first you must install WSL:

1. In the search box on your taskbar, type “Turn Windows features on or off,” then click **Open** in the search results.
2. In the Windows Features dialog, locate **Windows Subsystem for Linux** and ensure that it is checked. If it is, WSL is already installed. Otherwise, check it and click **OK**. Windows will install WSL and ask you to reboot your system.
3. Once the system reboots and you log in, open the **Microsoft Store** app and search for **Ubuntu**, select the app named **Ubuntu** and click **Install**. This installs the latest version of Ubuntu Linux.
4. Once installed, click the **Launch** button to display the Ubuntu Linux command-line window, which will continue the installation process. You'll be asked to create a username and password for your Ubuntu installation—these do not need to match your Windows username and password.
5. When the Ubuntu installation completes, execute the following two commands to install the GCC and the GNU debugger—you may be asked enter your Ubuntu password for the account you created in Step 6:

```
sudo apt-get update
sudo apt-get install build-essential gdb
```

liv Before You Begin

6. Confirm that `gcc` is installed by executing the following command:

```
gcc --version
```

To access our code files, use the `cd` command change the folder within Ubuntu to:

```
cd /mnt/c/Users/YourUserName/Documents/examples
```

Use your own user name and update the path to where you placed our examples on your system.

GNU Compiler Collection (GCC) Docker Container

Docker is a tool for packaging software into **containers** (also called **images**) that bundle everything required to execute software across platforms. Docker is particularly useful for software packages with complicated setups and configurations. You typically can download preexisting Docker containers (often at <https://hub.docker.com>) for free and execute them locally on your desktop or notebook computer. This makes Docker a great way to get started with new technologies quickly and conveniently.

Docker makes it easy to use the GNU Compiler Collection on most versions of Windows 10, and on macOS and Linux. The GNU Docker containers are located at

```
https://hub.docker.com/\_/gcc
```

Installing Docker

To use the GCC Docker container, first install Docker. Windows (64-bit)³ and macOS users should download and run the **Docker Desktop** installer from:

```
https://www.docker.com/get-started
```

then follow the on-screen instructions. Linux users should install **Docker Engine** from:

```
https://docs.docker.com/engine/install/
```

Also, sign up for a **Docker Hub** account on this webpage so you can install pre-configured containers from <https://hub.docker.com>.

Downloading the Docker Container

Once Docker is installed and running, open a Command Prompt (Windows), Terminal (macOS/Linux) or shell (Linux), then execute the command:

```
docker pull gcc:latest
```

Docker downloads the GNU Compiler Collection (GCC) container's current version.⁴ In one of Section 1.10's test-drives, we'll demonstrate how to execute the container and use it to compile and run C programs.

3. If you have Windows Home (64-bit), follow the instructions at <https://docs.docker.com/docker-for-windows/install-windows-home/>.

4. At the time of this writing, the current version of the GNU Compiler Collection is 10.2.

Introduction to Computers and C

I

Objectives

In this chapter, you'll:

- Learn about exciting recent developments in computing.
- Learn computer hardware, software and Internet basics.
- Understand the data hierarchy from bits to databases.
- Understand the different types of programming languages.
- Understand the strengths of C and other leading programming languages.
- Be introduced to the C standard library of reusable functions that help you avoid “reinventing the wheel.”
- Test-drive a C program that you compile with one or more of the popular C compilers we used to develop the book's hundreds of C code examples, exercises and projects (EEPs).
- Be introduced to big data and data science.
- Be introduced to artificial intelligence—a key intersection of computer science and data science.



