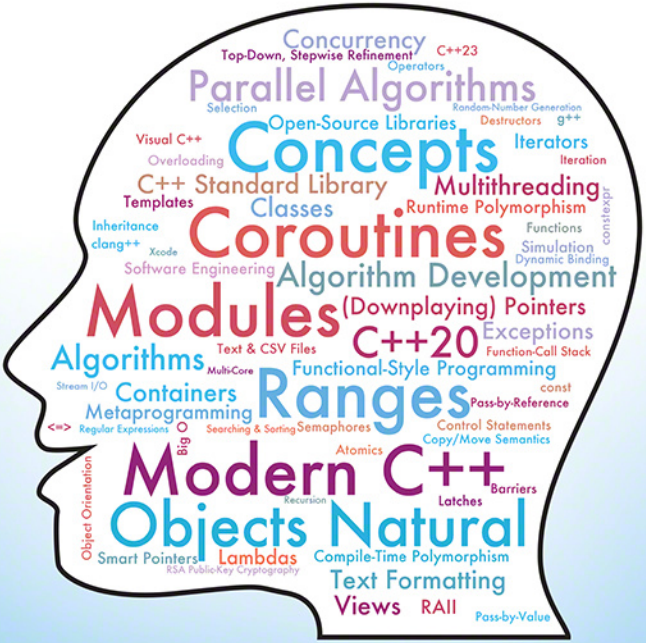
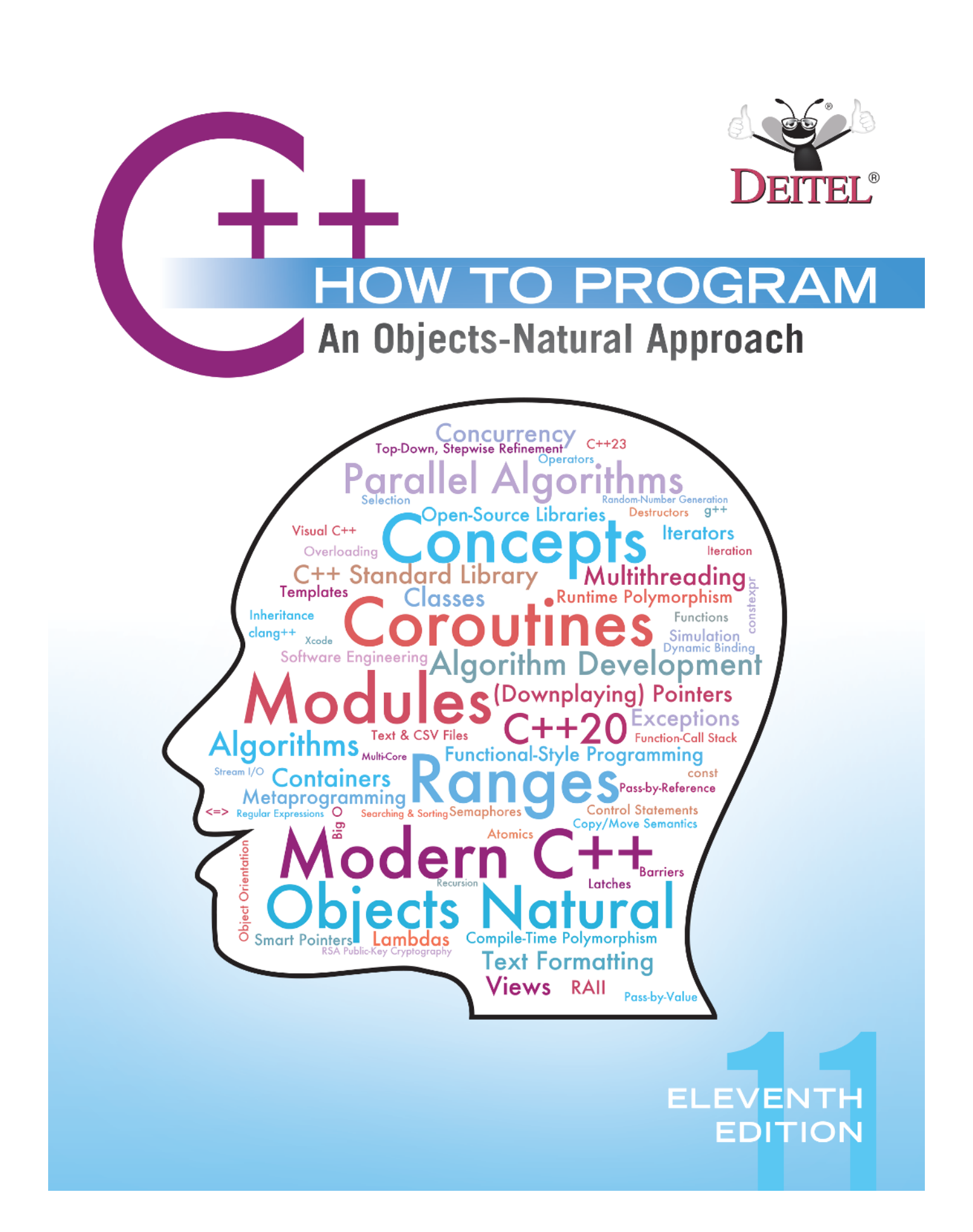
C++ HOW TO PROGRAM

An Objects-Natural Approach

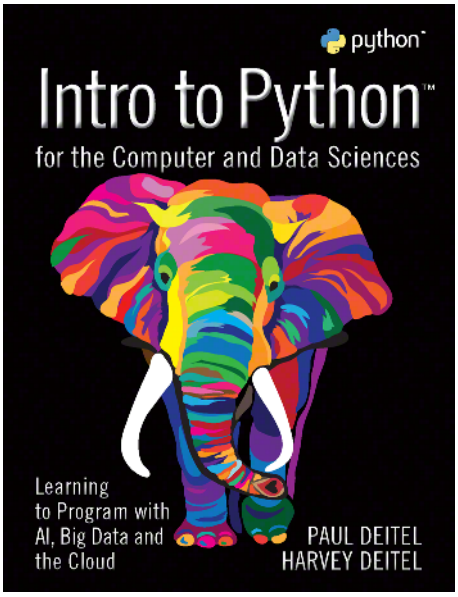


PAUL DEITEL
HARVEY DEITEL

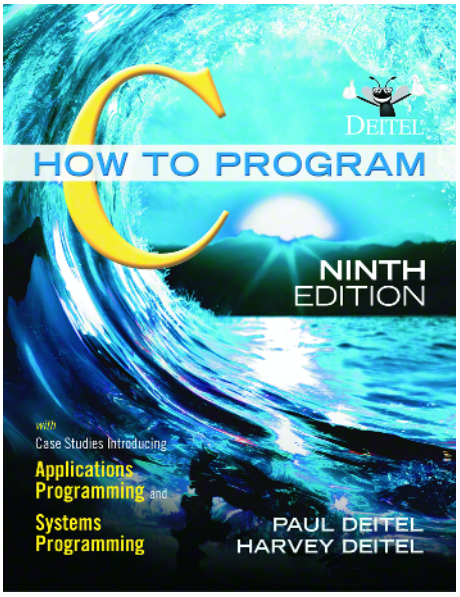
ELEVENTH EDITION


C++
HOW TO PROGRAM
An Objects-Natural Approach
DEITEL®
Concurrency
Top-Down, Stepwise Refinement
C++23
Operators
Parallel Algorithms
Selection
Random-Number Generation
Destructors
g++
Open-Source Libraries
Visual C++
Concepts
Iterators
Overloading
Iteration
C++ Standard Library
Multithreading
Templates
Classes
Runtime Polymorphism
constexpr
Inheritance
clang++
Xcode
Coroutines
Functions
Simulation
Dynamic Binding
Software Engineering
Algorithm Development
Modules
(Downplaying) Pointers
C++20
Exceptions
Algorithms
Text & CSV Files
Function-Call Stack
Multi-Core
Functional-Style Programming
Stream I/O
Containers
Ranges
const
Pass-by-Reference
Metaprogramming
<=>
Regular Expressions
Searching & Sorting
Semaphores
Control Statements
Big O
Copy/Move Semantics
Atomics
Modern C++
Object Orientation
Barriers
Recursion
Latches
Objects Natural
Smart Pointers
Lambdas
Compile-Time Polymorphism
RSA Public-Key Cryptography
Text Formatting
Views
RAII
Pass-by-Value
ELEVENTH
EDITION

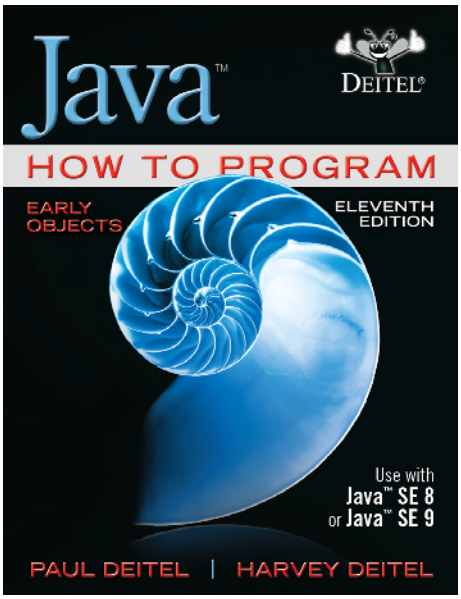
Recent Deitel Textbooks



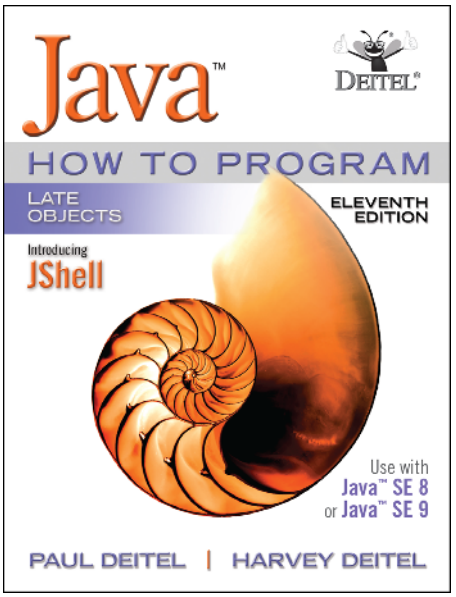
https://deitel.com/pycds



https://deitel.com/chtp9



https://deitel.com/jhtp11EOV

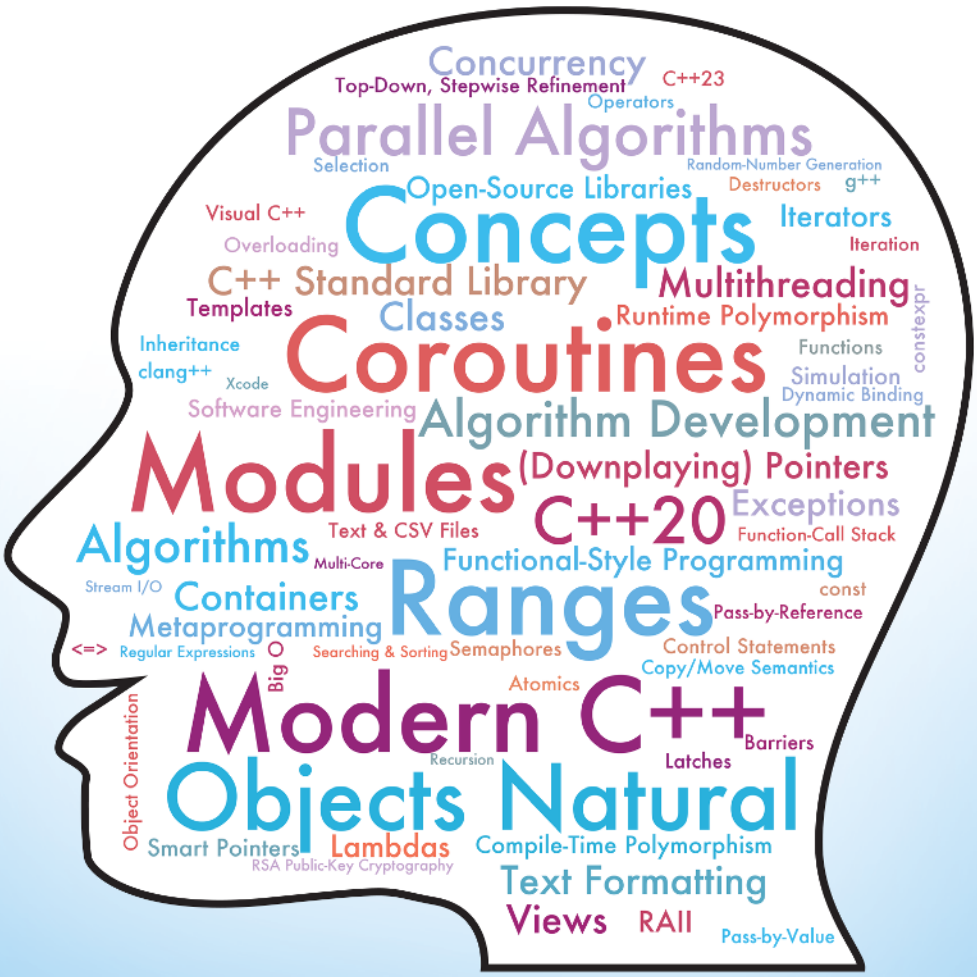


https://deitel.com/jhtp11LOV

PAUL DEITEL
Deitel & Associates, Inc.
HARVEY DEITEL
Deitel & Associates, Inc.

