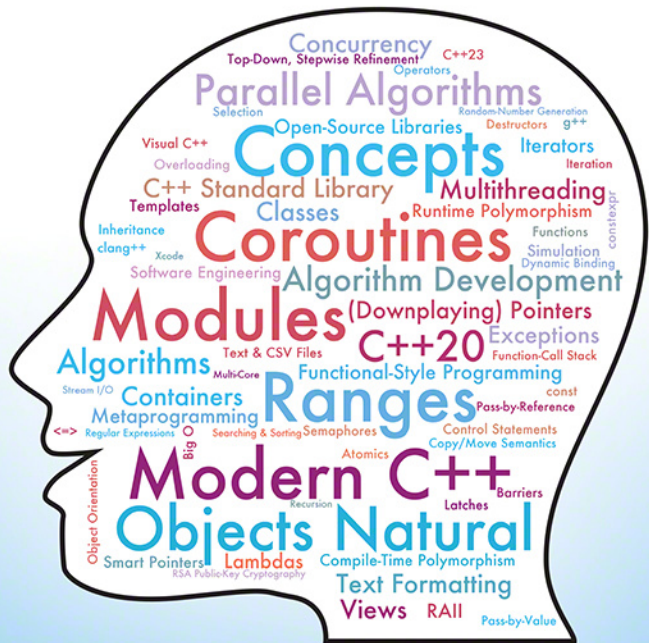




HOW TO PROGRAM

An Objects-Natural Approach



PAUL DEITEL
HARVEY DEITEL

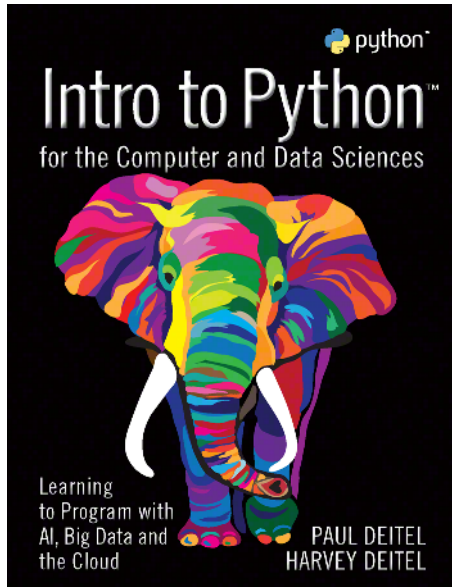
11
ELEVENTH
EDITION



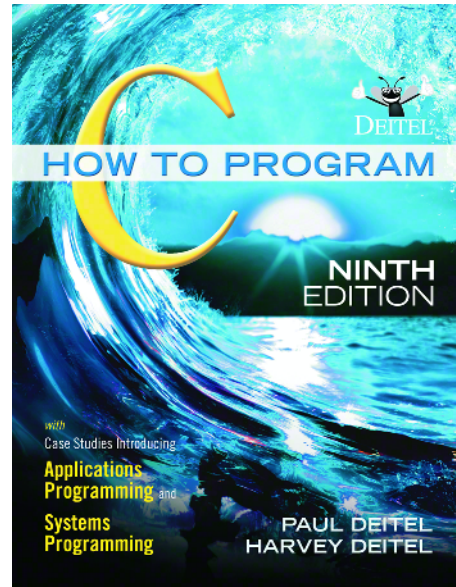
An Objects-Natural Approach



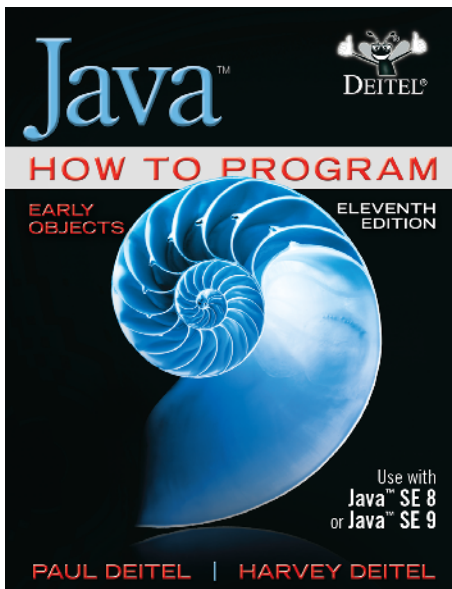
Recent Deitel Textbooks



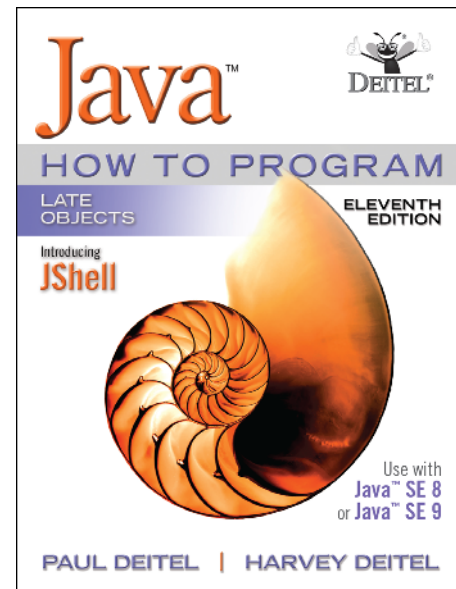
<https://deitel.com/pycds>



<https://deitel.com/cht9>



<https://deitel.com/jhtp11EOV>



<https://deitel.com/jhtp11LOV>

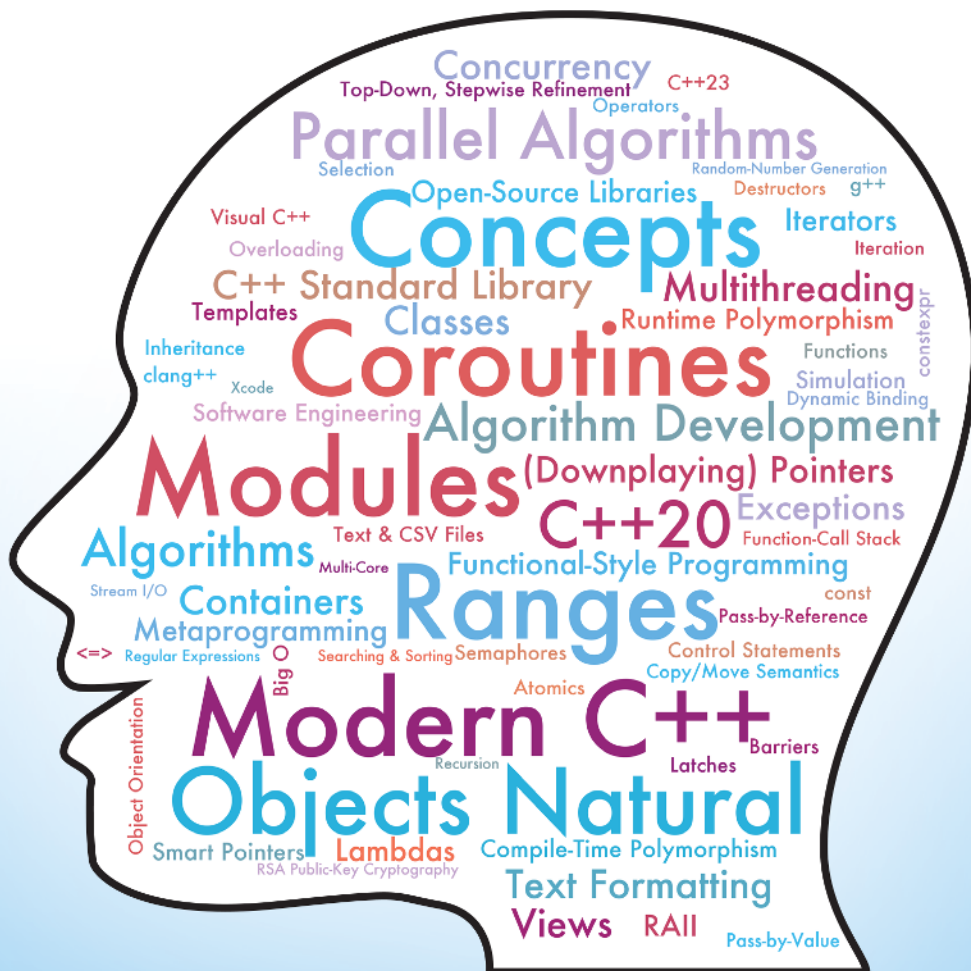


PAUL DEITEL
Deitel & Associates, Inc.

HARVEY DEITEL
Deitel & Associates, Inc.

HOW TO PROGRAM

An Objects-Natural Approach



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.

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

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

Microsoft® and Windows® are registered trademarks of the Microsoft Corporation in the U.S.A. and other countries. This book is not sponsored or endorsed by or affiliated with the Microsoft Corporation.

Copyright © 2024 by Pearson Education, Inc. or its affiliates, 221 River Street, Hoboken, NJ 07030. All Rights Reserved. Manufactured in the United States of America. This publication is protected by copyright, and permission should be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise. For information regarding permissions, request forms, and the appropriate contacts within the Pearson Education Global Rights and Permissions department, please visit www.pearsoned.com/permissions/.

Acknowledgments of third-party content appear on the appropriate page within the text.

PEARSON and REVEL are exclusive trademarks owned by Pearson Education, Inc. or its affiliates in the U.S. and/or other countries.

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

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

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.



xxiii

iii

I	Intro to Computers and C++	I
1.1	Introduction	2
1.2	Hardware	4
1.2.1	Computer Organization	4
1.2.2	Moore's Law, Multi-Core Processors and Concurrent Programming	7
1.3	Data Hierarchies	10
1.4	Machine Languages, Assembly Languages and High-Level Languages	13
1.5	Operating Systems	14
1.6	C and C++	18
1.7	C++ Standard Library and Open-Source Libraries	19
1.8	Other Popular Programming Languages	21
1.9	Introduction to Object Orientation	23
1.10	Simplified View of a C++ Development Environment	25
1.11	Test-Driving a C++20 Application Various Ways	28
1.11.1	Compiling and Running on Windows with Visual Studio Community Edition	29
1.11.2	Compiling and Running with GNU C++ on Linux	32
1.11.3	Compiling and Running with g++ in the GCC Docker Container	34
1.11.4	Compiling and Running with clang++ in a Docker Container	36
1.11.5	Compiling and Running with Xcode on macOS	37
1.12	Internet, World Wide Web, the Cloud and IoT	41
1.12.1	The Internet: A Network of Networks	41
1.12.2	The World Wide Web: Making the Internet User-Friendly	42
1.12.3	The Cloud	42
1.12.4	The Internet of Things (IoT)	42