- 1.1 Introduction
- 1.2 Hardware and Software
 - 1.2.1 Moore's Law
 - 1.2.2 Computer Organization
- 1.3 Data Hierarchy
- 1.4 Machine Languages, Assembly Languages and High-Level Languages
- 1.5 Operating Systems
- 1.6 The C Programming Language
- 1.7 The C Standard Library and Open-Source Libraries
- 1.8 Other Popular Programming Languages
- 1.9 Typical C Program-Development Environment
 - 1.9.1 Phase 1: Creating a Program
 - 1.9.2 Phases 2 and 3: Preprocessing and Compiling a C Program
 - 1.9.3 Phase 4: Linking
 - 1.9.4 Phase 5: Loading
 - 1.9.5 Phase 6: Execution
 - 1.9.6 Problems That May Occur at Execution Time
 - 1.9.7 Standard Input, Standard Output and Standard Error Streams
- 1.10 Test-Driving a C Application in Windows, Linux and macOS
 - 1.10.1 Compiling and Running a C Application with Visual Studio 2019 Community Edition on Windows 10
 - 1.10.2 Compiling and Running a C Application with Xcode on macOS
 - 1.10.3 Compiling and Running a C Application with GNU `gcc` on Linux
 - 1.10.4 Compiling and Running a C Application in a GCC Docker Container Running Natively over Windows 10, macOS or Linux
- 1.11 Internet, World Wide Web, the Cloud and IoT
 - 1.11.1 The Internet: A Network of Networks
 - 1.11.2 The World Wide Web: Making the Internet User-Friendly
 - 1.11.3 The Cloud
 - 1.11.4 The Internet of Things
- 1.12 Software Technologies
- 1.13 How Big Is Big Data?
 - 1.13.1 Big-Data Analytics
 - 1.13.2 Data Science and Big Data Are Making a Difference: Use Cases
- 1.14 Case Study—A Big-Data Mobile Application
- 1.15 AI—at the Intersection of Computer Science and Data Science

Self-Review Exercises | Answers to Self-Review Exercises | Exercises

1.1 Introduction

Welcome to C—one of the world's most senior computer programming languages and, according to the Tiobe Index, the world's most popular.¹ You're probably familiar with many of the powerful tasks computers perform. In this textbook, you'll get intensive, hands-on experience writing C instructions that command computers to perform those and other tasks. **Software** (that is, the C instructions you write, which are also called **code**) controls **hardware** (that is, computers and related devices).

1. "TIOBE Index." Accessed November 4, 2020. <https://www.tiobe.com/tiobe-index/>.

C is widely used in industry for a wide range of tasks.² Today’s popular desktop operating systems—Windows³, macOS⁴ and Linux⁵—are partially written in C. Many popular applications are partially written in C, including popular web browsers (e.g., Google Chrome⁶ and Mozilla Firefox⁷), database management systems (e.g., Microsoft SQL Server⁸, Oracle⁹ and MySQL¹⁰) and more.

In this chapter, we introduce terminology and concepts that lay the groundwork for the C programming you’ll learn, beginning in Chapter 2. We’ll introduce hardware and software concepts. We’ll also overview the data hierarchy—from individual bits (ones and zeros) to databases, which store the massive amounts of data that organizations need to implement contemporary applications such as Google Search, Netflix, Twitter, Waze, Uber, Airbnb and a myriad of others.

We’ll discuss the types of programming languages. We’ll introduce the C standard library and various C-based “open-source” libraries that help you avoid “reinventing the wheel.” You’ll use these libraries to perform powerful tasks with modest numbers of instructions. We’ll introduce additional software technologies that you’re likely to use as you develop software in your career.

Many development environments are available in which you can compile, build and run C applications. You’ll work through one or more of the four test-drives showing how to compile and execute C code using:

- Microsoft Visual Studio 2019 Community edition for Windows.
- Clang in Xcode on macOS.
- GNU gcc in a shell on Linux.
- GNU gcc in a shell running inside the GNU Compiler Collection (GCC) Docker container.

You can read only the test-drive(s) required for your course or projects in industry.

In the past, most computer applications ran on “standalone” computers (that is, not networked together). Today’s applications can communicate among the world’s computers via the Internet. We’ll introduce the Internet, the World Wide Web, the Cloud and the Internet of Things (IoT), each of which could play a significant part in the applications you’ll build in the 2020s (and probably long afterward).