C++ HOW TO PROGRAM

An Objects-Natural Approach



Pearson

ELEVENTH EDITION

Content Management: Tracy Johnson
Content Production: Bob Engelhardt, Abhijeet Gope, K Madhusudhan
Product Marketing: Krista Clark
Rights and Permissions: Anjali Singh
Cover Designer: Paul Deitel, Harvey Deitel, Chuti Prasertsith
Cover Art: Paul Deitel (produced using Andreas Müller's open-source Python library `word_cloud`—`https://github.com/amueller/word_cloud`)

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

Library of Congress Cataloging-in-Publication Data
On file

uPDF
ISBN-10: 0-13-809242-7
ISBN-13: 978-0-13809242-9

ePub
ISBN-10: 0-13-809236-2
ISBN-13: 978-0-13809236-8

In memory of Gordon Moore, co-founder of Intel and father of Moore's Law:

For a career devoted to improving people's lives through technology, innovation, industry, entrepreneurship and philanthropy.

Paul and Harvey Deitel

Pearson's Commitment to Diversity, Equity, and Inclusion

Pearson is dedicated to creating bias-free content that reflects the diversity, depth, and breadth of all learners' lived experiences. We embrace the many dimensions of diversity, including but not limited to race, ethnicity, gender, sex, sexual orientation, socioeconomic status, ability, age, and religious or political beliefs.

Education is a powerful force for equity and change in our world. It has the potential to deliver opportunities that improve lives and enable economic mobility. As we work with authors to create content for every product and service, we acknowledge our responsibility to demonstrate inclusivity and incorporate diverse scholarship so that everyone can achieve their potential through learning. As the world's leading learning company, we have a duty to help drive change and live up to our purpose to help more people create a better life for themselves and to create a better world.

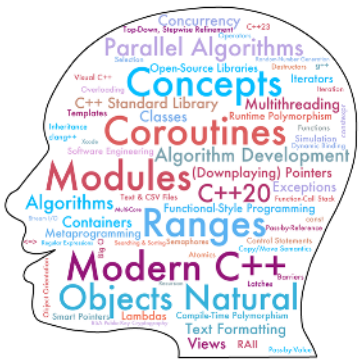
Our ambition is to purposefully contribute to a world where:

- Everyone has an equitable and lifelong opportunity to succeed through learning.
- Our educational products and services are inclusive and represent the rich diversity of learners.
- Our educational content accurately reflects the histories and lived experiences of the learners we serve.
- Our educational content prompts deeper discussions with students and motivates them to expand their own learning (and worldview).

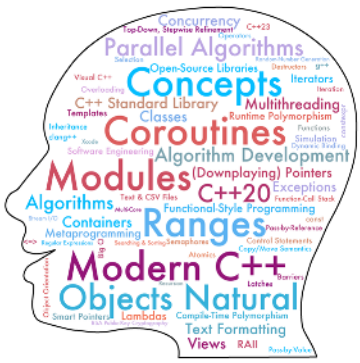
We are also committed to providing products that are fully accessible to all learners. As per Pearson's guidelines for accessible educational Web media, we test and retest the capabilities of our products against the highest standards for every release, following the WCAG guidelines in developing new products for copyright year 2022 and beyond. You can learn more about Pearson's commitment to accessibility at `https://www.pearson.com/us/accessibility.html`.

While we work hard to present unbiased, fully accessible content, we want to hear from you about any concerns or needs with this Pearson product so that we can investigate and address them.

- Please contact us with concerns about any potential bias at `https://www.pearson.com/report-bias.html`.
- For accessibility-related issues, such as using assistive technology with Pearson products, alternative text requests, or accessibility documentation, email the Pearson Disability Support team at `disability.support@pearson.com`.



Contents



Preface

1 An Innovative Modern C++ Programming Textbook

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

Welcome to ***C++ How to Program: An Objects-Natural Approach, 11/e***. We present a friendly, contemporary, code-intensive, case-study-oriented introduction to C++, the world's third most popular programming language according to the TIOBE Index.[2]

C++ is popular for building high-performance business-critical and mission-critical computing systems—operating systems, real-time systems, embedded systems, game systems, banking systems, air-traffic-control systems, communications systems and more. This book is an introductory- through intermediate-level college textbook presentation of the C++20 version of C++ and its associated standard libraries, with a look toward C++23 and C++26. In this Preface, we present the "soul of the book."

Live-Code Approach and Getting the Code

At the heart of the book is the Deitel signature **live-code approach**. Rather than code snippets, we show C++ as it's intended to be used in the context of 255 complete, working, real-world C++ programs with live outputs.

Read the Before You Begin section that follows this Preface to learn how to set up your Windows, macOS or Linux computer to run the code examples. For your convenience, we provide the book's examples in C++ source-code (`.cpp` and `.h`) files for use with integrated development environments and command-line compilers. All the source code is available for download at

- `https://github.com/pdeitel/CPlusPlusHowToProgram11e`
- `https://www.deitel.com/cpphtp11`

Chapter 1's Test-Drives (Section 1.11) shows how to compile and run the code examples with each of our preferred compilers. Executing each program in parallel with reading the text will make your learning experience "come alive." If you encounter a problem, you can reach us at `deitel@deitel.com`, and we'll respond promptly.

---

1. Martin Fowler (with contributions by Kent Beck). *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Professional. 2018.
2. Tiobe Index for February 2023. Accessed March 8, 2023. `https://www.tiobe.com/tiobe-index/`.

Key Computing Trends

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.

The Internet of Things (IoT) already connects tens of billions of computerized devices of every imaginable type, and that number is likely to grow quickly. These generate enormous volumes of data (one form of "big data") at rapidly increasing speeds and quantities. And most computing will eventually be performed 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 50 more significant case study examples and case study exercises will ease you into 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.

2 Modern C++

We cover Modern C++—C++20, C++17, C++14 and C++11—with a look toward key features coming in C++23 and anticipated for C++26. We employ industry best practices, emphasizing Modern C++ idioms—which change how developers write C++ programs—while focusing on performance, security and software engineering. We present rich treatments of C++20's "big four" features—ranges, concepts, modules and coroutines. We'll say more about these in Section 6 of this Preface.

3 Target Audiences

The book's modular architecture (see the diagram on the next page) 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 the latest Modern C++ idioms to prepare for upcoming projects.

C++ How to Program: An Objects-Natural Approach, 11/e

by Paul Deitel & Harvey Deitel

PART 1
C++20 Fundamentals Quickstart & Procedural Programming

1. Intro: Test-Driving Popular, Free C++ Compilers
Intro to Hardware, Software & Internet; Test-Driving the Visual C++, GNU g++ and LLVM clang++ compilers.

2. Intro to C++20 Programming
C++ fundamentals. "Objects-Natural" (ON) approach intro—using libraries to build powerful object-oriented applications with few lines of code.
ON: Manipulating `string` Objects

3. Control Statements, Part 1
Intro to C++20 text formatting.
ON: Super-Sized Integers with the Boost Multiprecision Library

4. Control Statements, Part 2
ON: Precise Monetary Calculations with the Boost Multiprecision Library

5. Functions and an Intro to Function Templates
ON: Lnfylun Lhqtomh Wjtz Qarcv: Qjwazkrplm xzz Xndmwwqhlz (encrypted title for our private-key cryptography case study)

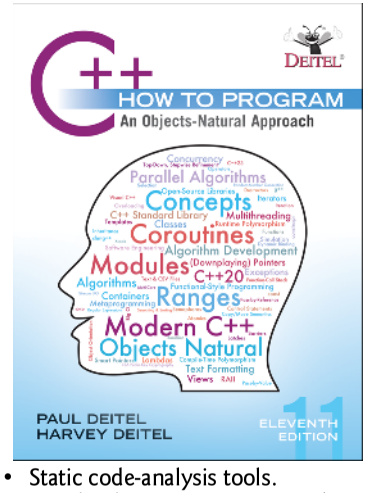
- ON = objects-natural case study.
- C++20's "Big Four" features: Ranges, Concepts, Modules and Coroutines.
- Live-code approach: 255 complete programs with live outputs.
- Communicate with the authors at **deitel@deitel.com**.
- Download source code at **https://deitel.com/cpphtp11**.

PART 2
Containers, C++20 Ranges, Pointers, Strings & Files

6. arrays, vectors, Ranges and Functional-Style Programming
Intro to functional-style programming.
ON: Class Template `vector`

7. (Downplaying) Pointers in Modern C++
Security & safe programming.
ON: C++20 `span`s

8. strings, string_views, Text Files, CSV Files and Regex
ON: Reading and Analyzing the Titanic Disaster Data (CSV)
ON: Intro to Regular Expressions



- Static code-analysis tools.
- Use developer resources: GitHub®, StackOverflow®, open-source, more.

PART 3
Modern Object-Oriented Programming & Exceptions

9. Custom Classes
ON: Studying the Vigenère Secret-Key Cipher Implementation

10. OOP: Inheritance and Runtime Polymorphism
Programming to an interface,

11. Operator Overloading, Copy/Move Semantics, Smart Pointers and RAII
Crafting valuable classes: Custom `MyArray` class, C++20 three-way comparison operator `<=>`, resource management via RAII (Resource Acquisition Is Initialization)

12. Exceptions and a Look Forward to Contracts

PART 4, Generic Programming: Templates, Concepts & Template Metaprogramming

13. Standard Library Containers and Iterators
Manipulating standard data structures

14. Standard Library Algorithms and C++20 Ranges & Views
Functional-style programming

15. Templates, C++20 Concepts and Metaprogramming
Compile-time polymorphism, function templates, C++20 abbreviated function templates, class templates, variadic templates, fold expressions

PART 5, Advanced Topics: Modules, Parallel Algorithms, Concurrency & Coroutines

16. C++20 Modules: Large-Scale Development
`import`, header units, module declarations, module fragments, partitions

17. Parallel Algorithms & Concurrency: A High-Level View
Multi-core performance with C++17 parallel algorithms, concurrency, multithreading

18. C++20 Coroutines
`co_yield`, `co_await`, `co_return`, coroutines support libraries, generators, executors and tasks

PART 6
Miscellaneous Topics

19. Stream I/O and C++20 Text Formatting

20. Other Topics and a Look Toward C++23 and C++26

21. Computer Science Thinking: Searching, Sorting and Big O

- Programming tips: C++ Core Guidelines, Software Engineering, Performance, Security, Errors, C++20 Modules, C++20 Concepts, Data Science.
- g++ & clang++ Docker containers.
- A look toward C++23 and C++26.
- Blog: **https://deitel.com/blog**.

4 "Objects-Natural" Learning Approach

Traditionally object-oriented programming textbooks have been designated as "late objects" or "early objects." What's really "late" or "early" in these textbooks is not "objects." Rather, it's teaching how to develop custom classes—the blueprints from which objects are built. We've written textbooks using both of these approaches.

What Is "Objects Natural?"

As we wrote our Python textbook,[3] we noticed that although our presentation fit the "late objects" model, it was actually something more—and *that* something is special. We call it the "objects-natural approach," and we're now applying it to C++ and the other object-oriented programming languages we write about.

Similar to "late objects," our objects-natural approach begins with programming fundamentals, but you'll work extensively in the early chapters with easy-to-use powerful pre-existing classes that do significant things. You'll quickly create objects of those classes (typically with one line of code) and tell them to "strut their stuff" with a minimal number of simple C++ statements.

Even if you're a programming novice, you can perform significant tasks long before you learn how to create custom C++ classes in Chapter 9. This is one of the most compelling aspects of working with a mature object-oriented language like C++. After covering programming fundamentals with the objects-natural approach, we provide a deep treatment of object-oriented programming beginning with a rich treatment of custom class creation.

An Abundance of Free Classes

We emphasize using the massive number of valuable free classes in the C++ ecosystem. These typically come from:

- the C++ standard library,
- platform-specific libraries, such as those provided with Microsoft Windows, Apple macOS or various Linux versions, and
- free third-party C++ libraries, often created by the open-source community.

We encourage you to view lots of free, open-source C++ code available on sites like GitHub. Reading other programmers' code is a great way to learn.