1.12.5	Edge Computing	43
1.12.6	Mashups	43
1.13	Metaverse	44
1.13.1	Virtual Reality (VR)	44
1.13.2	Augmented Reality (AR)	45
1.13.3	Mixed Reality	46
1.13.4	Blockchain	46
1.13.5	Bitcoin and Cryptocurrency	46

1.13.6	Ethereum	47
1.13.7	Non-Fungible Tokens (NFTs)	47
1.13.8	Web3	47
1.14	Software Development Technologies	48
1.15	How Big Is Big Data?	49
1.15.1	Big-Data Analytics	52
1.15.2	Data Science and Big Data Are Making a Difference: Use Cases	53
1.16	AI—at the Intersection of Computer Science and Data Science	54
1.16.1	Artificial Intelligence (AI)	54
1.16.2	Artificial General Intelligence (AGI)	55
1.16.3	Artificial Intelligence Milestones	55
1.16.4	Machine Learning	56
1.16.5	Deep Learning	56
1.16.6	Reinforcement Learning	57
1.16.7	Generative AI—ChatGPT and Dall-E 2	57
1.17	Wrap-Up	59
2	Intro to C++20 Programming	65
2.1	Introduction	66
2.2	First Program in C++: Displaying a Line of Text	66
2.3	Modifying Our First C++ Program	70
2.4	Another C++ Program: Adding Integers	71
2.5	Memory Concepts	75
2.6	Arithmetic	76
2.7	Decision Making: Equality and Relational Operators	79
2.8	Objects Natural Case Study: Creating and Using Objects of Standard-Library Class <code>string</code>	83
2.9	Wrap-Up	86
3	Algorithm Development and Control Statements: Part I	93
3.1	Introduction	94
3.2	Algorithms	95
3.3	Pseudocode	95
3.4	Control Structures	96
3.4.1	Sequence Structure	97
3.4.2	Selection Statements	98
3.4.3	Iteration Statements	98
3.4.4	Summary of Control Statements	99
3.5	<code>if</code> Single-Selection Statement	100
3.6	<code>if...else</code> Double-Selection Statement	101
3.6.1	Nested <code>if...else</code> Statements	102
3.6.2	Blocks	104
3.6.3	Conditional Operator (<code>?:</code>)	104
3.7	<code>while</code> Iteration Statement	105

3.8	Formulating Algorithms: Counter-Controlled Iteration	107
3.8.1	Pseudocode Algorithm with Counter-Controlled Iteration	107
3.8.2	Implementing Counter-Controlled Iteration	108
3.8.3	Integer Division and Truncation	110
3.8.4	Arithmetic Overflow	110
3.8.5	Input Validation	110
3.9	Formulating Algorithms: Sentinel-Controlled Iteration	111
3.9.1	Top-Down, Stepwise Refinement: The Top and First Refinement	112
3.9.2	Proceeding to the Second Refinement	112
3.9.3	Implementing Sentinel-Controlled Iteration	114
3.9.4	Mixed-Type Expressions and Implicit Type Promotions	116
3.9.5	Formatting Floating-Point Numbers	117
3.10	Formulating Algorithms: Nested Control Statements	118
3.10.1	Problem Statement	118
3.10.2	Top-Down, Stepwise Refinement: Pseudocode Representation of the Top	119
3.10.3	Top-Down, Stepwise Refinement: First Refinement	119
3.10.4	Top-Down, Stepwise Refinement: Second Refinement	120
3.10.5	Complete Second Refinement of the Pseudocode	120
3.10.6	Implementing the Program	121
3.10.7	Preventing Narrowing Conversions with Braced Initialization	123
3.11	Compound Assignment Operators	125
3.12	Increment and Decrement Operators	125
3.13	Fundamental Types Are Not Portable	128
3.14	Objects Natural Case Study: Super-Sized Integers	129
3.15	Wrap-Up	134

