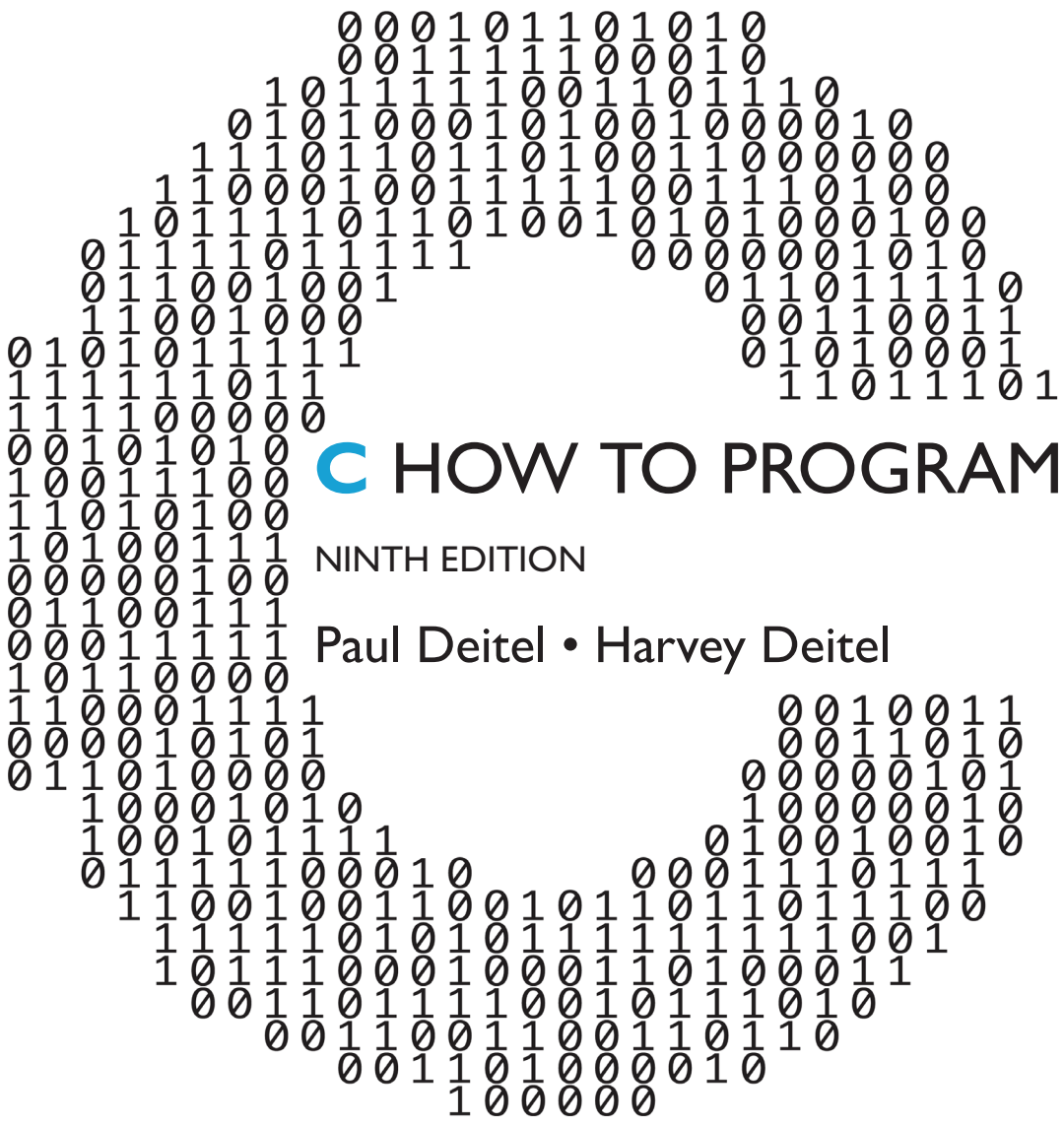
GLOBAL EDITION



C HOW TO PROGRAM

NINTH EDITION

Paul Deitel • Harvey Deitel

with case studies introducing

Applications Programming and

Systems Programming

DEITEL®

HOW TO PROGRAM

NINTH EDITION

GLOBAL EDITION

with

Case Studies Introducing

Applications Programming and

Systems Programming

PAUL DEITEL
HARVEY DEITEL

Credits and acknowledgments borrowed from other sources and reproduced, with permission, in this textbook appear on the appropriate page of appearance or in the Credits on pages.

Cover image by Ink Drop/ Shutterstock

Pearson Education Limited
KAO Two
KAO Park
Hockham Way
Harlow
Essex
CM17 9SR
United Kingdom

and Associated Companies throughout the world

Visit us on the World Wide Web at: www.pearsonglobaleditions.com

Authorized adaptation from the United States edition, entitled C How to Program*, 9th Edition, ISBN 978-0-13-739839-3 by Paul Deitel and Harvey Deitel published by Pearson Education © 2022.*

ISBN 10: 1-292-43707-3 (print)
ISBN 13: 978-1-292-43707-1 (print)
eBook ISBN 13: 978-1-292-43699-9 (uPDF)

British Library Cataloguing-in-Publication Data
A catalogue record for this book is available from the British Library

1 22

Typeset in Times NR MT Pro by B2R Technologies Pvt. Ltd.

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

Paul and Harvey Deitel

Trademarks

Contents

Appendices E–H are PDF documents posted online at the book's Companion Website (located at `https://www.pearsonglobaleditions.com`).

Online Appendices

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://www.pearsonglobaleditions.com
```

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 walkthrough of the enormously popular RSA algorithm's steps, providing hints to help you build a working, simple, small-scale implementation.
- We've enhanced existing case studies and added new ones focusing on AI and data science, including simulations with random-number generation, survey data analysis, natural language processing (NLP) and artificial intelligence (machine-learning with simple linear regression). Data science is emphasized in the latest ACM/IEEE computing curricula documents.
- We've added exercises in which students use the Internet to research **ethics** and **privacy** issues in computing.
- We tuned our **mutltithreading and multicore performance** case study. We also show a **performance icon in the margin** whenever we discuss a performance-related issue in the text.

- 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.

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
- **Game Simulation**—Rock, Paper, Scissors
- **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 1: 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—**Rock, Paper, Scissors**, 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, `typedef`s 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.

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)`. 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 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 32,000[13] 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 languages[16]—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/`.