The Boost Project

Boost provides 168 powerful open-source C++ libraries[4] and serves as a "breeding ground" for new capabilities that might eventually be incorporated into the C++ standard libraries. The following StackOverflow post lists Modern C++ libraries and language features that evolved from the Boost libraries:[5]

```
https://stackoverflow.com/a/8852421
```

3. *Intro to Python for Computer Science and Data Science: Learning to Program with AI, Big Data and the Cloud* (`https://deitel.com/pycds`).
4. "Boost 1.81.0 Library Documentation." Accessed March 8, 2023. `https://www.boost.org/doc/libs/1_81_0/`.
5. Kennytm, Answer to "Which Boost Features Overlap with C++11?" Accessed March 8, 2023. `https://stackoverflow.com/a/8852421`.

We use the Boost Multiprecision library in our **objects-natural case studies** on **super-sized integers** (Section 3.14) and **precise monetary calculations** (Section 4.14).

Objects-Natural Case Studies

Chapter 1 presents a friendly introduction to the basic concepts and terminology of object technology. In the early chapters, you'll create and use objects of preexisting classes long before Chapter 9 discusses how to create custom classes. See Section 6's Tour of the Book for descriptions of our objects-natural case studies in Chapters 2–9. A perfect example of the objects-natural approach is using objects of standard library classes, like `array` and `vector` (Chapter 6), without knowing how to write classes in general or how those classes are implemented in particular. Throughout the rest of the book, we use C++ standard library capabilities extensively.

5 Programming Wisdom and Key C++20 Features

We integrate smoothly into the flow of the text software-development wisdom, data science topics, C++20 modules and C++20 concepts features:

- **Software engineering observations** highlight architectural and design issues for proper software construction, especially for larger systems. 

- **Security best practices** help you strengthen your programs against attacks. 

- **Performance tips** highlight opportunities to make your programs run faster or minimize the amount of memory they occupy. 

- **Common programming errors** help reduce the likelihood that you'll make the same mistakes. 

- **C++ Core Guidelines recommendations** (introduced in Section 12). 

- **C++20's new modules features**. 

- **C++20's new concepts features**. 

- We present and use **data science topics** in several examples and exercises. 

6 Tour of the Book

The one-page Table of Contents diagram earlier in this Preface provides a high-level overview of the book's modular architecture from "40,000 feet." We recommend that you refer to that diagram as you read this section.

The early chapters establish a solid foundation in C++20 fundamentals. The mid-range to high-end chapters introduce Modern C++ software development. We discuss C++'s programming models:

- procedural programming,
- functional-style programming,
- object-oriented programming,
- generic programming and
- template metaprogramming.

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, this detailed Tour of the Book will help you make the best decisions.

Part 1: Programming Fundamentals Quickstart

Chapter 1, Intro and Test-Driving Popular, Free C++ Compilers, engages programming novices with intriguing facts and figures to excite them about studying computers and computer programming. The chapter includes current technology trends, hardware, software and Internet concepts, and a sample data hierarchy from bits to bytes, fields, records and databases. It lays the groundwork for the C++ programming discussions in Chapters 2–21 and the substantial integrated case study examples, exercises and projects.

We discuss the programming-language types and technologies you'll likely use as you develop software. We introduce the C++ standard library—existing, reusable, top-quality, high-performance capabilities that 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 three of the most popular C++ development environments:

- Microsoft's Visual C++ in Visual Studio on Windows,
- GNU's `g++` on macOS/Linux and
- The LLVM Compiler Infrastructure's `clang++` on macOS/Linux.

We tested the book's 255 code examples using each compiler.[6] Choose whichever you prefer—the book also works well with many others. See the Before You Begin section that follows this Preface for compiler installation instructions.

We also demonstrate running `g++` and `clang++` using Docker containers. Docker is an important tool that enables you to run the latest versions of these compilers on Windows, macOS or Linux. See Section 7 of this Preface for more details on Docker and Docker containers.

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 computer-science and data-science. AI and data science will likely 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, arithmetic operators and their precedence, and decision-making. As part of our objects-natural approach, Section 2.8's **objects-natural case study** demonstrates **creating and using objects of the C++ standard library class `string`**—without you having to know how to develop custom classes in general or how the large complex class `string` is implemented in particular).

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

Chapter 3, Algorithm Development and Control Statements: Part 1, 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—Counter-Controlled Iteration**, **Sentinel-Controlled Iteration** and **Nested Control Statements**. Section 3.14's **objects-natural case study** demonstrates **using the open-source Boost Multiprecision library's `cpp_int` class to create super-sized integers**.

Chapter 4, Control Statements: Part 2, 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**. We introduce C++20's `format` function, which provides powerful new text-formatting capabilities. Pre-C++20 text formatting is complex and verbose. The `format` function greatly simplifies data formatting using a concise syntax based on the Python programming language's text formatting. We present a few C++20 text-formatting features throughout the book, then take a deeper look in Chapter 19. In introductory computer science courses, instructors may wish to present Section 19.9 after C++20 text formatting is introduced in Chapter 4. Section 4.14's **objects-natural case study** demonstrates **using the open-source Boost Multiprecision library's `cpp_dec_float_50` class for precise monetary calculations**.

Chapter 5, Functions and an Intro to Function Templates, introduces custom functions. We introduce random-number generation and simulation techniques and use them in our first of several **Random-Number Simulation case studies** throughout the book to **implement a popular casino dice game**. We discuss C++'s secure library of random-number capabilities that can produce "nondeterministic" random numbers—a set of random numbers that can't be predicted. Such random-number generators are used in simulations and security scenarios where predictability is undesirable. 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 rich treatment of the powerful computer science topic of recursion.

Section 5.19's **objects-natural case study** title—**Pqyoaf X Nylfomigrob Qwbbfmh Mndogvk: Rboqlrut yua Boklnxhmywex**—looks like gibberish. This is not a mistake! This case study continues our security emphasis by introducing **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. You'll use our implementation of the Vigenère secret-key cipher[7] algorithm to encrypt and decrypt messages and to decrypt this section's title. Then, in Chapter 9's **objects-natural case study**, you'll study our class that implements the **Vigenère secret-key cipher** using classes and array-processing techniques. In **Chapter 9 case study exercises**, you'll also explore far more secure **public-key cryptography with the RSA algorithm**.

7. "Vigenère Cipher." Accessed April 29, 2023. `https://en.wikipedia.org/wiki/Vigenère_cipher`.

Part 2: Arrays, Pointers and Strings

Chapter 6, `arrays`, `vectors`, Ranges and Functional-Style Programming, begins our early coverage of the C++ standard library's containers, iterators and algorithms. We present the C++ standard library's `array` container for representing lists and tables of values. You'll define and initialize `array`s and access their elements. We discuss passing `array`s to functions, sorting and searching `array`s and manipulating multidimensional `array`s.

Like many modern languages, C++ offers "functional-style" programming features. These can help you write more concise code that's less likely to contain errors and is easier to read, debug and modify. Chapter 6 begins our introduction to functional-style programming using `array`s, lambda expressions (anonymous functions), and C++20 ranges—a C++20 "big four" feature that simplifies how you call many of the C++ standard library's predefined algorithms. We continue that discussion in Chapters 13 and 14.

Section 6.15's **objects-natural case study** demonstrates the **C++ standard library class template `vector`**. Chapter 6 is essentially a large **objects-natural case study of both `array`s and `vector`s.** The code in this chapter is a good example of Modern C++ coding idioms. This chapter's exercises include a case study on the famous **Knight's Tour problem**, which we approach in various ways, including an AI strategy called **heuristic programming**.

Chapter 7, (Downplaying) Pointers in Modern C++, explains pointer concepts, such as declaring pointers, initializing pointers, getting the memory address of a variable, dereferencing pointers, pointer arithmetic and the close relationship among built-in pointers, pointer-based arrays and pointer-based strings, each of which C++ inherited from the C programming language. Pointers are powerful but challenging to work with. So, we focus on Modern C++ features that eliminate the need for most pointers and make your code more robust and secure, including references, "smart pointer" objects, `array`s and `vector`s, `string`s, C++20 `span`s and C++17 `string_view`s. We still cover built-in arrays because they remain useful in C++, and so you'll be able to read legacy C++ code that you'll encounter in industry. In new development, you should favor Modern C++ capabilities. Section 7.10's **objects-natural case study** demonstrates one such capability—**C++20 `span`s**. These enable you to view and manipulate elements of contiguous containers, such as pointer-based arrays and standard library `array`s and `vector`s, without using pointers directly.

In a **Random-Number Simulation case study exercise**, you'll implement the **famous race between the tortoise and the hare**. This chapter also contains the first of our two **systems programming case study exercises—Building Your Own Computer (as a virtual machine)**. In the context of several **case study 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 **virtual machines—one of the most important systems-architecture concepts in modern computing**. Chapter 13 contains the intimately related **systems programming case study exercise—Building Your Own Compiler**.

Chapter 8, `strings`, `string_views`, Text Files, CSV Files and Regex, presents many of the standard library `string` class's features; shows how to write text to and read text from both plain text files and comma-separated values (CSV) files (popular for representing data science datasets); and introduces string pattern matching with the standard library's regular-expression capabilities. C++ offers two types of strings—`string` objects and C-style pointer-based strings. We use `string` objects to make programs more robust and eliminate many of the security problems of C strings. In new development, you should favor `string` objects. We also present C++17's `string_views`—a lightweight, flexible mechanism for passing any type of string to a function. This chapter presents two **objects-natural case studies**:

- Section 8.19 introduces data analytics by reading and **analyzing a CSV file containing the Titanic Disaster dataset**.
- Section 8.20 introduces **regular-expression pattern matching and text replacement**.

This chapter includes three **AI/Data Science case study exercises**. In the first case study, **Machine Learning with Simple Linear Regression: Statistics Can Be Deceiving**, you'll learn that an essential aspect of data analytics is "getting to know your data." One way to do this is via descriptive statistics—but these can be deceiving. To illustrate this, we'll consider visualizations of **Anscombe's Quartet**[8] (Exercise 8.40)—a famous example of four dramatically different datasets containing *x*–*y* coordinate pairs with nearly identical descriptive statistics. You'll then study a **completely coded example** in which we use the popular **AI/machine-learning** statistical technique called **simple linear regression** that, given a collection of *x*–*y* coordinate pairs representing an **independent variable** (*x*) and a **dependent variable** (*y*), determines the equation of a straight line ($y = mx + b$) that most closely fits the data. This equation describes the relationship between the dependent and independent variables, enabling us to predict *y*'s value for any given *x*. It also allows us to plot a **regression line**. You'll see that the **regression lines for Anscombe's Quartet are visually identical** for all four dramatically different datasets. The code you'll study uses the popular **open-source gnuplot package** to create attractive **visualizations** of Anscombe's Quartet. The gnuplot package uses its own plotting language, different from C++, so we provide extensive code comments that explain the gnuplot commands.

In the **AI/Data Science case study exercise, Machine Learning with Simple Linear Regression: Time Series Analysis** (Exercise 8.41), you'll use what you learned in the preceding exercise to analyze a **time series**, a sequence of values (called **observations**) associated with points in time. Time series examples include daily closing stock prices, hourly temperature readings, the changing positions of a plane in flight, annual crop yields and quarterly company profits. You'll run a **simple linear regression** on **126 years of New York City average January temperature data** (stored in a CSV file) and **use gnuplot to plot the data and the regression line** so you can determine if there is a cooling or warming trend.

This chapter's final **AI/Data Science case study** (Exercise 8.42) presents an intro to **similarity detection with very basic natural language processing (NLP)**—an important data science and artificial intelligence topic. NLP helps computers understand, analyze and process text. While writing this book, we used the paid (NLP) tool Grammarly[9] to help tune the writing and ensure the text's readability for a broad audience. Some people believe

8. "Anscombe's quartet." Accessed March 8, 2023. `https://en.wikipedia.org/wiki/Anscombe%27s_quartet`.

that the works of William Shakespeare actually might have been penned by Christopher Marlowe or Sir Francis Bacon, among others.[10,11] In this 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***. You'll determine how alike they are by comparing the statistics you calculate, such as the percentages of each unique word among all words in each play. You may be surprised by the results.

Part 3: Object-Oriented Programming

SE 

Chapter 9, Custom Classes, begins our substantially upgraded, multi-chapter, Modern C++, object-oriented programming treatment. C++ is extensible—each class you create becomes a new type you can use to create objects. In Chapters 9–11, you'll learn C++'s features for crafting valuable classes and manipulating objects of these classes.

In Section 9.22, we conclude our **objects natural case study track** by studying the **Vigenère Secret-Key Cipher class implementation** that we demonstrated in Chapter 5's objects-natural case study. **Objects-natural case study** Exercises 9.32–9.33 demonstrate **how to serialize objects with JSON (JavaScript Object Notation)**—a popular human-and-machine-readable data format commonly used to transmit data over the Internet.