4 Control Statements, Part 2 **I45**

4.1	Introduction	146
4.2	Essentials of Counter-Controlled Iteration	146
4.3	for Iteration Statement	148
4.4	Examples Using the for Statement	151
4.5	Application: Summing Even Integers; Introducing C++20 Text Formatting	152
4.6	Application: Compound-Interest Calculations; Introducing Format Specifiers	153
4.7	do...while Iteration Statement	158
4.8	switch Multiple-Selection Statement	159
4.9	Selection Statements with Initializers	165
4.10	break and continue Statements	167
4.11	Logical Operators	169
4.11.1	Logical AND (&&) Operator	169
4.11.2	Logical OR () Operator	170
4.11.3	Short-Circuit Evaluation	170
4.11.4	Logical Negation (!) Operator	171
4.11.5	Example: Producing Logical-Operator Truth Tables	171

4.12	Confusing the Equality (==) and Assignment (=) Operators	173
4.13	Structured-Programming Summary	175
4.14	Objects Natural Case Study: Precise Monetary Calculations with the Boost Multiprecision Library	180
4.15	Wrap-Up	183

5 Functions and an Intro to Function Templates 189

5.1	Introduction	190
5.2	C++ Program Components	191
5.3	Math Library Functions	192
5.4	Function Definitions and Function Prototypes	194
5.5	Order of Evaluation of a Function's Arguments	197
5.6	Function-Prototype and Argument-Coercion Notes	197
5.6.1	Function Signatures and Function Prototypes	198
5.6.2	Argument Coercion	198
5.6.3	Argument-Promotion Rules and Implicit Conversions	198
5.7	C++ Standard Library Headers	200
5.8	Case Study: Random-Number Generation	203
5.8.1	Rolling a Six-Sided Die	204
5.8.2	Rolling a Six-Sided Die 60,000,000 Times	205
5.8.3	Seeding the Random-Number Generator	207
5.8.4	Seeding the Random-Number Generator with <code>random_device</code>	208
5.9	Case Study: Game of Chance; Introducing Scoped <code>enums</code>	209
5.10	Function-Call Stack and Activation Records	214
5.11	Inline Functions	217
5.12	References and Reference Parameters	219
5.13	Default Arguments	222
5.14	Function Overloading	224
5.15	Function Templates	227
5.16	Recursion	229
5.16.1	Factorials	230
5.16.2	Recursive Factorial Example	230
5.16.3	Recursive Fibonacci Series Example	234
5.16.4	Recursion vs. Iteration	237
5.17	Scope Rules	239
5.18	Unary Scope Resolution Operator	244
5.19	<code>Lnfylun Lhqtomh Wjtz Qarcv: Qjwazkrplm xzz Xndmwwqhlz</code>	245
5.20	Wrap-Up	248

6 arrays, vectors, Ranges and Functional-Style Programming 259

6.1	Introduction	260
6.2	arrays	261
6.3	Declaring arrays	262
6.4	Initializing array Elements in a Loop	262

6.5	Initializing an array with an Initializer List	265
6.6	Range-Based <code>for</code> Statement	267
6.7	Calculating array Element Values and an Intro to <code>constexpr</code>	269
6.8	Totaling array Elements	271
6.9	Using a Primitive Bar Chart to Display array Data Graphically	272
6.10	Using array Elements as Counters	274
6.11	Using arrays to Summarize Survey Results	275
6.12	Sorting and Searching arrays	277
6.13	Multidimensional arrays	279
6.14	Intro to Functional-Style Programming	283
6.14.1	What vs. How	283
6.14.2	Passing Functions as Arguments to Other Functions: Introducing Lambda Expressions	284
6.14.3	Filter, Map and Reduce: Intro to C++20's Ranges Library	286
6.15	Objects-Natural Case Study: C++ Standard Library Class Template <code>vector</code>	290
6.16	Wrap-Up	297
7	(Downplaying) Pointers in Modern C++	309
7.1	Introduction	310
7.2	Pointer Variable Declarations and Initialization	312
7.2.1	Declaring Pointers	312
7.2.2	Initializing Pointers	312
7.2.3	Null Pointers	312
7.3	Pointer Operators	313
7.3.1	Address (<code>&</code>) Operator	313
7.3.2	Indirection (<code>*</code>) Operator	314
7.3.3	Using the Address (<code>&</code>) and Indirection (<code>*</code>) Operators	315
7.4	Pass-by-Reference with Pointers	316
7.5	Built-In Arrays	320
7.5.1	Declaring and Accessing a Built-In Array	320
7.5.2	Initializing Built-In Arrays	321
7.5.3	Passing Built-In Arrays to Functions	322
7.5.4	Declaring Built-In Array Parameters	322
7.5.5	Standard Library Functions <code>begin</code> and <code>end</code>	323
7.5.6	Built-In Array Limitations	323
7.6	Using C++20 <code>to_array</code> to Convert a Built-in Array to a <code>std::array</code>	324
7.7	Using <code>const</code> with Pointers and the Data Pointed To	325
7.7.1	Using a Nonconstant Pointer to Nonconstant Data	326
7.7.2	Using a Nonconstant Pointer to Constant Data	326
7.7.3	Using a Constant Pointer to Nonconstant Data	327
7.7.4	Using a Constant Pointer to Constant Data	327
7.8	<code>sizeof</code> Operator	329
7.9	Pointer Expressions and Pointer Arithmetic	331
7.9.1	Adding Integers to and Subtracting Integers from Pointers	332

7.9.2	Subtracting One Pointer from Another	333
7.9.3	Pointer Assignment	333
7.9.4	Cannot Dereference a <code>void*</code> Pointer	333
7.9.5	Comparing Pointers	334
7.10	Objects-Natural Case Study: C++20 <code>spans</code> —Views of Contiguous Container Elements	334
7.11	A Brief Intro to Pointer-Based Strings	340
7.11.1	Command-Line Arguments	341
7.11.2	Revisiting C++20's <code>to_array</code> Function	342
7.12	Looking Ahead to Other Pointer Topics	344
7.13	Wrap-Up	344

