starting out with >>>

C++

From Control Structures through Objects

TENTH EDITION

TONY GADDIS

STARTING OUT WITH

C++

From Control Structures through Objects

TENTH EDITION

This page intentionally left blank

STARTING OUT WITH

C++

From Control Structures through Objects

TENTH EDITION

Tony Gaddis
Haywood Community College

Pearson

Library of Congress Cataloging-in-Publication Data

Names: Gaddis, Tony, author.
Title: Starting out with C++ : from control structures through objects / Tony Gaddis, Haywood Community College.
Description: Tenth edition. | Hoboken, NJ : Pearson, [2021] | Includes index.
Identifiers: LCCN 2019059946 (print) | LCCN 2019059947 (ebook) | ISBN 9780135921043 (paperback) | ISBN 9780135928349 (epub)
Subjects: LCSH: C++ (Computer program language)
Classification: LCC QA76.73.C153 G334 2021 (print) | LCC QA76.73.C153 (ebook) | DDC 005.13/3--dc23
LC record available at https://lccn.loc.gov/2019059946
LC ebook record available at https://lccn.loc.gov/2019059947

ScoutAutomatedPrintCode

Print Offer
ISBN-13: 978-0-13-592829-5
ISBN-10: 0-13-592829-X

Contents at a Glance

Preface xvii

CHAPTER 1 Introduction to Computers and Programming 1

CHAPTER 2 Introduction to C++ 27

CHAPTER 3 Expressions and Interactivity 87

CHAPTER 4 Making Decisions 153

CHAPTER 5 Loops and Files 237

CHAPTER 6 Functions 309

CHAPTER 7 Arrays and Vectors 387

CHAPTER 8 Searching and Sorting Arrays 471

CHAPTER 9 Pointers 511

CHAPTER 10 Characters, C-Strings, and More about the `string` Class 565

CHAPTER 11 Structured Data 621

CHAPTER 12 Advanced File Operations 675

CHAPTER 13 Introduction to Classes 751

CHAPTER 14 More about Classes 849

CHAPTER 15 Inheritance, Polymorphism, and Virtual Functions 941

CHAPTER 16 Exceptions and Templates 1023

CHAPTER 17 The Standard Template Library 1065

CHAPTER 18 Linked Lists 1169

CHAPTER 19 Stacks and Queues 1213

CHAPTER 20 Recursion 1271

CHAPTER 21 Binary Trees 1309

Appendix A: The ASCII Character Set 1339

Appendix B: Operator Precedence and Associativity 1341

Quick References 1343

Index 1345

Credit 1363

Online The following appendices are available at www.pearsonhighered.com/cs-resources.

Appendix C: Introduction to Flowcharting
Appendix D: Using UML in Class Design
Appendix E: Namespaces
Appendix F: Passing Command Line Arguments
Appendix G: Binary Numbers and Bitwise Operations
Appendix H: STL Algorithms
Appendix I: Multi-Source File Programs
Appendix J: Stream Member Functions for Formatting
Appendix K: Unions
Appendix L: Answers to Checkpoints
Appendix M: Answers to Odd Numbered Review Questions

Case Study 1: C-String Manipulation
Case Study 2: High Adventure Travel Agency—Part 1
Case Study 3: Loan Amortization
Case Study 4: Creating a String Class
Case Study 5: High Adventure Travel Agency—Part 2
Case Study 6: High Adventure Travel Agency—Part 3
Case Study 7: Intersection of Sets
Case Study 8: Sales Commission

Contents

Preface xvii

CHAPTER 1 Introduction to Computers and Programming 1

1.1 Why Program? 1
1.2 Computer Systems: Hardware and Software 2
1.3 Programs and Programming Languages 8
1.4 What Is a Program Made of? 14
1.5 Input, Processing, and Output 17
1.6 The Programming Process 18
1.7 Procedural and Object-Oriented Programming 22
Review Questions and Exercises 24

CHAPTER 2 Introduction to C++ 27

2.1 The Parts of a C++ Program 27
2.2 The `cout` Object 31
2.3 The `#include` Directive 37
2.4 Variables, Literals, and Assignment Statements 39
2.5 Identifiers 43
2.6 Integer Data Types 44
2.7 The `char` Data Type 50
2.8 The C++ `string` Class 54
2.9 Floating-Point Data Types 56
2.10 The `bool` Data Type 59
2.11 Determining the Size of a Data Type 60
2.12 More about Variable Assignments and Initialization 61
2.13 Scope 64
2.14 Arithmetic Operators 64
2.15 Comments 72
2.16 Named Constants 74
2.17 Programming Style 76
Review Questions and Exercises 78
Programming Challenges 83