In the context of several **Random-Number Simulation case study exercises**, you'll use `arrays` of `strings`, random-number generation and simulation techniques to implement

Sec 

a **text-based, card-shuffling-and-dealing program**. In a **Security and Cryptography case study exercise**, you'll also 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 secret keys 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, OOP: Inheritance and Runtime Polymorphism, focuses on the relationships among classes in an inheritance hierarchy and the powerful runtime polymorphic processing capabilities (for "programming in the general") that these relationships enable. In this chapter's **runtime-polymorphism case study, you'll implement an `Employee` class hierarchy in an application that performs polymorphic payroll calculations**.

An important aspect of this chapter is understanding how polymorphism works. **A key feature of the chapter is its detailed diagram and explanation of how C++ typically implements polymorphism, `virtual` functions and dynamic binding "under the hood."** You'll see that it can use an elegant pointer-based data structure. We also discuss programming to an interface, not an implementation. In Chapter 20, we discuss other more advanced mechanisms for achieving runtime polymorphism, including the non-virtual interface idiom (NVI) and `std::variant`/`std::visit`.

---

9. Grammarly has free and paid versions (`https://www.grammarly.com`). They provide free plug-ins you can use in several popular web browsers.
10. "Did Shakespeare Really Write His Own Plays?" Accessed November 13, 2020. `https://www.history.com/news/did-shakespeare-really-write-his-own-plays`.
11. "Shakespeare authorship question." Accessed November 13, 2020. `https://en.wikipedia.org/wiki/Shakespeare_authorship_question`.

Chapter 11, Operator Overloading, Copy/Move Semantics and Smart Pointers, shows how to enable C++'s existing operators to work with custom class objects and introduces smart pointers and dynamic memory management. Smart pointers help you avoid dynamic memory management errors and "resource leaks" by providing additional functionality beyond that of built-in pointers. We discuss `unique_ptr` in this chapter and `shared_ptr` and `weak_ptr` in Chapter 20, Other Topics and a Look Toward the Future of C++.

A key aspect of Chapter 11 is crafting valuable classes. We begin with a `string` class test-drive, presenting an elegant use of operator overloading before you implement your own customized class with overloaded operators. Then, in our **Crafting Valuable Classes case study—one of the book's most important examples**—you'll build your own custom `MyArray` class using overloaded operators and other capabilities to solve various problems with C++'s native pointer-based arrays.[12] We introduce and implement the five special member functions you can define in each class—the copy constructor, copy assignment operator, move constructor, move assignment operator and destructor. We discuss copy semantics and move semantics, which enable a compiler to move resources from one object to another to avoid costly, unnecessary copies. We introduce C++20's three-way comparison operator (`<=>`; also called the "spaceship operator") and show how to implement custom conversion operators. In Chapter 15, you'll convert a portion of the `MyArray` class into a class template that can store elements of a specified type. You will then have truly "crafted valuable classes."

Chapter 12, Exceptions and a Look Forward to Contracts, continues our exception-handling discussion that began in Chapter 6. We've enhanced Chapter 12's coverage with discussions of when to use exceptions, exception safety guarantees, and using exceptions in the context of constructors and destructors. We show how to handle dynamic memory allocation failures. We discuss why some libraries provide dual interfaces, enabling developers to choose whether to use versions of functions that throw exceptions or versions that set error codes. The chapter concludes with a **case study** that introduces contracts—a possible C++26 feature. **One goal of contracts is to make most functions `noexcept`—meaning they do not throw exceptions—which might enable the compiler to perform additional optimizations and eliminate the overhead and complexity of exception handling. Another goal is to find errors faster, eliminate them during development and, hopefully, create more robust code for deployment.** We introduce preconditions, postconditions and assertions, and we discuss how they can be implemented as contracts that are tested at execution time. We demonstrate the example code using GCC's experimental contracts implementation on `https://godbolt.org`.

Part 4: Standard Library Containers, Iterators and Algorithms

Chapter 13, Standard Library Containers and Iterators, begins our broader and deeper treatment of three key C++ standard library components:

- containers (templatized data structures),
- iterators (for traversing containers and accessing their elements) and
- algorithms (which use iterators to manipulate containers).

12. In industrial-strength systems, you'll use standard library classes for this, but this example enables us to go "under the hood" to demonstrate many key Modern C++ concepts.

We'll discuss containers, container adaptors and near containers. **You'll see that the C++ standard library provides commonly used data structures, so you do not need to create your own—the vast majority of your data structures needs can be fulfilled by reusing these standard library capabilities**. We demonstrate most standard library containers and introduce how iterators enable algorithms to be applied to various container types. We continue showing how C++20 ranges can simplify your code.

This chapter presents the second of our two **systems programming case study exercises—Building Your Own Compiler**. In the context of several exercises, you'll build a simple compiler that translates programs written in a small high-level programming language into our Simpletron Machine Language (SML). You'll write programs in this small new high-level language, compile them on the compiler you build, then run them on your Simpletron **virtual machine** you built in Chapter 7's **systems programming case study exercise—Building Your Own Computer**. And with Chapter 8's file-processing techniques, 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 systems-programming exercise sequence for novice computing students.

Chapter 14, Standard Library Algorithms and C++20 Ranges & Views, presents many of the standard library's 115 algorithms, focusing on the C++20 range-based algorithms, which are easier to use than their pre-C++20 versions. As you'll see, range-based algorithms specify their requirements using **C++20 concepts**—a C++20 "big four" feature that makes generic programming with templates more convenient and powerful. We briefly introduce C++20 concepts as needed for you to understand the requirements for working with these algorithms—Chapter 15 discusses concepts in more depth. Algorithms we present include filling containers with values, generating values, comparing elements or entire containers, removing elements, replacing elements, mathematical operations, searching, sorting, swapping, copying, merging, set operations, and calculating minimums and maximums. We discuss each algorithm's minimum iterator requirements so you can determine which containers can be used with each algorithm. We also continue our discussion of C++'s functional-style programming features with C++20 ranges and views.

Concepts

Part 5: Advanced Topics

Chapter 15, Templates, C++20 Concepts and Metaprogramming, presents our substantially enhanced treatment of compile-time (static) polymorphism, generic programming with templates, C++20 concepts and template metaprogramming. The importance of templates has increased with each new C++ release. **A major Modern C++ theme is to do more at compile-time for better type checking and better runtime performance—anything resolved at compile-time avoids runtime overhead and makes systems faster**. As you'll see, templates and especially template metaprogramming are the keys to powerful compile-time operations.

Perf

We demonstrate C++20's new template capabilities, including abbreviated function templates, templated lambdas and concepts. We introduce type traits for testing type attributes at compile-time. We show variadic function templates that receive a variable number of parameters and use fold expressions to conveniently apply an operation to all the arguments passed to a variadic template.

Concepts

A feature of this chapter is the **Crafting Valuable Classes case study**—you'll reimplement Chapter 11's `MyArray` **case study** as a class template with custom iterators that enable most C++ standard library algorithms to manipulate `MyArray` objects. We also define a custom algorithm that can process `MyArray` elements and standard library container class objects. We show that you can use concepts to overload functions based on the type requirements of their parameters. Finally, we introduce template metaprogramming for performing compile-time calculations, enabling you to improve a program's execution-time performance, possibly reducing both execution time and memory consumption.

Chapter 16, C++20 Modules, presents another of C++20's "big four" features. Modules provide a new way to organize your code, precisely control which declarations you expose to client code and encapsulate implementation details. Modules help you be more productive, especially when building, maintaining and evolving large software systems. Modules help such systems build faster and make them more scalable. C++ creator Bjarne Stroustrup says, "*Modules offer a historic opportunity to improve code hygiene and compile times for C++ (bringing C++ into the 21st century).*"[13] You'll see that, even in small systems, modules offer immediate benefits in every program by eliminating the need for the C++ preprocessor. In several **Software Engineering case studies**, you'll learn several ways to **separate interface from implementation using modules**.

Chapter 17, Parallel Algorithms and Concurrency: A High-Level View, is the first of two extensive chapters on concurrency and multi-core programming. Chapter 17 is one of the most important chapters in the book, presenting C++'s features for building applications that create and manage multiple tasks. These can significantly improve program performance and responsiveness on today's multi-core processors.

This chapter presents several **multithreading and multicore systems performance case studies**. In the **Profiling Sequential and Parallel Sorting Algorithms case study**, we show how to use prepackaged parallel algorithms to create multithreaded programs that will run faster (often much faster) on today's multi-core computer architectures. For example, we sort 100 million values using a sequential sort, then a parallel sort. **We use timing operations from C++'s `<chrono>` library features to profile the performance improvement we get on today's popular multi-core systems, as we employ more cores. You'll see that the parallel sort runs 6.76 times faster than the sequential sort on our computer with an 8-core Intel processor**.

In the **Producer–Consumer: Synchronizing Access to Shared Mutable Data case studies**, we discuss the producer–consumer relationship and demonstrate various ways to implement it using low-level and high-level C++ concurrency primitives. We also present several **Coordinating Threads case studies using C++20's new latch, barrier and semaphore capabilities**. We emphasize that concurrent programming is difficult to get right, so you should prefer the easier-to-use, higher-level concurrency features. Lower-level features like semaphores and atomics can be used to implement higher-level features like latches.

Chapter 18, C++20 Coroutines, is the second of our chapters on concurrency and multi-core programming. This chapter presents coroutines—the last of C++20's "big four" features. **A coroutine is a function that can suspend its execution and be resumed later,**

13. Bjarne Stroustrup, "Modules and Macros." February 11, 2018. Accessed March 8, 2023. `http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2018/p0955r0.pdf`.

enabling you to do concurrent programming with a simple sequential-like coding style. The mechanisms supporting this are handled entirely by code that's written for you by the compiler. A function containing any of the keywords `co_await`, `co_yield` or `co_return` is a coroutine.

Coroutines require sophisticated infrastructure, which you can write yourself, but doing so is complex, tedious and error-prone. Instead, most experts agree that you should SE use high-level coroutine support libraries, which is the approach we show. The open-source community has created several experimental libraries for developing coroutines quickly and conveniently—we use two in our presentation. C++23 has standard library support for generator coroutines, and more coroutine support is expected in C++26.

Perf We present three **multithreading and multicore systems performance case studies**:

- **Creating a Generator Coroutine with `co_yield` and the `generator` Library**
- **Launching Tasks with `concurrencpp`**
- **Creating a Coroutine with `co_await` and `co_return`**

To help readers understand how coroutines work, the first and third case studies include diagrams illustrating each application's flow of control.

Chapter 19, Stream I/O & C++20 Text Formatting, discusses C++ stream input/output capabilities and formatting features. We include stream formatting primarily for people who might encounter it in legacy code. Section 19.9 presents an in-depth **case study** on **C++20's new text-formatting features**. In introductory computer science courses, instructors may wish to present Section 19.9 after we introduce C++20 text formatting in Chapter 4.

Chapter 20, Other Topics and a Look Toward the Future of C++, continues our discussion of runtime polymorphism from Chapter 10 with **case studies** on runtime type information (RTTI), inheriting base-class constructors, the non-virtual interface idiom, duck typing with `std::variant` and `std::visit` (for runtime polymorphism with objects of classes that are not related by inheritance), and multiple inheritance. The chapter also presents miscellaneous C++ topics, including storage classes, storage duration, `mutable` class members, namespaces, operator keywords, pointers to class members, the `[[nodiscard]]` attribute, the `std::shared_ptr` and `std::weak_ptr` smart pointers, determining types at compile-time with `decltype`, and the `[[likely]]` and `[[unlikely]]` attributes. The chapter ends with a look forward to features coming in the C++23 and C++26 standards.

Chapter 21, Computer Science Thinking: Searching, Sorting 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.

Perf The chapter includes two **case studies** that visualize the **high-speed binary search and merge sort algorithms** to illustrate how these algorithms work. In introductory computer science courses, instructors can present this chapter after Chapter 6.

Modern C++ Data Structures Courses

Our recursion (Chapter 5), `arrays` (Chapter 6), searching (Chapters 6 and 21), sorting (Chapters 6 and 21), Big O (Chapter 21), containers (Chapter 13), iterators (Chapter 13)

and algorithms (Chapter 14) coverage provides nice content for a course emphasizing Modern C++ data structures.

7 Compilers, Docker and Static Code Analysis Tools

Industrial-Strength Compilers

We tested all the code for correctness on the Windows, macOS and Linux operating systems using the latest versions of

- Visual C++® in Microsoft® Visual Studio® Community edition on Windows®,
- GNU® C++ (`g++`) and
- Clang C++ (`clang++`).

See the Before You Begin section that follows this Preface for software installation instructions.

Most C++20 features are now fully implemented in these compilers. We point out exceptions as appropriate. As coverage improves, we'll post code updates to the book's GitHub repository:

```
https://github.com/pdeitel/CPlusPlusHowToProgram11e
```

and both code and text updates on the book's website:

```
https://www.deitel.com/books/cpphtp11
```

At the time of this writing, Apple's Xcode integrated development environment (IDE) did not support various key C++20 features we use. Once these features become available in Xcode, we'll post Xcode instructions on the preceding website.