8 strings, string_views, Text Files, CSV Files and Regex 357

8.1	Introduction	358
8.2	<code>string</code> Assignment and Concatenation	359
8.3	Comparing strings	361
8.4	Substrings	363
8.5	Swapping strings	364
8.6	<code>string</code> Characteristics	365
8.7	Finding Substrings and Characters in a <code>string</code>	367
8.8	Replacing and Erasing Characters in a <code>string</code>	370
8.9	Inserting Characters into a <code>string</code>	372
8.10	Numeric Conversions	373
8.11	<code>string_view</code>	374
8.12	Files and Streams	377
8.13	Creating a Sequential File	378
8.14	Reading Data from a Sequential File	381
8.15	Reading and Writing Quoted Text	384
8.16	Updating Sequential Files	385
8.17	String Stream Processing	386
8.18	Raw String Literals	389
8.19	Objects-Natural Data Science Case Study: Reading and Analyzing a CSV File Containing <i>Titanic</i> Disaster Data	390
8.19.1	Using <code>rapidcsv</code> to Read the Contents of a CSV File	390
8.19.2	Reading and Analyzing the <i>Titanic</i> Disaster Dataset	392
8.20	Objects-Natural Data Science Case Study: Intro to Regular Expressions	399
8.20.1	Matching Complete Strings to Patterns	400
8.20.2	Replacing Substrings	405
8.20.3	Searching for Matches	405
8.21	Wrap-Up	408

9 Custom Classes 431

9.1	Introduction	432
-----	--------------	-----

9.2	Test-Driving an Account Object	433
9.3	Account Class with a Data Member and <i>Set</i> and <i>Get</i> Member Functions	435
9.3.1	Class Definition	435
9.3.2	Access Specifiers <i>private</i> and <i>public</i>	437
9.4	Account Class: Custom Constructors	438
9.5	Software Engineering with <i>Set</i> and <i>Get</i> Member Functions	442
9.6	Account Class with a Balance	444
9.7	Time Class Case Study: Separating Interface from Implementation	447
9.7.1	Interface of a Class	448
9.7.2	Separating the Interface from the Implementation	448
9.7.3	Class Definition	449
9.7.4	Member Functions	450
9.7.5	Including the Class Header in the Source-Code File	450
9.7.6	Scope Resolution Operator (<i>::</i>)	451
9.7.7	Member Function <i>setTime</i> and Throwing Exceptions	451
9.7.8	Member Functions <i>to24HourString</i> and <i>to12HourString</i>	451
9.7.9	Implicitly Inlining Member Functions	452
9.7.10	Member Functions vs. Global Functions	452
9.7.11	Using Class <i>Time</i>	452
9.7.12	Object Size	454
9.8	Compilation and Linking Process	454
9.9	Class Scope and Accessing Class Members	455
9.10	Access Functions and Utility Functions	456
9.11	Time Class Case Study: Constructors with Default Arguments	457
9.11.1	Class <i>Time</i>	457
9.11.2	Overloaded Constructors and Delegating Constructors	462
9.12	Destructors	463
9.13	When Constructors and Destructors Are Called	464
9.14	Time Class Case Study: A Subtle Trap — Returning a Reference or a Pointer to a <i>private</i> Data Member	468
9.15	Default Assignment Operator	470
9.16	<i>const</i> Objects and <i>const</i> Member Functions	472
9.17	Composition: Objects as Members of Classes	474
9.18	<i>friend</i> Functions and <i>friend</i> Classes	480
9.19	The <i>this</i> Pointer	482
9.19.1	Implicitly and Explicitly Using the <i>this</i> Pointer to Access an Object's Data Members	482
9.19.2	Using the <i>this</i> Pointer to Enable Cascaded Function Calls	484
9.20	<i>static</i> Class Members: Classwide Data and Member Functions	487
9.21	Aggregates in C++20	492
9.21.1	Initializing an Aggregate	493
9.21.2	C++20 Designated Initializers	493
9.22	Concluding Our Objects Natural Case Study Track: Studying the Vigenère Secret-Key Cipher Implementation	494
9.23	Wrap-Up	507

10 OOP: Inheritance and Runtime Polymorphism 529

10.1	Introduction	530
10.2	Base Classes and Derived Classes	532
10.2.1	CommunityMember Class Hierarchy	533
10.2.2	Shape Class Hierarchy and public Inheritance	534
10.3	Relationship Between Base and Derived Classes	535
10.3.1	Creating and Using a SalariedEmployee Class	535
10.3.2	Creating a SalariedEmployee–SalariedCommissionEmployee Inheritance Hierarchy	538
10.4	Constructors and Destructors in Derived Classes	544
10.5	Intro to Runtime Polymorphism: Polymorphic Video Game	545
10.6	Relationships Among Objects in an Inheritance Hierarchy	546
10.6.1	Invoking Base-Class Functions from Derived-Class Objects	547
10.6.2	Aiming Derived-Class Pointers at Base-Class Objects	549
10.6.3	Derived-Class Member-Function Calls via Base-Class Pointers	550
10.7	Virtual Functions and Virtual Destructors	551
10.7.1	Why virtual Functions Are Useful	551
10.7.2	Declaring virtual Functions	552
10.7.3	Invoking a virtual Function	553
10.7.4	virtual Functions in the SalariedEmployee Hierarchy	553
10.7.5	virtual Destructors	557
10.7.6	final Member Functions and Classes	557
10.8	Abstract Classes and Pure virtual Functions	558
10.8.1	Pure virtual Functions	559
10.8.2	Device Drivers: Polymorphism in Operating Systems	559
10.9	Case Study: Payroll System Using Runtime Polymorphism	560
10.9.1	Creating Abstract Base Class Employee	561
10.9.2	Creating Concrete Derived Class SalariedEmployee	563
10.9.3	Creating Concrete Derived Class CommissionEmployee	565
10.9.4	Demonstrating Runtime Polymorphic Processing	567
10.10	Runtime Polymorphism, Virtual Functions and Dynamic Binding “Under the Hood”	570
10.11	Program to an Interface, Not an Implementation	573
10.11.1	Rethinking the Employee Hierarchy—CompensationModel Interface	575
10.11.2	Class Employee	575
10.11.3	CompensationModel Implementations	577
10.11.4	Testing the New Hierarchy	579
10.11.5	Dependency Injection Design Benefits	580
10.12	Wrap-Up	581