-
2. “After All These Years, the World is Still Powered by C Programming.” Accessed Nov. 4, 2020. <https://www.toptal.com/c/after-all-these-years-the-world-is-still-powered-by-c-programming>.
 3. “What Programming Language is Windows written in?” Accessed Nov. 4, 2020. <https://social.microsoft.com/Forums/en-US/65a1fe05-9c1d-48bf-bd40-148e6b3da9f1/what-programming-language-is-windows-written-in>.
 4. “macOS.” Accessed Nov. 4, 2020. <https://en.wikipedia.org/wiki/MacOS>.
 5. “Linux kernel.” Accessed Nov. 4, 2020. https://en.wikipedia.org/wiki/Linux_kernel.
 6. “Google Chrome.” Accessed Nov. 4, 2020. https://en.wikipedia.org/wiki/Google_Chrome.
 7. “Firefox.” Accessed Nov. 4, 2020. <https://en.wikipedia.org/wiki/Firefox>.
 8. “Microsoft SQL Server.” Accessed Nov. 4, 2020. https://en.wikipedia.org/wiki/Microsoft_SQL_Server.
 9. “Oracle Database.” Accessed Nov. 4, 2020. https://en.wikipedia.org/wiki/Oracle_Database.
 10. “MySQL.” Accessed Nov. 4, 2020. <https://en.wikipedia.org/wiki/MySQL>.

1.2 Hardware and Software

Computers can perform calculations and make logical decisions phenomenally faster than human beings can. Today’s personal computers and smartphones can perform billions of calculations in one second—more than a human can perform in a lifetime. **Supercomputers** already perform *thousands of trillions (quadrillions)* of instructions per second! As of December 2020, Fujitsu’s Fugaku¹¹ is the world’s fastest supercomputer—it can perform 442 quadrillion calculations per second (442 *petaflops*)!¹² To put that in perspective, this supercomputer *can perform in one second almost 58 million calculations for every person on the planet!*¹³ And supercomputing upper limits are growing quickly.

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

A computer consists of various physical devices referred to as hardware—such as the keyboard, screen, mouse, solid-state disks, hard disks, memory, DVD drives and processing units. Computing costs are dropping dramatically due 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 computer 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 and computerized devices have become commodities.

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. Over the years, hardware costs have fallen rapidly.

For decades, every couple of years, computer processing power approximately doubled inexpensively. This remarkable trend often is called **Moore’s Law**, named for Gordon Moore, co-founder of Intel and the person who identified this trend in the 1960s. Intel is a leading manufacturer of the processors in today’s computers and **embedded systems**, such as smart home appliances, home security systems, robots, intelligent traffic intersections and more.

11. “Top 500.” Accessed December 24, 2020. https://en.wikipedia.org/wiki/TOP500#TOP_500.

12. “Flops.” Accessed November 1, 2020. <https://en.wikipedia.org/wiki/FLOPS>.

13. For perspective on how far computing performance has come, consider this: In his early computing days in the 1960s, Harvey Deitel used the Digital Equipment Corporation PDP-1 (<https://en.wikipedia.org/wiki/PDP-1>), which was capable of performing only 93,458 operations per second, and the IBM 1401 (<http://www.ibm-1401.info/1401GuidePosterV9.html>), which performed only 86,957 operations per second.

Key executives at computer-processor companies NVIDIA and Arm have indicated that Moore’s Law no longer applies.^{14,15} Computer processing power continues to increase but relies on new processor designs, such as multicore processors (Section 1.2.2).

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 hard disks and solid-state drive storage) they have to hold programs and data, and
- their processor speeds—that is, the speeds at which computers **execute** programs to do their work.

Similar growth has occurred in the communications field. Costs have plummeted as enormous demand for communications **bandwidth** (that is, 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**.

1.2.2 Computer Organization

Regardless of physical differences, computers can be envisioned as divided into various **logical units** or sections.