Docker

Docker is a tool for packaging software into containers that bundle everything required to execute that software conveniently and portably across platforms. Docker provides a simple way to help you get started with new technologies quickly, conveniently and economically on your desktop or notebook computers. We show how to install and execute Docker containers preconfigured with

- the GNU Compiler Collection (GCC), which includes the `g++` compiler, and
- the latest version of Clang's `clang++` compiler.

Each can run in Docker on Windows, macOS and Linux, enabling users to try the latest versions of these compilers. Chapter 1 includes test-drives showing how to compile programs and run them in the context of cross-platform Docker containers.

Static Code Analysis Tools

Static code analysis tools let you quickly check your code for common errors and security problems and provide insights for code improvement. Using these tools is like having world-class experts checking your code. To help us adhere to the C++ Core Guidelines and improve our code in general, we used the following static-code analyzers:

- `clang-tidy`—`https://clang.llvm.org/extra/clang-tidy/`
- `cppcheck`—`https://cppcheck.sourceforge.io/`

- Microsoft's C++ Core Guidelines static code analysis tools, which are built into Visual Studio's static code analyzer

We used these three tools on all the book's code examples to check for

- adherence to the C++ Core Guidelines,
- adherence to coding standards,
- adherence to modern C++ idioms,
- possible security problems,
- common bugs,
- possible performance issues,
- code readability
- and more.

We also used the compiler flag `-Wall` in the GNU `g++` and LLVM `clang++` compilers to enable all compiler warnings. Most of our programs compile without warning messages. See the Before You Begin section that follows this Preface for information on configuring the C++ Core Guidelines checker in Microsoft Visual C++.

8 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.[14]—William Gates

You'll work with such popular websites as GitHub and StackOverflow, and you'll do lots of Internet research.

- StackOverflow is one of the most popular programming question-and-answer sites. Many problems you might encounter have already been discussed here. It's a great place to ask code-oriented questions. Many of our Google searches for various, often complex, issues throughout our writing effort returned StackOverflow posts as their first results.
- GitHub is an excellent venue for finding free, open-source code to explore and incorporate into your projects—and for you to contribute your code to the open-source community if you like. One hundred million developers use GitHub.[15] The site hosts over 330 million repositories for code in many programming languages[16]—developers made 413 million contributions to repositories in 2022.[17]

14. William Gates, quoted in *Programmers at Work: Interviews With 19 Programmers Who Shaped the Computer Industry* by Susan Lammers. Microsoft Press, 1986, p. 83.
15. "Let's build from here: The complete developer platform to build, scale, and deliver secure software." Accessed March 8, 2023. `https://github.com/about`.
16. "Let's build from here: The complete developer platform to build, scale, and deliver secure software." Accessed March 8, 2023. `https://github.com/about`.
17. "OCTOVERSE 2022: The state of open source software." Accessed March 8, 2023. `https://octoverse.github.com`.

GitHub is a crucial element of the professional software developer's arsenal, with version-control tools that help developer teams manage public open-source projects and private projects. There is a massive C++ open-source community on GitHub where developers contribute to almost 58,000[18] C++ code repositories. We encourage you to study and execute lots of developers' open-source C++ code. This is a great way to learn and is a natural extension of our live-code teaching approach.[19]

9 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 curricula, which call for covering security, data science, ethics, privacy and performance concepts and using real-world data:

- Computer Science Curricula 2013,[20]
- CC2020: A Vision on Computing Curricula,[21]
- Computing Curricula 2020 recommendations,[22]
- Information Technology Curricula 2017,[23]
- Cybersecurity Curricula 2017,[24]
- the 2016 data science initiative "Curriculum Guidelines for Undergraduate Programs in Data Science"[25] from the faculty group sponsored by the NSF and the Institute for Advanced Study, and
- ACM Data Science Task Force's Computing Competencies for Undergraduate Data Science Curricula Final Report.[26]

18. "C++." Accessed March 8, 2023. https://github.com/topics/cpp.
19. You'll need to become familiar with the variety of open-source licenses for software on GitHub.
20. 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). Accessed March 8, 2023. http://ai.stanford.edu/users/sahami/CS2013/final-draft/CS2013-final-report.pdf.
21. A. Clear, A. Parrish, G. van der Veer and M. Zhang "CC2020: A Vision on Computing Curricula." Accessed March 8, 2023. https://dl.acm.org/citation.cfm?id=3017690.
22. CC2020 Task Force, *Computing Curricula 2020*. Accessed March 8, 2023. https://www.acm.org/binaries/content/assets/education/curricula-recommendations/cc2020.pdf.
23. *Information Technology Curricula 2017*. Accessed March 8, 2023. http://www.acm.org/binaries/content/assets/education/it2017.pdf.
24. *Cybersecurity Curricula 2017*. Accessed March 8, 2023. https://cybered.hosting.acm.org/wp-content/uploads/2018/02/newcover_csec2017.pdf.
25. "Curriculum Guidelines for Undergraduate Programs in Data Science" Accessed March 8, 2023. http://www.annualreviews.org/doi/full/10.1146/annurev-statistics-060116-053930.
26. ACM Data Science Task Force, *Computing Competencies for Undergraduate Data Science Curricula*. Accessed March 8, 2023. https://dstf.acm.org/DSTF_Final_Report.pdf.

Computing Curricula

- According to "CC2020: A Vision on Computing Curricula,"[27] the curriculum "needs to be reviewed and updated to include the new and emerging areas of computing such as cybersecurity and data science."[28]

DS 10 Data Science Overlaps with Computer Science[29]

The undergraduate data science curriculum proposal[30] 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. We work some basic 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.[31] We cover each of the following:

- 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, functional and object-oriented programming.
- Explore concepts via simulations.
- Use development environments (we tested all our code on Microsft Visual C++, GNU `g++` and LLVM `clang++`).
- Work with real-world data in practical case studies and projects.
- Create data visualizations.
- Work with existing software.
- Work with high-performance tools, such as C++'s multithreading libraries.
- Focus on data's ethics, security and privacy issues.

11 Appendices on Deitel.com

On the textbook's webpage at `https://deitel.com/cpphtp11`, we provide several appendices to support the book:

---

27. 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`.
28. `http://delivery.acm.org/10.1145/3020000/3017690/p647-clear.pdf`.
29. This section is intended primarily for data science instructors but includes important information for computer science instructors as well.
30. "Curriculum Guidelines for Undergraduate Programs in Data Science," `http://www.annualreviews.org/doi/full/10.1146/annurev-statistics-060116-053930`.
31. "Appendix—Detailed Courses for a Proposed Data Science Major," `http://www.annualreviews.org/doi/suppl/10.1146/annurev-statistics-060116-053930/suppl_file/st04_de_veaux_supmat.pdf`.

- Appendix A, Character Set, contains the letters, digits and symbols of the ASCII character set.
- Appendix B, Number Systems, overviews the binary, octal, decimal and hexadecimal number systems.
- Appendix C, Preprocessor, discusses additional features of the C++ preprocessor. Template metaprogramming (Chapter 15) and C++20 Modules (Chapter 16) eliminate the need for many of this appendix's features.
- Appendix D, Bit Manipulation, discusses bitwise operators for manipulating the individual bits of integral operands and bit fields for compactly representing integer data.

Other Web-Based Materials on `deitel.com`

The book's webpage also contains:

- Links to our GitHub repository containing the downloadable C++ source code
- Blog posts—`https://deitel.com/blog`
- Book updates

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

12 C++ Core Guidelines

The C++ Core Guidelines

`https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines`

are recommendations "to help people use modern C++ effectively."[32] They're edited by Bjarne Stroustrup (C++'s creator) and Herb Sutter (Convener of the ISO C++ Standards Committee). According to the overview:

> "*The guidelines are focused on relatively high-level issues, such as interfaces, resource management, memory management, and concurrency. Such rules affect application architecture and library design. Following the rules will lead to code that is statically type safe, has no resource leaks, and catches many more programming logic errors than is common in code today. And it will run fast—you can afford to do things right.*"[33]

Throughout this book, we adhere to these guidelines as appropriate. You'll want to pay close attention to their wisdom. We point out many C++ Core Guidelines recommendations with a **CG** icon. There are hundreds of core guidelines divided into scores of categories and subcategories. Though this might seem overwhelming, the static code analysis tools we discussed earlier can check your code against the guidelines.

32. C++ Core Guidelines, "Abstract." Accessed March 8, 2023. `https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines#S-abstract`.
33. C++ Core Guidelines, "Abstract."

Guidelines Support Library

The C++ Core Guidelines often refer to capabilities of the Guidelines Support Library (GSL), which provides reusable C++ components that support various recommendations.[34] Microsoft provides an open-source GSL implementation on GitHub at

```
https://github.com/Microsoft/GSL
```

For your convenience, we include with the book's code examples the version of this library that we used in a few examples. Some GSL features have been incorporated into the C++ standard library.

13 Pedagogic Features and Conventions

SE 

C++ How to Program: An Objects-Natural Approach, 11/e contains hundreds of live-code examples. We stress program clarity and concentrate on building well-engineered software.

Using Fonts for Emphasis

Our 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. Our e-book syntax-coloring conventions are:

```
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.

Tables and Illustrations

Abundant tables and line drawings are included.

Programming Tips and Key Features

We call out programming tips and key features with icons in the margins (see Section 5).

Index

For convenient reference, we've included an extensive index, with defining occurrences of key terms highlighted with a **bold** page number.

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 document in preference to general programming terms.

---

34. C++ Core Guidelines, "GSL: Guidelines Support Library." Accessed March 8, 2023. `https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines#S-gsl`.

- We avoid heavy math, leaving it to upper-level courses. Optional mathematical exercises and projects are included for science and engineering courses.

Innovation: "Intro-to" Pedagogy with 452 Integrated Checkpoint Exercises

This book uses our new "Intro to" pedagogy with integrated Checkpoint exercises and answers. We introduced this pedagogy in our textbook, *Intro to Python for Computer Science and Data Science: Learning to Program with AI, Big Data and the Cloud* (`https://deitel.com/pycds`). Chapter sections are intentionally small. We use a "read-a-little, code-a-little, test-a-little" approach. In the core computer science chapters (1–12 and 21), you read about a new concept, study and execute the corresponding code examples, then test your understanding via the integrated fill-in-the-blank, true/false, discussion and code-based Checkpoint exercises immediately followed by their answers. This will help you keep a brisk learning pace.

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—To "take the pulse" of Modern C++, which changes the way developers write C++ programs, we read, browsed or watched approximately 6,000 current articles, research papers, white papers, books, documentation pieces, blog posts, forum posts, webinars and videos.
- 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.

Over 700 Contemporary Examples, Exercises and Projects (EEPs)

Consistent with our live-code and object-natural approaches, you'll learn hands-on from a broad selection of 255 real-world examples and case studies, and 476 exercises and projects drawn from computer science, data science and other fields:

- Our code examples, exercises and projects familiarize students with current topics of interest to developers. We begin in Chapter 1 by briefly touring topics of current interest including open-source software, virtualization, simulation, web services, multithreading, multicore hardware architecture, systems programming, artificial intelligence, natural language processing, data science, robust secure programming, cryptography, Docker, GitHub, StackOverflow, forums, the metaverse, blockchain, NFTs (nonfungible tokens), cryptocurrencies (like Bitcoin and Ethereum), generative AI (ChatGPT, Dall-E), general artificial intelligence and more.

- We added a variety of systems programming and application programming case studies. Some are book sections that walk through the complete source code, some are exercises with detailed specifications from which you should be able to develop the code solution on your own, and some are exercises requiring additional research. We enumerate the 50 case studies and case-study exercises in this preface.

- You'll attack exciting and entertaining challenges in our larger case studies, such as building a casino game, building your own computer (using simulation to build a virtual machine), using AI/data-science technologies such as basic natural language processing, building your own compiler, writing multithreaded code to take advantage of today's multicore computer architectures to get the best performance from your computer and many more.
- Research and project exercises ask 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 elaborate than the exercises—some might require days or weeks of implementation effort. Many are appropriate for class projects, term projects, directed-study courses, capstone-course projects and thesis research. We do not provide solutions for the projects.
- We've enhanced existing case studies and added new ones focusing on AI and data science, including simulations with random-number generation, Anscombe's Quartet, natural language processing (NLP) and artificial intelligence via heuristic programming.
- Instructors can tailor their courses to their audience's unique requirements and vary labs and exam questions each semester.

Extensive Videos

In the Pearson interactive eText and Revel versions of this book, we provide extensive videos in which Paul Deitel discusses the material in the Before You Begin section and Chapters 1–10.

Glossary Items

In the Pearson interactive eText and Revel versions of the book, we added over 300 glossary items for the core computer science chapters (1–12 and 21). These are also used in student learning tools to create flashcards and matching exercises.

Performance

Software developers 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.

Security Emphasis and Cryptography Case Studies

Consistent with our richer treatment of security, we've added case studies on secret-key and public-key cryptography. The latter is a project exercise that includes a detailed walkthrough of the enormously popular RSA algorithm's steps, providing hints to help you build a working, simple, small-scale classroom implementation.

Working with Open-Source Software

Open source is software with source code that anyone can inspect, modify, and enhance."[35] We encourage you to try lots of demos and view free, open-source code examples (available on sites such as GitHub) for inspiration.

35. "What is open source?" Accessed March 8, 2023. `https://opensource.com/resources/what-open-source`.

Data Experiences

In Chapter 9, you'll work with real-world text data. You'll read and analyze the Titanic Disaster dataset—popular for introducing data analytics. Datasets are often stored in CSV (comma-separated values) files, which we introduce in Chapter 9. You'll also download and analyze Shakespeare's play *Romeo and Juliet* and Christopher Marlowe's play *Edward the Second* from Project Gutenberg—a source of free downloadable texts for analysis. The site contains over 60,000 e-books in various formats, including plain-text files—these are out of copyright in the United States.

Privacy

The ACM/IEEE's curricula recommendations[36] 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 and 3. We also discuss cryptography, which is critical in maintaining privacy, in Chapters 5 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), 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[37] 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 Amazon Alexa, Apple Siri, Google Assistant and Microsoft Cortana. And we'll look at the excitement and controversy surrounding OpenAI's ChatGPT[38] and Dall-E 2.[39]

14 Instructor Supplements

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

- The Instructor Solutions Manual contains solutions to most of the end-of-chapter exercises. Solutions are not provided for "project" exercises.
- A Test Item File containing multiple-choice questions and answers.
- Lecture slides containing diagrams, tables and bulleted items summarizing key points in the text.

---

36. "Curricula Recommendations." Accessed March 8, 2023. `https://www.acm.org/education/curricula-recommendations`.
37. "Curricula Recommendations." Accessed March 8, 2023. `https://www.acm.org/education/curricula-recommendations`.
38. "Introducing ChatGPT." Accessed March 8, 2023. `https://openai.com/blog/chatgpt`.
39. "Dall-E 2." Accessed March 8, 2023. `https://openai.com/product/dall-e-2`.

The lecture slides do not include the source code—the source-code files[40] for the hundreds of live-code examples are available to instructors and students in the book's GitHub repository at