11 Operator Overloading, Copy/Move Semantics and Smart Pointers 587

11.1	Introduction	588
11.2	Using the Overloaded Operators of Standard Library Class string	590

11.3	Operator Overloading Fundamentals	596
11.3.1	Operator Overloading Is Not Automatic	596
11.3.2	Operators That Cannot Be Overloaded	596
11.3.3	Operators That You Do Not Have to Overload	596
11.3.4	Rules and Restrictions on Operator Overloading	597
11.4	(Downplaying) Dynamic Memory Management with <code>new</code> and <code>delete</code>	598
11.5	Modern C++ Dynamic Memory Management: RAII and Smart Pointers	600
11.5.1	Smart Pointers	601
11.5.2	Demonstrating <code>unique_ptr</code>	601
11.5.3	<code>unique_ptr</code> Ownership	603
11.5.4	<code>unique_ptr</code> to a Built-In Array	603
11.6	<code>MyArray</code> Case Study: Crafting a Valuable Class with Operator Overloading	604
11.6.1	Special Member Functions	605
11.6.2	Using Class <code>MyArray</code>	606
11.6.3	<code>MyArray</code> Class Definition	615
11.6.4	Constructor That Specifies a <code>MyArray</code> 's Size	617
11.6.5	Passing a Braced Initializer to a Constructor	618
11.6.6	Copy Constructor and Copy Assignment Operator	619
11.6.7	Move Constructor and Move Assignment Operator	622
11.6.8	Destructor	626
11.6.9	<code>toString</code> and <code>size</code> Functions	626
11.6.10	Overloading the Equality (<code>==</code>) and Inequality (<code>!=</code>) Operators	627
11.6.11	Overloading the Subscript (<code>[]</code>) Operator	629
11.6.12	Overloading the Unary <code>bool</code> Conversion Operator	630
11.6.13	Overloading the Preincrement Operator	631
11.6.14	Overloading the Postincrement Operator	632
11.6.15	Overloading the Addition Assignment Operator (<code>+=</code>)	633
11.6.16	Overloading the Binary Stream Extraction (<code>>></code>) and Stream Insertion (<code><<</code>) Operators	633
11.6.17	<code>friend</code> Function <code>swap</code>	636
11.7	C++20 Three-Way Comparison Operator (<code><=></code>)	636
11.8	Converting Between Types	640
11.9	<code>explicit</code> Constructors and Conversion Operators	641
11.10	Overloading the Function Call Operator (<code>()</code>)	644
11.11	Wrap-Up	644

12 Exceptions and a Look Forward to Contracts 653

12.1	Introduction	654
12.2	Exception-Handling Flow of Control	658
12.2.1	Defining a Custom Exception Class	658
12.2.2	Demonstrating Exception Handling	659
12.2.3	Enclosing Code in a <code>try</code> Block	660
12.2.4	Defining a <code>catch</code> Handler for <code>DivideByZeroExceptions</code>	661
12.2.5	Termination Model of Exception Handling	662
12.2.6	Flow of Control When the User Enters a Nonzero Denominator	663
12.2.7	Flow of Control When the User Enters a Zero Denominator	663

12.3	Exception Safety Guarantees and <code>noexcept</code>	664
12.4	Rethrowing an Exception	665
12.5	Stack Unwinding and Uncaught Exceptions	667
12.6	When to Use Exception Handling	669
12.6.1	<code>assert</code> Macro	671
12.6.2	Failing Fast	671
12.7	Constructors, Destructors and Exception Handling	672
12.7.1	Throwing Exceptions from Constructors	672
12.7.2	Catching Exceptions in Constructors via Function <code>try</code> Blocks	673
12.7.3	Exceptions and Destructors: Revisiting <code>noexcept(false)</code>	675
12.8	Processing <code>new</code> Failures	676
12.8.1	<code>new</code> Throwing <code>bad_alloc</code> on Failure	677
12.8.2	<code>new</code> Returning <code>nullptr</code> on Failure	678
12.8.3	Handling <code>new</code> Failures Using Function <code>set_new_handler</code>	679
12.9	Standard Library Exception Hierarchy	680
12.10	C++'s Alternative to the <code>finally</code> Block: Resource Acquisition Is Initialization (RAII)	683
12.11	Some Libraries Support Both Exceptions and Error Codes	683
12.12	Logging	685
12.13	Looking Ahead to Contracts	685
12.14	Wrap-Up	694

13 Data Structures: Standard Library Containers and Iterators

697

13.1	Introduction	698
13.2	A Brief Intro to Big O	700
13.3	A Brief Intro to Hash Tables	703
13.4	Introduction to Containers	704
13.4.1	Common Nested Types in Sequence and Associative Containers	706
13.4.2	Common Container Member and Non-Member Functions	707
13.4.3	Requirements for Container Elements	710
13.5	Working with Iterators	710
13.5.1	Using <code>istream_iterator</code> for Input and <code>ostream_iterator</code> for Output	710
13.5.2	Iterator Categories	712
13.5.3	Container Support for Iterators	712
13.5.4	Predefined Iterator Type Names	713
13.5.5	Iterator Operators	713
13.6	A Brief Introduction to Algorithms	714
13.7	Sequence Containers	715
13.8	<code>vector</code> Sequence Container	715
13.8.1	Using <code>vectors</code> and Iterators	716
13.8.2	<code>vector</code> Element-Manipulation Functions	719
13.9	<code>list</code> Sequence Container	723
13.10	<code>deque</code> Sequence Container	728

13.11	Associative Containers	730
13.11.1	multiset Associative Container	730
13.11.2	set Associative Container	734
13.11.3	multimap Associative Container	736
13.11.4	map Associative Container	738
13.12	Container Adaptors	739
13.12.1	stack Adaptor	740
13.12.2	queue Adaptor	742
13.12.3	priority_queue Adaptor	743
13.13	bitset Near Container	744
13.14	Wrap-Up	746

14 Standard Library Algorithms and C++20 Ranges & Views **773**

14.1	Introduction	774
14.2	Algorithm Requirements: C++20 Concepts	776
14.3	Lambdas and Algorithms	778
14.4	Algorithms	781
14.4.1	fill, fill_n, generate and generate_n	781
14.4.2	equal, mismatch and lexicographical_compare	783
14.4.3	remove, remove_if, remove_copy and remove_copy_if	786
14.4.4	replace, replace_if, replace_copy and replace_copy_if	790
14.4.5	Shuffling, Counting, and Minimum and Maximum Element Algorithms	792
14.4.6	Searching and Sorting Algorithms	796
14.4.7	swap, iter_swap and swap_ranges	800
14.4.8	copy_backward, merge, unique, reverse, copy_if and copy_n	802
14.4.9	inplace_merge, unique_copy and reverse_copy	805
14.4.10	Set Operations	807
14.4.11	lower_bound, upper_bound and equal_range	810
14.4.12	min, max and minmax	812
14.4.13	Algorithms gcd, lcm, iota, reduce and partial_sum from Header <numeric>	813
14.4.14	Heapsort and Priority Queues	816
14.5	Function Objects (Functors)	821
14.6	Projections	825
14.7	C++20 Views and Functional-Style Programming	828
14.7.1	Range Adaptors	828
14.7.2	Working with Range Adaptors and Views	830
14.8	Intro to Parallel Algorithms	834
14.9	Standard Library Algorithm Summary	836
14.10	Future Ranges Enhancements	839
14.11	Wrap-Up	840