CHAPTER 3 Expressions and Interactivity 87

3.1 The `cin` Object 87
3.2 Mathematical Expressions 93
3.3 When You Mix Apples and Oranges: Type Conversion 102
3.4 Overflow and Underflow 104
3.5 Type Casting 105
3.6 Multiple Assignment and Combined Assignment 108
3.7 Formatting Output 112
3.8 Working with Characters and `string` Objects 122
3.9 More Mathematical Library Functions 128
3.10 Focus on Debugging: Hand Tracing a Program 133
3.11 Focus on Problem Solving: A Case Study 135
Review Questions and Exercises 140
Programming Challenges 146

CHAPTER 4 Making Decisions 153

4.1 Relational Operators 153
4.2 The `if` Statement 158
4.3 Expanding the `if` Statement 166
4.4 The `if/else` Statement 170
4.5 Nested `if` Statements 173
4.6 The `if/else if` Statement 180
4.7 The `if` Statement with Initialization 185
4.8 Flags 187
4.9 Logical Operators 188
4.10 Checking Numeric Ranges with Logical Operators 195
4.11 Menus 196
4.12 Focus on Software Engineering: Validating User Input 199
4.13 Comparing Characters and Strings 201
4.14 The Conditional Operator 205
4.15 The `switch` Statement 208
4.16 The `switch` Statement with Initialization 217
4.17 More about Blocks and Variable Scope 218
Review Questions and Exercises 222
Programming Challenges 227

CHAPTER 5 Loops and Files 237

5.1 The Increment and Decrement Operators 237
5.2 Introduction to Loops: The `while` Loop 242
5.3 Using the `while` Loop for Input Validation 249
5.4 Counters 251
5.5 The `do-while` Loop 252
5.6 The `for` Loop 257
5.7 Keeping a Running Total 267
5.8 Sentinels 270
5.9 Focus on Software Engineering: Deciding Which Loop to Use 271
5.10 Nested Loops 272
5.11 Using Files for Data Storage 275
5.12 Optional Topics: Breaking and Continuing a Loop 294
Review Questions and Exercises 296
Programming Challenges 301

CHAPTER 6 Functions 309

6.1 Focus on Software Engineering: Modular Programming 309
6.2 Defining and Calling Functions 310
6.3 Function Prototypes 319
6.4 Sending Data into a Function 321
6.5 Passing Data by Value 326
6.6 Focus on Software Engineering: Using Functions in a Menu-Driven Program 328
6.7 The `return` Statement 332
6.8 Returning a Value from a Function 334
6.9 Returning a Boolean Value 342
6.10 Local and Global Variables 344
6.11 Static Local Variables 352
6.12 Default Arguments 355
6.13 Using Reference Variables as Parameters 358
6.14 Overloading Functions 365
6.15 The `exit()` Function 369
6.16 Stubs and Drivers 372
Review Questions and Exercises 374
Programming Challenges 377

CHAPTER 7 Arrays and Vectors 387

7.1 Arrays Hold Multiple Values 387
7.2 Accessing Array Elements 389
7.3 No Bounds Checking in C++ 401
7.4 The Range-Based `for` Loop 404
7.5 Processing Array Contents 408
7.6 Focus on Software Engineering: Using Parallel Arrays 417
7.7 Arrays as Function Arguments 420
7.8 Two-Dimensional Arrays 431
7.9 Arrays with Three or More Dimensions 438
7.10 Focus on Problem Solving and Program Design: A Case Study 440
7.11 Introduction to the STL `vector` 442
Review Questions and Exercises 456
Programming Challenges 461

CHAPTER 8 Searching and Sorting Arrays 471

8.1 Focus on Software Engineering: Introduction to Search Algorithms 471
8.2 Focus on Problem Solving and Program Design: A Case Study 477
8.3 Focus on Software Engineering: Introduction to Sorting Algorithms 484
8.4 Focus on Problem Solving and Program Design: A Case Study 494
8.5 Sorting and Searching `vectors` (Continued from Section 7.11) 503
Review Questions and Exercises 506
Programming Challenges 507

CHAPTER 9 Pointers 511

9.1 Getting the Address of a Variable 511
9.2 Pointer Variables 513
9.3 The Relationship between Arrays and Pointers 520
9.4 Pointer Arithmetic 525

9.5 Initializing Pointers 527
9.6 Comparing Pointers 528
9.7 Pointers as Function Parameters 530
9.8 Dynamic Memory Allocation 540
9.9 Returning Pointers from Functions 544
9.10 Using Smart Pointers to Avoid Memory Leaks 550
9.11 Focus on Problem Solving and Program Design: A Case Study 553
Review Questions and Exercises 559
Programming Challenges 562