```
https://github.com/pdeitel/CPlusPlusHowToProgram11e
```

If you're not a GitHub user, click the green **Code** button and select **Download ZIP** to download a ZIP file containing the code. See the Before You Begin section for more information.

Please do not write to us requesting access to the Pearson Instructor's Resource Center. Access is restricted to college instructors who have adopted the book for their courses. Instructors may obtain access only through their Pearson representatives. If you're not a registered faculty member, contact your Pearson representative or visit

```
https://pearson.com/replocator
```

15 Some Key C++ Documentation and Resources

The book includes almost 800 citations to videos, blog posts, articles, whitepapers and online documentation we studied while writing the manuscript. You may want to access some of these resources to investigate more advanced features and idioms. The website `cppreference.com` has become the defacto C++ documentation site. We reference it frequently so you can get more details about the standard C++ classes and functions we use throughout the book. We also frequently cite the final draft of the C++20 standard document, which is available free on GitHub at

```
https://timsong-cpp.github.io/cppwp/n4861/
```

The C++ standard committee's evolving working draft (currently C++23) is available at:

```
https://eel.is/c++draft/
```

You may also find the following C++ resources helpful as you work through the book.

Documentation

- C++ Reference at `https://cppreference.com/`
- Microsoft's C++ language documentation: `https://docs.microsoft.com/en-us/cpp/cpp/`
- The GNU C++ Standard Library Reference Manual: `https://gcc.gnu.org/onlinedocs/libstdc++/manual/index.html`

16 Getting Your Questions Answered

Popular C++ and general programming online forums include

- `https://stackoverflow.com`
- `https://www.reddit.com/r/cpp/`
- `https://groups.google.com/g/comp.lang.c++`

---

40. We recommend that instructors present source code in their favorite C++ integrated development environments (IDEs) or code-oriented text editors. These typically provide customizable syntax highlighting and adjustable font sizes for presentation purposes.

For a list of other valuable 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.

Communicating with the Authors

As you read the book, if you have questions, we're easy to reach at

```
deitel@deitel.com
```

We'll respond promptly.

17 Join the Deitel & Associates, Inc. Social Media Communities

You can keep up-to-date with Deitel on the following social media platforms:

- LinkedIn®—`https://www.linkedin.com/company/deitel-&-associates/`
- YouTube®—`https://youtube.com/DeitelTV`
- Twitter®—`https://twitter.com/deitel`
- Facebook®—`https://facebook.com/DeitelFan`
- Instagram®—`https://instagram.com/DeitelFan`

18 Live Instructor-Led Training with Paul Deitel

Paul Deitel has been teaching programming languages to academic and professional audiences for three decades. He presents a variety of one- to five-day C++, C, Python and Java courses, and teaches Python with an Introduction to Data Science for the UCLA Anderson School of Management's Master of Science in Business Analytics (MSBA) program. The longer classes include intense, hands-on labs. His courses are delivered worldwide onsite or virtually. Please contact `deitel@deitel.com` for a proposal customized to meet your programming-language training needs.

19 College Textbook Digital Formats

Our college textbook, ***C++ How to Program: An Objects-Natural Approach, 11/e***, is available in three digital formats:

- Online e-books offered through popular e-book providers.
- Interactive Pearson eText (see below).
- Interactive Pearson Revel with assessment (see below).

All of these textbook versions include standard "How to Program" features such as:

- A chapter introducing hardware, software and Internet concepts.
- An introduction to programming for novices.

- End-of-section programming and non-programming Checkpoint self-review exercises with answers.
- End-of-chapter exercises.

Deitel Pearson eTexts and Revels include:

- Videos in which Paul Deitel discusses the material in the book's core CS1 chapters (1–10).
- Interactive programming and non-programming Checkpoint self-review exercises with answers.
- Glossaries, flashcards, matching exercises and other learning tools.

In addition, Pearson Revels include interactive programming and non-programming automatically graded exercises, as well as instructor course-management tools, such as a grade book.

Supplements available to qualified college instructors teaching from the textbook include:

- **Instructor solutions manual** with solutions to most of the end-of-chapter exercises.
- **Test-item file** with four-part, code-based and non-code-based multiple-choice questions with answers.
- Customizable **PowerPoint lecture slides**.

Please write to **`deitel@deitel.com`** for more information.

20 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, energy and editorial savvy of Tracy Johnson (Pearson Education, Global Content Manager, Computer Science)—on all our academic publications. She challenges us at every step of the process to "get it right" and make the best books. Bob Engelhardt managed the book's production. Erin Sullivan recruited and managed the academic reviewers; Charvi Arora recruited and managed the professional reviewers. We selected the cover art, and Chuti Prasertsith designed the cover, adding his special touch of graphics magic.

Reviewers

We were fortunate on this project to have 14 distinguished academics and professionals review the manuscript. Most of the professional reviewers are either on the ISO C++ Standards Committee, have served on it or have a working relationship with it. Many have contributed features to the language. They helped us make a better book—any remaining flaws are our own.

Academic Review Team—Prof. Jeffrey Davis, School of Electrical and Computer Engineering, Georgia Institute of Technology; M. Michael Hadavi, The MathWorks, Inc., MET Computer Science at Boston University; Dr. Ningfang Mi, Associate Professor

of Electrical and Computer Engineering, Northeastern University; Prof. Patrice Roy, Université de Sherbrooke, ISO C++ Standards Committee Member.

Professional Review Team—Andreas Fertig, Independent C++ Trainer and Consultant, Creator of `cppinsights.io`, Author of *Programming with C++20*; Marc Gregoire, Software Architect, Nikon Metrology, Microsoft Visual C++ MVP and author of *Professional C++, 5/e*; Dr. Daisy Hollman, ISO C++ Standards Committee Member; Danny Kalev, Ph.D. and Certified System Analyst and Software Engineer, Former ISO C++ Standards Committee Member; Dietmar Kühl, Senior Software Developer, Bloomberg L.P., ISO C++ Standard Committee Member; Inbal Levi, SolarEdge Technologies, ISO C++ Foundation director, ISO C++ SG9 (Ranges) chair, ISO C++ Standards Committee member; Arthur O'Dwyer, C++ trainer, Chair of CppCon's Back to Basics track, author of several accepted C++17/20/23 proposals and the book *Mastering the C++17 STL*; Saar Raz, Senior Software Engineer, Swimm.io and Implementor of C++20 Concepts in Clang; José Antonio González Seco, Parliament of Andalusia; Anthony Williams, Member of the British Standards Institution C++ Standards Panel, Director of Just Software Solutions Ltd., Author of C++ *Concurrency in Action, 2/e* (Anthony is the author or co-author of many C++ Standard Committee papers that led to C++'s standardized concurrency features).

Google Search

Thanks to Google, whose search engine answers our constant stream of queries, each in a fraction of a second, at any time—and at no charge. It's the single best productivity enhancement tool we've added to our research process in the last 20 years.

Grammarly

We use the paid version of **Grammarly** on all our manuscripts. They describe their tools as helping you "compose bold, clear, mistake-free writing" with their "AI-powered writing assistant."[41] Grammarly also provides free tools that you can integrate into several popular web browsers, Microsoft® Office 365™ and Google Docs™.

As you read the book and work through the code examples, 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. We've enjoyed writing 11 editions of our academic and professional C++ content over the last 30 years. We hope you have an informative, challenging and entertaining learning experience with ***C++ How to Program: An Objects-Natural Approach, 11/e*** and enjoy this look at Modern C++ software development.

Paul Deitel
Harvey Deitel

41. "Grammarly." Accessed March 8, 2023. `https://www.grammarly.com`.

21 About the Authors

Paul J. Deitel, CEO and Chief Technical Officer of Deitel & Associates, Inc., is an MIT graduate with 43 years in computing. He 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 of Deitel & Associates, Inc. internationally, including UCLA, SLB (formerly Schlumberger), 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, Puma, iRobot and many more.

Dr. Harvey M. Deitel, Chairman and Chief Strategy Officer of Deitel & Associates, Inc., has 62 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 departments. He has extensive college and professional teaching experience, including earning tenure and serving as the Chairman of the Computer Science Department at Boston College before founding Deitel & Associates in 1991 with his son, Paul. 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.

22 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 virtually worldwide for Pearson Education on O'Reilly Online Learning (`https://learning.oreilly.com`), formerly called Safari Books Online.

Through its 48-year publishing partnership with Pearson, Deitel & Associates, Inc. publishes leading-edge computer programming college textbooks and professional books in print and digital formats, LiveLessons video courses, O'Reilly Online Learning live training courses and Revel™ and eText 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 virtual and on-site corporate training, visit

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

Individuals wishing to purchase Deitel books can do so at

```
https://amazon.com
https://www.barnesandnoble.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
```

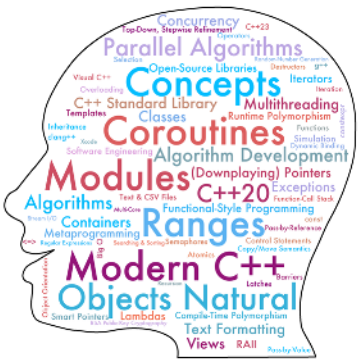
Deitel e-books are available in various formats from

```
https://www.amazon.com/
https://www.barnesandnoble.com/
https://www.informit.com/
https://www.vitalsource.com/
https://www.redshelf.com/
https://www.chegg.com/
```

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

```
https://learning.oreilly.com/register/
```

This page intentionally left blank



Before You Begin

Before using this book, please read this section to understand our conventions and set up your computer to compile and run our example programs. If there are changes to the instructions presented here, we'll post updates on the book's webpage:

```
https://deitel.com/cpphtp11
```

Font and Naming Conventions

We use fonts to distinguish application elements and C++ code elements from regular text:

- We use a **bold sans-serif font** for on-screen application elements, such as "the **File** menu."
- We use a `sans-serif font` for commands and C++ code elements, as in `sqrt(9)`.

Obtaining the Code Examples

Download the *C++ How to Program: An Objects-Natural Approach, 11/e* code examples from our GitHub repository at

```
https://github.com/pdeitel/CPlusPlusHowToProgram11e
```

If you're familiar with Git and GitHub, clone the repository to your system. If you're not a GitHub user, click the green **Code** button and select **Download ZIP** to download a ZIP file containing the code. To extract the ZIP file's contents:

- Windows: Right-click the ZIP file, select **Extract All...**, select your user account's `Documents` folder, then click **Extract**.
- macOS: Move the ZIP file to your user account's `Documents` folder, then double-click the ZIP file.
- Linux (varies by distribution): When you download the ZIP file on Ubuntu Linux, you can choose to open the file with the **Archive Manager** or save it. Choose **Archive Manager**, then click **Extract** in the window that appears. Select your user account's `Documents` folder, then click **Extract** again.

Throughout the book, our instructions assume the code 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

```
https://guides.github.com/activities/hello-world/
```

Compilers We Use

Ensure that you have a recent C++ compiler installed. We tested the book's code examples using the following free compilers:

- For Microsoft Windows, we used Microsoft Visual Studio Community edition, which includes the Visual C++ compiler and other Microsoft development tools.
- For Linux, we used the GNU C++ compiler (`g++`)[1]—part of the GNU Compiler Collection (GCC). Typically, a version of GNU C++ is pre-installed on most Linux systems. You might need to update the compiler to a more recent version. GNU C++ also can be installed on macOS and Windows systems.
- For macOS, we used both the GNU C++ compiler (`g++`) and the Apple Xcode[2] C++ compiler, which uses a version of the LLVM Clang C++ compiler (`clang++`).
- You can run the latest versions of GNU C++ (`g++`) and LLVM Clang C++ (`clang++`)[3] conveniently on Windows, macOS and Linux via Docker containers. See the "Docker and Docker Containers" section in this Before You Begin section.

At the time of this writing, Apple Xcode does not support several key C++20 features we use throughout this book, so we recommend using the most recent version of `g++`. When Xcode's C++20 support changes, we'll post updates at

```
https://deitel.com/cpphtp11
```

This Before You Begin describes installing the compilers and Docker. Section 1.11's test-drives demonstrate how to compile and run C++ programs using these compilers.

Installing Visual Studio Community Edition on Windows

If you are a Windows user, first ensure that your system meets the requirements for Microsoft Visual Studio Community edition at

```
https://docs.microsoft.com/en-us/visualstudio/releases/2022/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 where you can click **Run** to start the installation process. If not, double-click the installer file in your **Downloads** folder when the download completes.
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.