15	Templates, C++20 Concepts and Metaprogramming	845
15.1	Introduction	846
15.2	Custom Class Templates and Compile-Time Polymorphism	849
15.3	C++20 Function Template Enhancements	854
15.3.1	C++20 Abbreviated Function Templates	854
15.3.2	C++20 Templated Lambdas	856
15.4	C++20 Concepts: A First Look	856
15.4.1	Unconstrained Function Template <code>multiply</code>	857
15.4.2	Constrained Function Template with a C++20 Concepts <code>requires</code> Clause	860
15.4.3	C++20 Predefined Concepts	862
15.5	Type Traits	864
15.6	C++20 Concepts: A Deeper Look	868
15.6.1	Creating a Custom Concept	868
15.6.2	Using a Concept	869
15.6.3	Using Concepts in Abbreviated Function Templates	870
15.6.4	Concept-Based Overloading	871
15.6.5	<code>requires</code> Expressions	874
15.6.6	C++20 Exposition-Only Concepts	877
15.6.7	Techniques Before C++20 Concepts: SFINAE and Tag Dispatch	878
15.7	Testing C++20 Concepts with <code>static_assert</code>	879
15.8	Creating a Custom Algorithm	881
15.9	Creating a Custom Container and Iterators	883
15.9.1	Class Template <code>ConstIterator</code>	885
15.9.2	Class Template <code>Iterator</code>	888
15.9.3	Class Template <code>MyArray</code>	890
15.9.4	<code>MyArray</code> Deduction Guide for Braced Initialization	893
15.9.5	Using <code>MyArray</code> with <code>std::ranges</code> Algorithms	894
15.10	Default Arguments for Template Type Parameters	898
15.11	Variable Templates	898
15.12	Variadic Templates and Fold Expressions	899
15.12.1	<code>tuple</code> Variadic Class Template	899
15.12.2	Variadic Function Templates and an Intro to Fold Expressions	902
15.12.3	Types of Fold Expressions	906
15.12.4	How Unary Fold Expressions Apply Their Operators	906
15.12.5	How Binary-Fold Expressions Apply Their Operators	909
15.12.6	Using the Comma Operator to Repeatedly Perform an Operation	910
15.12.7	Constraining Parameter Pack Elements to the Same Type	911
15.13	Template Metaprogramming	913
15.13.1	C++ Templates Are Turing Complete	914
15.13.2	Computing Values at Compile-Time	914
15.13.3	Conditional Compilation with Template Metaprogramming and <code>constexpr if</code>	919
15.13.4	Type Metafunctions	921
15.14	Wrap-Up	925

16 C++20 Modules: Large-Scale Development 933

16.1	Introduction	934
16.2	Compilation and Linking Before C++20	936
16.3	Advantages and Goals of Modules	937
16.4	Example: Transitioning to Modules—Header Units	938
16.5	Modules Can Reduce Translation Unit Sizes and Compilation Times	941
16.6	Example: Creating and Using a Module	942
16.6.1	<code>module</code> Declaration for a Module Interface Unit	943
16.6.2	Exporting a Declaration	945
16.6.3	Exporting a Group of Declarations	945
16.6.4	Exporting a <code>namespace</code>	945
16.6.5	Exporting a <code>namespace</code> Member	946
16.6.6	Importing a Module to Use Its Exported Declarations	946
16.6.7	Example: Attempting to Access Non-Exported Module Contents	948
16.7	Global Module Fragment	951
16.8	Separating Interface from Implementation	951
16.8.1	Example: Module Implementation Units	951
16.8.2	Example: Modularizing a Class	954
16.8.3	<code>:private</code> Module Fragment	958
16.9	Partitions	958
16.9.1	Example: Module Interface Partition Units	959
16.9.2	Module Implementation Partition Units	962
16.9.3	Example: “Submodules” vs. Partitions	962
16.10	Additional Modules Examples	967
16.10.1	Example: Importing the C++ Standard Library as Modules	967
16.10.2	Example: Cyclic Dependencies Are Not Allowed	969
16.10.3	Example: <code>imports</code> Are Not Transitive	970
16.10.4	Example: Visibility vs. Reachability	971
16.11	Migrating Code to Modules	972
16.12	Future of Modules and Modules Tooling	973
16.13	Wrap-Up	975

17 Parallel Algorithms and Concurrency: A High-Level View 987

17.1	Introduction	988
17.2	Standard Library Parallel Algorithms	991
17.2.1	Example: Profiling Sequential and Parallel Sorting Algorithms	991
17.2.2	When to Use Parallel Algorithms	994
17.2.3	Execution Policies	995
17.2.4	Example: Profiling Parallel and Vectorized Operations	996
17.2.5	Additional Parallel Algorithm Notes	998
17.3	Multithreaded Programming	999
17.3.1	Thread States and the Thread Life Cycle	999
17.3.2	Deadlock and Indefinite Postponement	1001

17.4	Launching Tasks with <code>std::jthread</code>	1003
17.4.1	Defining a Task to Perform in a Thread	1003
17.4.2	Executing a Task in a <code>jthread</code>	1005
17.4.3	How <code>jthread</code> Fixes <code>thread</code>	1007
17.5	Producer–Consumer Relationship: A First Attempt	1008
17.6	Producer–Consumer: Synchronizing Access to Shared Mutable Data	1015
17.6.1	Class <code>SynchronizedBuffer</code> : Mutexes, Locks and Condition Variables	1017
17.6.2	Testing <code>SynchronizedBuffer</code>	1023
17.7	Producer–Consumer: Minimizing Waits with a Circular Buffer	1027
17.8	Readers and Writers	1036
17.9	Cooperatively Canceling <code>jthreads</code>	1037
17.10	Launching Tasks with <code>std::async</code>	1040
17.11	Thread-Safe, One-Time Initialization	1047
17.12	A Brief Introduction to Atomics	1048
17.13	Coordinating Threads with C++20 Latches and Barriers	1052
17.13.1	C++20 <code>std::latch</code>	1052
17.13.2	C++20 <code>std::barrier</code>	1055
17.14	C++20 Semaphores	1058
17.15	C++23: A Look to the Future of C++ Concurrency	1062
17.15.1	Parallel Ranges Algorithms	1062
17.15.2	Concurrent Containers	1062
17.15.3	Other Concurrency-Related Proposals	1063
17.16	Wrap-Up	1063

18 C++20 Coroutines **1073**

18.1	Introduction	1074
18.2	Coroutine Support Libraries	1075
18.3	Installing the <code>conurrencpp</code> and <code>generator</code> Libraries	1077
18.4	Creating a Generator Coroutine with <code>co_yield</code> and the <code>generator</code> Library	1077
18.5	Launching Tasks with <code>conurrencpp</code>	1081
18.6	Creating a Coroutine with <code>co_await</code> and <code>co_return</code>	1085
18.7	Low-Level Coroutines Concepts	1093
18.8	Future Coroutines Enhancements	1096
18.9	Wrap-Up	1096

19 Stream I/O & C++20 Text Formatting **1101**

19.1	Introduction	1102
19.2	Streams	1102
19.2.1	Classic Streams vs. Standard Streams	1103
19.2.2	<code>iostream</code> Library Headers	1103
19.2.3	Stream Input/Output Classes and Objects	1103