Input Unit

This “receiving” section obtains information (data and computer programs) from **input devices** and places it at the other units’ disposal for processing. Computers receive most user input through keyboards, touch screens, mice and touchpads. Other forms of input include:

- receiving voice commands,
- scanning images and barcodes,
- reading data from secondary storage devices (such as solid-state drives, hard drives, Blu-ray Disc™ drives and USB flash drives—also called “thumb drives” or “memory sticks”),
- receiving video from a webcam,
- receiving information from the Internet (such as when you stream videos from YouTube® or download e-books from Amazon),
- receiving position data from a GPS device,
- receiving motion and orientation information from an **accelerometer** (a device that responds to up/down, left/right and forward/backward accelera-

14. “Moore’s Law turns 55: Is it still relevant?” Accessed November 2, 2020. <https://www.techrepublic.com/article/moores-law-turns-55-is-it-still-relevant/>.

15. “Moore’s Law is dead: Three predictions about the computers of tomorrow.” Accessed November 2, 2020. <https://www.techrepublic.com/article/moores-law-is-dead-three-predictions-about-the-computers-of-tomorrow/>.

tion) in a smartphone or wireless game controllers, such as those for Microsoft® Xbox®, Nintendo Switch™ and Sony® PlayStation®, and

- receiving voice input from intelligent assistants like Apple Siri®, Amazon Alexa® and Google Home®.

Output Unit

This “shipping” section takes information the computer has processed and places it on various **output devices** to make it available outside the computer. Most information that’s output from computers today is

- displayed on screens,
- printed on paper (“going green” discourages this),
- played as audio or video on smartphones, tablets, PCs and giant screens in sports stadiums,
- transmitted over the Internet, or
- used to control other devices, such as self-driving cars (and **autonomous vehicles** in general), robots and “intelligent” appliances.

Information is also commonly output to secondary storage devices, such as solid-state drives (SSDs), hard drives, USB flash drives and DVD drives. Popular recent forms of output are smartphone and game-controller vibration, virtual reality devices like Oculus Rift®, Oculus Quest®, Sony® PlayStation® VR and Samsung Gear VR®, and mixed reality devices like Magic Leap® One and Microsoft HoloLens™.

Memory Unit

This rapid-access, relatively low-capacity “warehouse” section retains information 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 8 to 16 GB is most common. GB stands for gigabytes; a **gigabyte** is approximately one billion bytes. A **byte** is eight bits. A **bit** (short for “*binary digit*”) is either a 0 or a 1.

Arithmetic and Logic Unit (ALU)

This “manufacturing” section performs calculations (e.g., addition, subtraction, multiplication and division) and makes decisions (e.g., comparing two items from the memory unit to determine whether they’re equal). In today’s systems, the ALU is part of the next logical unit, the CPU.

Central Processing Unit (CPU)

This “administrative” section coordinates and supervises the operation of the other sections. The CPU tells

- the input unit when to read information into the memory unit,
- the ALU when to use information from the memory unit in calculations, and
- the output unit when to send information from the memory unit to specific output devices.

Most computers today have **multicore processors** that economically implement multiple processors on a single integrated circuit chip. Such processors can perform many operations simultaneously. A **dual-core processor** has two CPUs, a **quad-core processor** has four and an **octa-core processor** has eight. Intel has some processors with up to 72 cores.

Secondary Storage Unit

This is the long-term, high-capacity “warehousing” section. Programs and data not actively being used by the other units are placed on secondary storage devices 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 byte is much less. Examples of secondary storage devices include solid-state drives (SSDs), USB flash drives, hard drives and read/write Blu-ray drives. Many current drives hold terabytes (TB) of data. A **terabyte** is approximately one trillion bytes. Typical desktop and notebook-computer hard drives hold up to 4 TB, and some recent desktop-computer hard drives hold up to 20 TB.¹⁶ The largest commercial SSD holds up to 100 TB (and costs \$40,000).¹⁷

✓ Self Check

1 (Fill-In) For many decades, every year or two, computers’ capacities have approximately doubled inexpensively. This remarkable trend often is called _____.

Answer: Moore’s Law.

2 (True/False) Information in the memory unit is persistent—it’s preserved even when the computer’s power is turned off

Answer: False. Information in the memory unit is volatile—it’s typically lost when the computer’s power is turned off.

3 (Fill-In) Most computers today have _____ processors that implement multiple processors on a single integrated-circuit chip. Such processors can perform many operations simultaneously.