---

1. GNU C++ (`g++`) 13.1 at the time of this writing.
2. Xcode 14.3.1 at the time of this writing.
3. Clang C++ (`clang++`) 16 at the time of this writing.

5. For this book's examples, select the option **Desktop Development with C++**, which includes the Visual C++ compiler and the C++ standard libraries.
6. Click **Install**. 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. In the **App Store**'s **Search** field, type **Xcode**.
3. Click the **Get** button to install Xcode.

Installing GNU C++ (g++) 13 on macOS

On macOS, perform the following steps to install GNU C++:

1. In the **Finder**'s **Go** menu, select **Utilities**, then double-click **Terminal** to open a **Terminal** (command line) window.
2. Check if the `brew` command is installed by typing `brew` and pressing *Enter* (or *return*). If macOS does not recognize the command, go to `https://brew.sh` and copy the installation command below **Install Homebrew**. Paste this command into the **Terminal** window, then press *Enter* (or *return*).
3. Type the following command, then press *Enter* (or *return*) to install the GNU Compiler Collection (GCC), which includes `g++`:

```
brew install gcc@13
```

Installing the GNU C++ (g++) 13 on Linux

There are many Linux distributions, and they often use different software upgrade techniques. Check your distribution's online documentation for instructions on how to upgrade GNU C++ to the latest version. You also can download GNU C++ for various platforms at

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

Docker and Docker Containers

Docker is a tool for packaging software into **containers** (also called **images**) that bundle everything required to execute that software across platforms, which is particularly useful for software packages with complicated setups and configurations. For many such packages, there are free preexisting Docker containers (often at `https://hub.docker.com`) that you can download and execute locally on your system. Docker is a great way to get started with new technologies quickly and to experiment with new compiler versions.

Installing Docker

To use a Docker container, you must first install Docker. Windows and macOS users should download and run the **Docker Desktop** installer from

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

Then follow the on-screen instructions. Also, sign up for a **Docker Hub** account on this site, which gives you access to the many containers at `https://hub.docker.com`. Linux users should install **Docker Engine** from

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

Getting the GNU Compiler Collection Docker Container

The GNU team maintains official Docker containers at

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

Once Docker is installed and running, open a Command Prompt[4] (Windows), Terminal (macOS/Linux) or shell (Linux), then execute the command

```
docker pull gcc:latest
```

Docker downloads a container configured with the GNU Compiler Collection (GCC)'s most current version—13.1 at the time of this writing. In one of Section 1.11's test-drives, we'll demonstrate how to execute the container and use it to compile and run C++ programs.

Getting an LLVM Clang C++ Docker Container

Currently, the LLVM Clang team does not provide an official Docker container, but many working containers are available on `https://hub.docker.com`. For this book, we used a popular one from

```
https://hub.docker.com/r/teeks99/clang-ubuntu
```

Open a Command Prompt (Windows), Terminal (macOS/Linux) or shell (Linux), then execute the command

```
docker pull teeks99/clang-ubuntu:16
```

Docker downloads a container configured with LLVM Clang's most current version—16 at the time of this writing. In one of Section 1.11's test-drives, we'll demonstrate how to execute the container and use it to compile and run C++ programs.

Getting Your C++ Questions Answered

As you read the book, if you have questions, we're easy to reach at

```
deitel@deitel.com
```

and

```
https://deitel.com/contact-us
```

We'll respond promptly.

The web is loaded with programming information. An invaluable resource for nonprogrammers and programmers alike is the website

```
https://stackoverflow.com
```

on which you can

- search for answers to common programming questions,
- search for error messages to see what causes them,

4. Windows users should choose **Run as administrator** when opening the Command Prompt.

- ask programming questions to get answers from programmers worldwide and
- gain valuable insights about programming in general.

For live C++ discussions, check out the Slack channel cpplang:

```
https://cpplang-inviter.cppalliance.org
```

and the Discord server `#include<C++>`:

```
https://www.includecpp.org/discord/
```

Online C++ Documentation

For C++ standard library documentation, visit

```
https://cppreference.com
```

Also, be sure to check out the C++ FAQ at

```
https://isocpp.org/faq
```

Static Code Analysis Tools

We used the following static code analyzers to check our code examples for adherence to the C++ Core Guidelines, adherence to coding standards, adherence to Modern C++ idioms, possible security problems, common bugs, possible performance issues, code readability and more:

- **`clang-tidy`**—`https://clang.llvm.org/extra/clang-tidy/`
- **`cppcheck`**—`https://cppcheck.sourceforge.io/`
- **Microsoft's C++ Core Guidelines static code analysis tools**, which are built into Visual Studio's static code analyzer

You can install `clang-tidy` on Linux with the following commands:

```
sudo apt-get update -y
sudo apt-get install -y clang-tidy
```

You can install `cppcheck` for various operating-system platforms by following the instructions at `https://cppcheck.sourceforge.io/`.

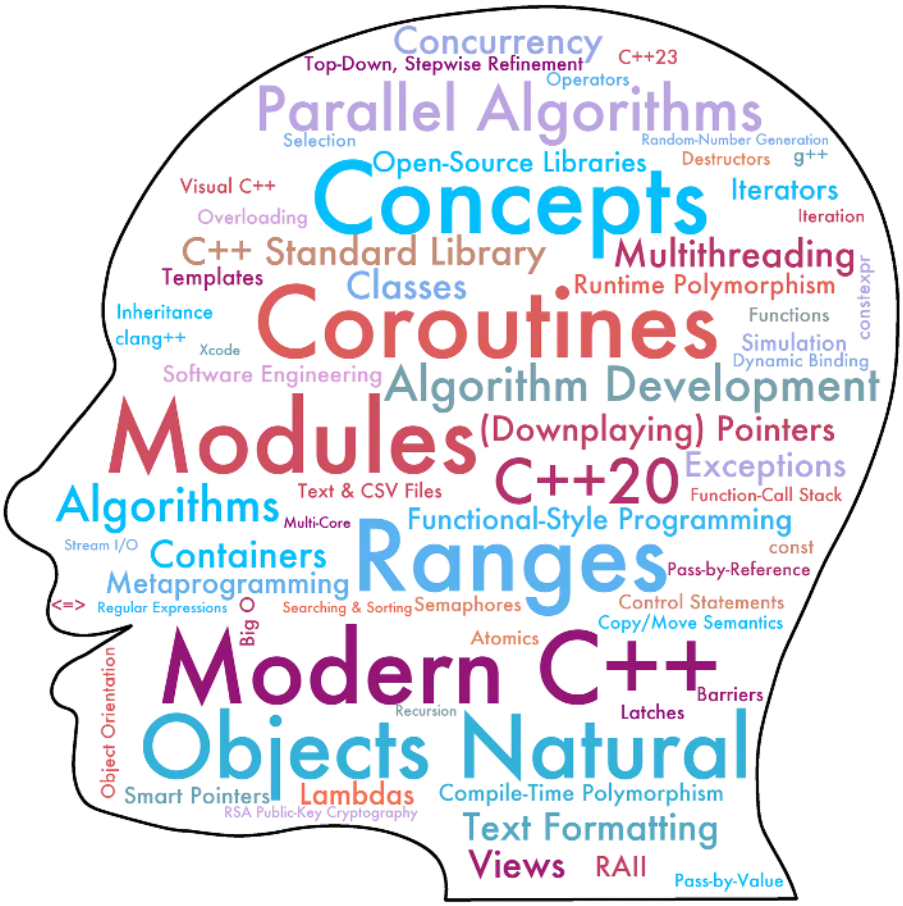
For Visual C++, once you learn how to create a project in Section 1.11's test-drives, you can configure Microsoft's C++ Core Guidelines static code analysis tools as follows:

1. Right-click your project name in the **Solution Explorer** and select **Properties**.
2. In the dialog that appears, select **Code Analysis > General** in the left column, then set **Enable Code Analysis on Build** to **Yes** in the right column.
3. Next, select **Code Analysis > Microsoft** in the left column. Then, in the right column, you can select a subset of the analysis rules from the drop-down list. We used the option **<Choose multiple rule sets...>** to select all the rules that begin with **C++ Core Check**. Click **Save As...**, give your custom rule set a name, click **Save**, then click **Apply**.

This page intentionally left blank

1

Intro to Computers and C++



Objectives

In this chapter, you'll:

- Learn computer hardware, software and Internet basics.
- Understand a data hierarchy from bits to databases.
- Understand the types of programming languages.
- Understand the strengths of C++ and other languages.
- Be introduced to the C++ standard library of reusable components.
- Compile and run a C++ application using our three preferred compilers and Docker containers.
- Be introduced to object-technology concepts used in the objects-natural case studies (Chapters 2–9) and discussed in detail in the object-oriented programming Chapters 9–11.
- Understand how concurrent programming helps maximize performance on multi-core processors.
- Be introduced to big data and data science.
- Learn about exciting recent developments in computing, including the Metaverse, artificial intelligence and related technologies.

Outline

1.1 Introduction

Welcome to C++—one of the world's most senior computer programming languages and, according to various programming-language rankings, one of the world's most popular.[1,2,3] You're probably familiar with many of the powerful tasks computers perform. This textbook provides an intensive, hands-on experience in which you'll write C++ instructions that command computers to perform many of those and other tasks. **Software**—the C++ instructions you write, which also are called **code**—controls **hardware** (that is, computers and related devices).

1. "TIOBE Index." Accessed March 18, 2023. `https://www.tiobe.com/tiobe-index/`.
2. "Top Programming Languages 2022." Accessed March 18, 2023. `https://spectrum.ieee.org/top-programming-languages-2022`.
3. "PYPL PopularitY of Programming Language." Accessed March 18, 2023. `https://pypl.github.io/PYPL.html`.

C++ is widely used in industry.[4] Portions of today's most popular desktop operating systems—Windows[5] and macOS[6]—are written in C++. Many applications and systems are partially written in C++, including popular web browsers (e.g., Google Chrome[7] and Mozilla Firefox[8]), database management systems (e.g., Microsoft SQL Server[9], Oracle[10], MySQL[11]), Adobe's creative applications, Amazon.com, Bloomberg, Facebook, various Microsoft apps (including Office and Visual Studio) and much more.[12]

The chapter introduces terminology and concepts that lay the groundwork for the C++ programming you'll learn, beginning in Chapter 2. We introduce hardware and software concepts. We overview a sample data hierarchy—from 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 discuss the types of programming languages. We introduce the C++ standard library and various C++ "open-source" libraries that help you avoid "reinventing the wheel"—you'll write only modest numbers of C++ instructions to make these libraries perform powerful tasks.

We overview additional technologies you'll likely use as you build software in your career. We discuss the Internet, the web, the cloud and the Internet of Things (IoT). We introduce the Metaverse and its related technologies—blockchain, cryptocurrencies, NFTs, Web3, augmented reality, virtual reality and mixed reality. Finally, we introduce big data, data science and artificial intelligence (AI) technologies—machine learning, deep learning and generative AIs like the recently introduced to much excitement ChatGPT and Dall-E 2.