19.3	Stream Output	1104
19.3.1	Output of <code>char*</code> Variables	1105
19.3.2	Character Output Using Member Function <code>put</code>	1106
19.4	Stream Input	1106
19.4.1	<code>get</code> and <code>getline</code> Member Functions	1106
19.4.2	<code>istream</code> Member Functions <code>peek</code> , <code>putback</code> and <code>ignore</code>	1109
19.5	Unformatted I/O Using <code>read</code> , <code>write</code> and <code>gcount</code>	1110
19.6	Stream Manipulators	1111
19.6.1	Integral Stream Base: <code>dec</code> , <code>oct</code> , <code>hex</code> and <code>setbase</code>	1112
19.6.2	Floating-Point Precision (<code>setprecision</code> , <code>precision</code>)	1112
19.6.3	Field Width (<code>width</code> , <code>setw</code>)	1114
19.6.4	User-Defined Output Stream Manipulators	1115
19.6.5	Trailing Zeros and Decimal Points (<code>showpoint</code>)	1116
19.6.6	Alignment (<code>left</code> , <code>right</code> and <code>internal</code>)	1117
19.6.7	Padding (<code>fill</code> , <code>setfill</code>)	1118
19.6.8	Integral Stream Base (<code>dec</code> , <code>oct</code> , <code>hex</code> , <code>showbase</code>)	1119
19.6.9	Floating-Point Numbers; Scientific and Fixed Notation (<code>scientific</code> , <code>fixed</code>)	1120
19.6.10	Uppercase/Lowercase Control (<code>uppercase</code>)	1121
19.6.11	Specifying Boolean Format (<code>boolalpha</code>)	1122
19.6.12	Setting and Resetting the Format State via Member Function flags	1123
19.7	Stream Error States	1124
19.8	Tying an Output Stream to an Input Stream	1127
19.9	C++20 Text Formatting	1127
19.9.1	C++20 <code>std::format</code> Presentation Types	1128
19.9.2	C++20 <code>std::format</code> Field Widths and Alignment	1130
19.9.3	C++20 <code>std::format</code> Numeric Formatting	1131
19.9.4	C++20 <code>std::format</code> Field Width and Precision Placeholders	1132
19.10	Wrap-Up	1133

20 Other Topics and a Look Toward the Future of C++

1141

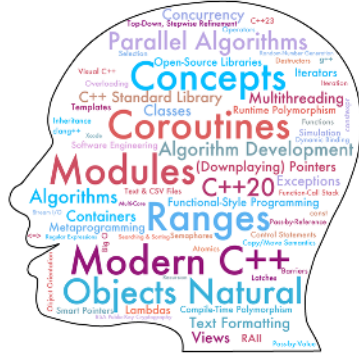
20.1	Introduction	1142
20.2	<code>shared_ptr</code> and <code>weak_ptr</code> Smart Pointers	1143
20.2.1	Reference Counted <code>shared_ptr</code>	1143
20.2.2	<code>weak_ptr</code> : <code>shared_ptr</code> Observer	1147
20.3	Runtime Polymorphism with <code>std::variant</code> and <code>std::visit</code>	1154
20.4	protected Class Members: A Deeper Look	1160
20.5	Non-Virtual Interface (NVI) Idiom	1161
20.6	Inheriting Base-Class Constructors	1168
20.7	Multiple Inheritance	1169
20.7.1	Diamond Inheritance	1174
20.7.2	Eliminating Duplicate Subobjects with <code>virtual</code> Base-Class Inheritance	1176

20.8	<code>public</code> , <code>protected</code> and <code>private</code> Inheritance	1177
20.9	namespaces: A Deeper Look	1179
20.9.1	Defining namespaces	1180
20.9.2	Accessing namespace Members with Qualified Names	1181
20.9.3	<code>using</code> Directives Should Not Be Placed in Headers	1181
20.9.4	Nested Namespaces	1181
20.9.5	Aliases for namespace Names	1181
20.10	Storage Classes and Storage Duration	1182
20.10.1	Storage Duration	1182
20.10.2	Local Variables and Automatic Storage Duration	1182
20.10.3	Static Storage Duration	1183
20.10.4	<code>mutable</code> Class Members	1184
20.11	Operator Keywords	1185
20.12	<code>decltype</code> Operator	1186
20.13	Trailing Return Types for Functions	1187
20.14	<code>[[nodiscard]]</code> Attribute	1187
20.15	Some Key C++23 Features	1189
20.16	Wrap-Up	1194

21 Computer Science Thinking: Searching, Sorting and Big O 1197

21.1	Introduction	1198
21.2	Efficiency of Algorithms: Big O	1199
21.2.1	$O(1)$ Algorithms	1199
21.2.2	$O(n)$ Algorithms	1199
21.2.3	$O(n^2)$ Algorithms	1200
21.3	Linear Search	1201
21.3.1	Implementation	1201
21.3.2	Efficiency of Linear Search	1202
21.4	Binary Search	1203
21.4.1	Implementation	1203
21.4.2	Efficiency of Binary Search	1207
21.5	Insertion Sort	1208
21.5.1	Implementation	1209
21.5.2	Efficiency of Insertion Sort	1210
21.6	Selection Sort	1210
21.6.1	Implementation	1211
21.6.2	Efficiency of Selection Sort	1213
21.7	Merge Sort (A Recursive Implementation)	1213
21.7.1	Implementation	1214
21.7.2	Efficiency of Merge Sort	1219
21.7.3	Summarizing Various Algorithms' Big O Notations	1219
21.8	Wrap-Up	1221

Index 1225



Preface

I 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/cpphttp11>

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++ 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: Qjvazkplm 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. (Down)playing Pointers in Modern C++

Security & safe programming.
ON: C++20 spans

8. strings, string_views, Text Files, CSV Files and Regex

ON: Reading and Analyzing the Titanic Disaster Data (CSV)
ON: Intro to Regular Expressions

PART 3

Modern Object-Oriented Programming & Exceptions

9. Custom Classes

ON: Studying the Vigenere 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 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

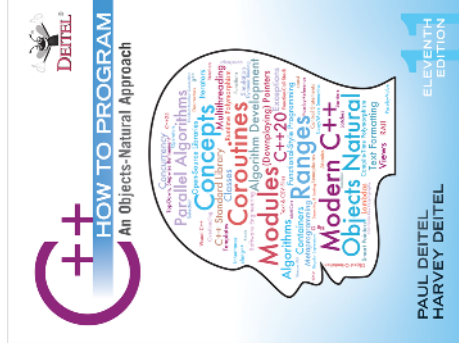
Miscellaneous Topics

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>.



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

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++ libraries⁴ 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.  SE
- **Security best practices** help you strengthen your programs against attacks.  Sec
- **Performance tips** highlight opportunities to make your programs run faster or minimize the amount of memory they occupy.  Perf
- **Common programming errors** help reduce the likelihood that you'll make the same mistakes.  Err
- **C++ Core Guidelines recommendations** (introduced in Section 12).  CG
- **C++20's new modules features**.  Mod
- **C++20's new concepts features**.  Concepts
- We present and use **data science topics** in several examples and exercises.  DS

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 I: 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 Qwbbbfmh 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 cipher⁷ 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 arrays and access their elements. We discuss passing arrays to functions, sorting and searching arrays and manipulating multidimensional arrays.

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 arrays, 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 arrays and vectors**. 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, (Down)playing 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, arrays and vectors, strings, C++20 spans and C++17 `string_views`. 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 spans**. These enable you to view and manipulate elements of contiguous containers, such as pointer-based arrays and standard library arrays and vectors, without using pointers directly.

Sec 

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 Grammarly⁹ 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



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 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 (templated 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 

Concepts 

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.

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**.  Mod

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.  Perf

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.  SE

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.  Perf

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.