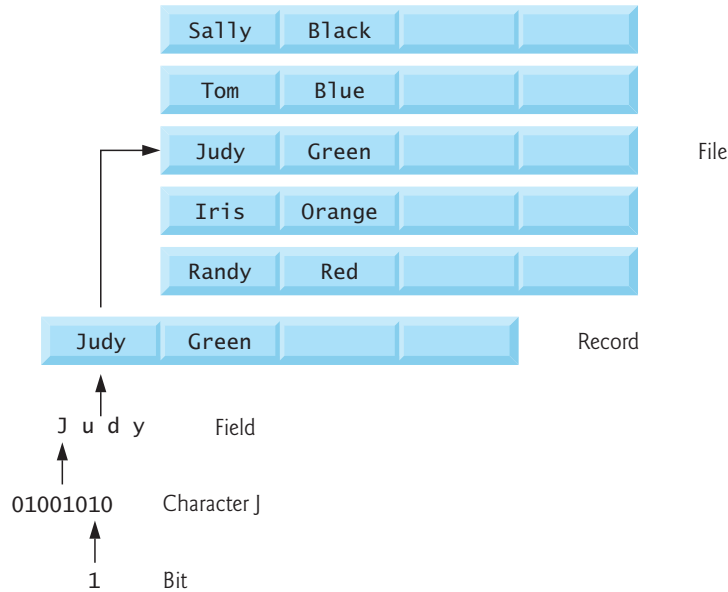
Answer: multicore.

16. “History of hard disk drives.” Accessed November 1, 2020. https://en.wikipedia.org/wiki/History_of_hard_disk_drives.

17. “At 100TB, the world’s biggest SSD gets an (eye-watering) price tag.” Accessed November 1, 2020. <https://www.techradar.com/news/at-100tb-the-worlds-biggest-ssd-gets-an-eye-watering-price-tag>.

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. The following diagram illustrates a portion of the data hierarchy:



Bits

A bit is short for “*binary digit*”—a digit that can assume one of *two* values—and is a computer’s smallest data item. It can have the value 0 or 1. Remarkably, computers’ impressive functions 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). Bits form the basis of the binary number system, which we discuss in our “Number Systems” appendix.

Characters

Work with data in the low-level form of bits is tedious. Instead, people prefer to work with **decimal digits** (0–9), **letters** (A–Z and a–z) and **special symbols** such as

\$ @ % & * () - + " : ; , ? /

Digits, letters and special symbols are known as **characters**. The computer’s **character set** contains the characters used to write programs and represent data items. Computers process only 1s and 0s, so a computer’s character set represents each character as a pattern of 1s and 0s. C uses the **ASCII (American Standard Code for Information Interchange)** character set by default. C also supports **Unicode[®]** characters composed of one, two, three or four bytes (8, 16, 24 or 32 bits, respectively).¹⁸

18. “Programming with Unicode.” Accessed November 1, 2020. https://unicodebook.readthedocs.io/programming_languages.html.

Unicode contains characters for many of the world's languages. ASCII is a (tiny) subset of Unicode representing letters (a–z and A–Z), digits and some common special characters. You can view the ASCII subset of Unicode at

<https://www.unicode.org/charts/PDF/U0000.pdf>

For the lengthy Unicode charts for all languages, symbols, emojis and more, visit

<http://www.unicode.org/charts/>

Fields

Just as characters are composed of bits, **fields** are composed of characters or bytes. A field is a group of characters or bytes that conveys meaning. For example, a field consisting of uppercase and lowercase letters could represent a person's name, and a field consisting of decimal digits could represent a person's age in years.

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 group of characters).
- Address (a group 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. All the fields listed above 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. More generally, a file contains arbitrary data in arbitrary formats. Some operating systems view a file 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. You'll see how to do that in Chapter 11, File Processing. It's not unusual for an organization to have many files, some containing billions, or even trillions, of characters of information. As we'll see below, with big data, far larger file sizes are becoming increasingly common.

Databases

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 informa-