Compilers

You can compile, build and run C++ applications in many development environments. You'll work through one or more of Section 1.11's test-drives showing how to compile and execute C++ code using:

- Microsoft Visual Studio Community edition for Windows.
- GNU `g++` in a macOS Terminal window or Linux shell.
- LLVM `clang++` in a macOS Terminal window or a Linux shell.

4. "C++." Wikipedia. Wikimedia Foundation. Accessed March 18, 2023. `https://en.wikipedia.org/wiki/C%2B%2B`.
5. "Windows 11." Wikipedia. Wikimedia Foundation, Accessed March 18, 2023. `https://en.wikipedia.org/wiki/Windows_11`.
6. "macOS." Wikipedia. Wikimedia Foundation, Accessed March 18, 2023. `https://en.wikipedia.org/wiki/MacOS`.
7. "Google Chrome." Wikipedia. Wikimedia Foundation. Accessed March 18, 2023. `https://en.wikipedia.org/wiki/Google_Chrome`.
8. "Firefox." Wikipedia. Wikimedia Foundation. Accessed March 18, 2023. `https://en.wikipedia.org/wiki/Firefox`.
9. "Microsoft SQL Server." Wikipedia. Wikimedia Foundation. Accessed March 18, 2023. `https://en.wikipedia.org/wiki/Microsoft_SQL_Server`.
10. "Oracle Database." Wikipedia. Wikimedia Foundation. Accessed March 18, 2023. `https://en.wikipedia.org/wiki/Oracle_Database`.
11. "MySQL." Wikipedia. Wikimedia Foundation. Accessed March 18, 2023. `https://en.wikipedia.org/wiki/MySQL`.
12. Bjarne Stroustrup, "C++ Applications." Accessed March 18. 2023. `https://www.stroustrup.com/applications.html`.

In addition, we'll introduce Docker and show how to use `g++` and `clang++` in Docker containers that can execute on Windows, macOS or Linux. Docker is particularly important for this book's macOS users. Apple's Xcode environment does not yet support several key C++20 features we use throughout the book. Docker will enable macOS users (and Windows and Linux users) to compile C++ programs with the latest `g++` and `clang++` versions. You may want to read only the test-drive(s) required for your course or projects in industry.

This Book's Architecture

Before you "dig in," we recommend getting a "40,000-foot" view of the book's architecture to understand where you're headed as you prepare to learn the powerful language that is C++20. To do so, we recommend reviewing the following items:

- In the Preface, the one-page, full-color Table of Contents diagram presents a high-level overview of the book's architecture. You can view a scalable PDF version of this diagram at

  ```
  https://deitel.com/cpphtp11
  ```

- The preceding website also contains a concise introduction to the book, a bullet list of its key features and testimonial comments from university professors and C++ subject-matter experts who reviewed the prepublication manuscript.
- The Preface presents the "soul of the book" and our approach to Modern C++ programming. We introduce the early chapters' "objects-natural approach," in which you'll use small numbers of simple C++ statements to make powerful existing classes perform significant tasks—long before you learn how to create your own custom classes in Chapter 9. Be sure to read the Tour of the Book, which points out the key features of each chapter and enumerates the book's 50 more substantial case studies and case-study exercises. As you read the Tour, you might also want to refer to the Table of Contents diagram.

1.2 Hardware

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

A computer's hardware consists of various physical devices, such as the keyboard, screen, mouse, solid-state disks, memory 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 ordinary sand. Silicon-chip technology has made computing so economical that computers and computerized devices have become commodities.

1.2.1 Computer Organization

Regardless of physical differences, computers have 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, barcodes or QR codes,
- reading data from secondary storage devices (such as solid-state 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 acceleration) 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®, Microsoft® Cortana® and Google Home™.

Output Unit

This "shipping" section takes the 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),
- printed on 3D printers,
- 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, 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 Meta Quest® 2, Meta Quest® Pro, Sony® PlayStation® VR and Samsung Gear VR®, and mixed reality devices like Magic Leap® 2 and Microsoft HoloLens® 2.

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 the output unit can place it on output devices. 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 32 GB is most common. GB stands for gigabytes; a **gigabyte** is approximately one billion bytes. A **byte** is eight bits. A **bit** (short for "*bi*nary digi*t*") 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 **multi-core processors** that economically implement multiple processors on a single integrated circuit chip. Such processors can perform many operations simultaneously. We say more about these in Section 1.2.2.

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 is much less than RAM. Examples of secondary storage devices include solid-state drives (SSDs), USB flash drives and read/write Blu-ray drives. Many current drives hold terabytes (TB) of data—each **terabyte** is approximately one trillion bytes. Typical desktop and notebook-computer secondary storage devices hold up to 8 TB, and some recent desktop-computer hard drives hold up to 26 TB.[13] The largest commercial SSD holds up to 100 TB (and costs about $40,000).[14]

13. "History of hard disk drives." Wikipedia. Wikimedia Foundation. Accessed March 18, 2023. `https://en.wikipedia.org/wiki/History_of_hard_disk_drives`.
14. Desire Athow. "Largest SSDs and hard drives of 2023." Accessed March 18, 2023. `https://www.techradar.com/best/large-hard-drives-and-ssds`.

Checkpoint

1 *(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.

2 *(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: multi-core.

1.2.2 Moore's Law, Multi-Core Processors and Concurrent Programming

Many of today's personal computers can perform billions of calculations in one second—more than a human can perform in a lifetime. **Supercomputers** already perform over one million trillion (a quintillion) instructions per second! As of November 2022, the USA's Frontier (from Hewlett Packard Enterprise) is the world's fastest supercomputer[15]—it can perform 1.102 quintillion calculations per second (1.102 exaflops)![16] For perspective, this supercomputer can perform in one second almost 140 million calculations for every person on the planet![17] And supercomputing "upper limits" are growing quickly.

Moore's Law

You probably expect to pay at least a little more every year 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.

Perf

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

- the amount of memory that computers have for programs and data,
- the amount of secondary 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.

Key executives at computer-processor companies NVIDIA and ARM have indicated that Moore's Law no longer applies. However, Intel's CEO says it is "alive and well"—

15. "Top 500." Wikipedia. Wikimedia Foundation. Accessed March 18, 2023. `https://en.wikipedia.org/wiki/TOP500#TOP_500`.
16. "Flops." Wikipedia. Wikimedia Foundation. Accessed March 18, 2023. `https://en.wikipedia.org/wiki/FLOPS`.
17. For perspective on how far computing performance has come, consider this: In his early computing days in the 1960s, one of the authors, 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.

they are working on manufacturing advances that will enable more transistors on each chip.[18,19] Computer processing power continues to increase but relies on new processor designs, such as multi-core processors (Section 1.2.1).

Multi-Core Processors and Performance

Most computers today have multi-core processors that economically implement multiple processors on a single integrated circuit chip. 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. Our primary testing computer uses an eight-core Intel processor. Apple's recent M2 Pro and M2 Max processors have 12-core CPUs. In addition, the top-of-the-line M2 Pro has a 19-core graphics processing unit (GPU), while the top-of-the-line M2 Max processor has a 28-core GPU. Both have a 16-core "neural engine" for machine learning.[20] Intel is working on processors with up to 80.[21] AMD is working on processors with 192 and 256 cores.[22] The number of cores will continue to grow. Today's most powerful GPUs used in high-end gaming computers and artificial-intelligence applications often have thousands of cores in their GPUs. For example, at the time of this writing, the forthcoming NVIDIA RTX 4090 Ti is expected to have 18,176 cores![23]

In multi-core systems, the hardware can put multiple processors to work truly simultaneously on different parts of your task, enabling your program to complete faster. **Taking full advantage of multi-core architecture requires writing multithreaded applications** (Chapter 17). When a program splits tasks into separate threads, a multi-core system can run those threads in parallel when a sufficient number of cores is available.

Interest in multithreading is rising quickly because of the proliferation of multi-core systems. Standard C++ multithreading was one of the most significant updates introduced in C++11. Each subsequent C++ standard has added higher-level capabilities to simplify multithreaded application development. Chapter 17, Parallel Algorithms and Concurrency: A High-Level View, discusses creating and managing multithreaded C++ applications. Chapter 18 introduces C++20 coroutines, which enable concurrent programming with a simple sequential-like coding style.

18. Esther Shein. "Moore's Law turns 55: Is it still relevant?" April 17. 2020. Accessed March 18, 2023. https://www.techrepublic.com/article/moores-law-turns-55-is-it-still-relevant.
19. Leswing, Kif. "Intel says Moore's Law is still alive and well. Nvidia says it's ended." September 27, 2022. Accessed March 18, 2023. https://www.cnbc.com/2022/09/27/intel-says-moores-law-is-still-alive-nvidia-says-its-ended.html.
20. "Apple unveils M2 Pro and M2 Max: next-generation chips for next-level workflows," January 23, 2023. Accessed March 18, 2023. https://www.apple.com/newsroom/2023/01/apple-unveils-m2-pro-and-m2-max-next-generation-chips-for-next-level-workflows/.
21. Anton Shilov, "Intel's Sapphire Rapids Could Have 72-80 Cores, According to New Die Shots." April 30, 2021. Accessed March 18, 2023. https://www.tomshardware.com/news/intel-sapphire-rapids-could-feature-80-cores.
22. Anthony Garreffa, "AMD EPYC 'Venice' CPU: Zen 6 with 256 cores, 512 threads... or MORE!" May 22, 2022. Accessed March 18, 2023. https://www.tweaktown.com/news/85915/amd-epyc-venice-cpu-zen-6-with-256-cores-512-threads-or-more/index.html.
23. "NVIDIA GeForce RTX 4090 Ti rumored to feature 18176 cores and 24GB/24Gbps memory." Accessed March 18, 2023. https://videocardz.com/newz/nvidia-geforce-rtx-4090-ti-rumored-to-feature-18176-cores-and-24gb-24gbps-memory.

Computing Power Over the Years

Data is getting more massive, and so is the computing power needed for processing it. Today's processor performance is often measured in terms of **FLOPS (floating-point operations per second)**. In the early to mid-1990s, the fastest supercomputer speeds were measured in gigaflops (10^9 FLOPS). By the late 1990s, Intel produced the first teraflop (10^{12} FLOPS) supercomputers. In the early-to-mid 2000s, speeds reached hundreds of teraflops, then in 2008, IBM released the first petaflop (10^{15} FLOPS) supercomputer. As of November 2022, Hewlett Packard Enterprise's Frontier is the world's fastest supercomputer[24,25]—it can perform 1.102 quintillion calculations per second (1.102 *exaflops*—10^{18} FLOPS)![26] Companies like IBM also are working toward exaflop supercomputers.[27]

The **quantum computers** now under development theoretically could operate at 18,000,000,000,000,000,000 times the speed of today's "conventional computers"![28] This number is so extraordinary that in one second, theoretically, a quantum computer could do staggeringly more calculations than the total that have been done by all computing devices since the beginning of time. This almost unimaginable computing power could wreak havoc with blockchain-based cryptocurrencies like Bitcoin. Engineers are already rethinking blockchain[29] to prepare for such massive increases in computing power.[30]

Computing power costs continue to decline, especially with advancements in processor architecture and cloud computing. People used to ask the question, "How much computing power do I need on my local system to deal with my peak processing needs?" Today, that thinking has shifted to "Can I quickly carve out in the cloud what I need temporarily for my most demanding computing workloads?" You pay for only what you use to accomplish a given task.

Checkpoint

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 ***(Fill-In)*** Taking full advantage of multi-core architecture requires writing _____.
Answer: multithreaded applications.

---

24. "Top 500." Wikipedia. Wikimedia Foundation. Accessed March 18, 2023. `https://en.wikipedia.org/wiki/TOP500#TOP_500`.
25. "Frontier (supercomputer)." Wikipedia. Wikimedia Foundation. Accessed March 18, 2023. `https://en.wikipedia.org/wiki/Frontier_(supercomputer)`.
26. "Flops." Wikipedia. Wikimedia Foundation. Accessed March 18, 2023. `https://en.wikipedia.org/wiki/FLOPS`.
27. "A new supercomputing-powered weather model may ready us for Exascale." Accessed March 18, 2023. `https://www.ibm.com/blogs/research/2017/06/supercomputing-weather-model-exascale/`.
28. Norbert Biedrzycki, "Only God can count that fast — the world of quantum computing." May 12, 2017. Accessed March 18, 2023. `https://medium.com/@n.biedrzycki/only-god-can-count-that-fast-the-world-of-quantum-computing-406a0a91fcf4`.
29. "Blockchain." Wikipedia. Wikimedia Foundation. Accessed March 18, 2023. `https://en.wikipedia.org/wiki/Blockchain`.
30. Nathana Sharma, "Is Quantum Computing an Existential Threat to Blockchain Technology?" November 5, 2017. Accessed March 18, 2023. `https://singularityhub.com/2017/11/05/is-quantum-computing-an-existential-threat-to-blockchain-technology/`.