SE 

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 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 `conurrencpp`**
- **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 languages¹⁶—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 58,000¹⁸ 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 Science²⁹

The undergraduate data science curriculum proposal³⁰ 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 Microsoft 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/cpphttp11>, 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.anualreviews.org/doi/full/10.1146/annurev-statistics-060116-053930>.
31. “Appendix—Detailed Courses for a Proposed Data Science Major,” http://www.anualreviews.org/doi/suppl/10.1146/annurev-statistics-060116-053930/suppl_file/st04_de_veaux_supmat.pdf.

- 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



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://iso-cpp.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 walk-through 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 recommendations³⁶ 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 recommendations³⁷ 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 ChatGPT³⁸ 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 files⁴⁰ 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/relocator>

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 on-site 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.vitalsource.com/>

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

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

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

<https://learning.oreilly.com/register/>

This page intentionally left blank



<https://deitel.com/cpphttp11>

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

- ## Obtaining the Code Examples

<https://github.com/pdeitel/CPlusPlusHowToProgram11e>

- 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.

<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 Xcode² 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/cpphttp11>

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>

iv Before You Begin

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 Prompt⁴ (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
```

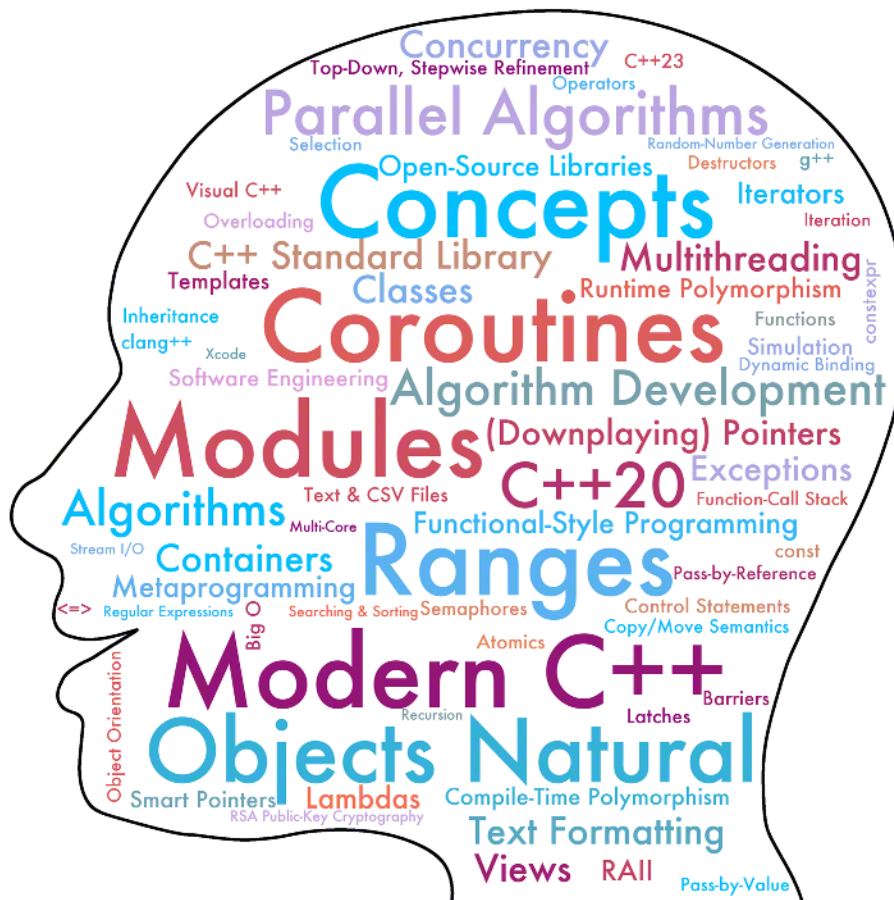
You can install `cppcheck` for various operating-system platforms by following the instructions at <https://cppcheck.sourceforge.io/>.

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



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
- 1.2 Hardware
 - 1.2.1 Computer Organization
 - 1.2.2 Moore's Law, Multi-Core Processors and Concurrent Programming
- 1.3 Data Hierarchies
- 1.4 Machine Languages, Assembly Languages and High-Level Languages
- 1.5 Operating Systems
- 1.6 C and C++
- 1.7 C++ Standard Library and Open-Source Libraries
- 1.8 Other Popular Programming Languages
- 1.9 Introduction to Object Orientation
- 1.10 Simplified View of a C++ Development Environment
- 1.11 Test-Driving a C++20 Application Various Ways
 - 1.11.1 Compiling and Running on Windows with Visual Studio Community Edition
 - 1.11.2 Compiling and Running with GNU C++ on Linux
 - 1.11.3 Compiling and Running with g++ in the GCC Docker Container
 - 1.11.4 Compiling and Running with clang++ in a Docker Container
 - 1.11.5 Compiling and Running with Xcode on macOS
- 1.12 Internet, World Wide Web, the Cloud and IoT
 - 1.12.1 The Internet: A Network of Networks
 - 1.12.2 The World Wide Web: Making the Internet User-Friendly
 - 1.12.3 The Cloud
 - 1.12.4 The Internet of Things (IoT)
 - 1.12.5 Edge Computing
 - 1.12.6 Mashups
- 1.13 Metaverse
 - 1.13.1 Virtual Reality (VR)
 - 1.13.2 Augmented Reality (AR)
 - 1.13.3 Mixed Reality
 - 1.13.4 Blockchain
 - 1.13.5 Bitcoin and Cryptocurrency
 - 1.13.6 Ethereum
 - 1.13.7 Non-Fungible Tokens (NFTs)
 - 1.13.8 Web3
- 1.14 Software Development Technologies
- 1.15 How Big Is Big Data?
 - 1.15.1 Big-Data Analytics
 - 1.15.2 Data Science and Big Data Are Making a Difference: Use Cases
- 1.16 AI—at the Intersection of Computer Science and Data Science
 - 1.16.1 Artificial Intelligence (AI)
 - 1.16.2 Artificial General Intelligence (AGI)
 - 1.16.3 Artificial Intelligence Milestones
 - 1.16.4 Machine Learning
 - 1.16.5 Deep Learning
 - 1.16.6 Reinforcement Learning
 - 1.16.7 Generative AI—ChatGPT and Dall-E 2
- 1.17 Wrap-Up Exercises

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://pypi.github.io/PYPL.html>.

C++ is widely used in industry.⁴ Portions of today’s most popular desktop operating systems—Windows⁵ and macOS⁶—are written in C++. Many applications and systems are partially written in C++, including popular web browsers (e.g., Google Chrome⁷ and Mozilla Firefox⁸), database management systems (e.g., Microsoft SQL Server⁹, Oracle¹⁰, MySQL¹¹), 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/cpphttp11>
- 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 “*binary digit*”) is either a 0 or a 1.

Arithmetic and Logic Unit (ALU)

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

Central Processing Unit (CPU)

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

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

Most computers today have **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 Athrow. “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 supercomputer¹⁵—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.

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

Perf 

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.

SE 

-
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 blockchain²⁹ 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_\(supercomputer\)](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/>.