CHAPTER 10 Characters, C-Strings, and More about the `string` Class 565

10.1 Character Testing 565
10.2 Character Case Conversion 569
10.3 C-Strings 572
10.4 Library Functions for Working with C-Strings 576
10.5 String/Numeric Conversion Functions 587
10.6 Focus on Software Engineering: Writing Your Own C-String-Handling Functions 593
10.7 More about the C++ `string` Class 599
10.8 Focus on Problem Solving and Program Design: A Case Study 611
Review Questions and Exercises 613
Programming Challenges 616

CHAPTER 11 Structured Data 621

11.1 Abstract Data Types 621
11.2 Structures 623
11.3 Accessing Structure Members 626
11.4 Initializing a Structure 630
11.5 Arrays of Structures 633
11.6 Focus on Software Engineering: Nested Structures 635
11.7 Structures as Function Arguments 639
11.8 Returning a Structure from a Function 642
11.9 Using Structured Binding Declarations with Structures 645
11.10 Pointers to Structures 647
11.11 Focus on Software Engineering: When to Use ., When to Use `->`, and When to Use `*` 651
11.12 Enumerated Data Types 653
Review Questions and Exercises 664
Programming Challenges 670

CHAPTER 12 Advanced File Operations 675

12.1 File Operations 675
12.2 File Output Formatting 681
12.3 Passing File Stream Objects to Functions 683
12.4 More Detailed Error Testing 685
12.5 Member Functions for Reading and Writing Files 688
12.6 Focus on Software Engineering: Working with Multiple Files 696
12.7 Binary Files 698
12.8 Creating Records with Structures 703
12.9 Random-Access Files 707

12.10 Opening a File for Both Input and Output 715
12.11 Working with the File System 720
Review Questions and Exercises 741
Programming Challenges 745

CHAPTER 13 Introduction to Classes 751

13.1 Procedural and Object-Oriented Programming 751
13.2 Introduction to Classes 758
13.3 Defining an Instance of a Class 763
13.4 Why Have Private Members? 776
13.5 Focus on Software Engineering: Separating Class Specification from Implementation 777
13.6 Inline Member Functions 783
13.7 Constructors 786
13.8 Passing Arguments to Constructors 791
13.9 Destructors 799
13.10 Overloading Constructors 803
13.11 Private Member Functions 807
13.12 Arrays of Objects 809
13.13 Focus on Problem Solving and Program Design: An OOP Case Study 813
13.14 Focus on Object-Oriented Programming: Simulating Dice with Objects 820
13.15 Focus on Object-Oriented Design: The Unified Modeling Language (UML) 824
13.16 Focus on Object-Oriented Design: Finding the Classes and Their Responsibilities 826
Review Questions and Exercises 835
Programming Challenges 840

CHAPTER 14 More about Classes 849

14.1 Instance and Static Members 849
14.2 Friends of Classes 857
14.3 Memberwise Assignment 862
14.4 Copy Constructors 863
14.5 Operator Overloading 869
14.6 Object Conversion 896
14.7 Aggregation 898
14.8 Focus on Object-Oriented Design: Class Collaborations 903
14.9 Focus on Object-Oriented Programming: Simulating the Game of Cho-Han 908
14.10 Rvalue References and Move Semantics 918
Review Questions and Exercises 929
Programming Challenges 934

CHAPTER 15 Inheritance, Polymorphism, and Virtual Functions 941

15.1 What Is Inheritance? 941
15.2 Protected Members and Class Access 950
15.3 Constructors and Destructors in Base and Derived Classes 956
15.4 Redefining Base Class Functions 970
15.5 Class Hierarchies 975
15.6 Polymorphism and Virtual Member Functions 981
15.7 Abstract Base Classes and Pure Virtual Functions 997
15.8 Multiple Inheritance 1004

Review Questions and Exercises 1011
Programming Challenges 1015

CHAPTER 16 Exceptions and Templates 1023

16.1 Exceptions 1023
16.2 Function Templates 1043
16.3 Focus on Software Engineering: Where to Start When Defining Templates 1049
16.4 Class Templates 1050
Review Questions and Exercises 1059
Programming Challenges 1062

CHAPTER 17 The Standard Template Library 1065

17.1 Introduction to the Standard Template Library 1065
17.2 STL Container and Iterator Fundamentals 1065
17.3 The `vector` Class 1076
17.4 The `map`, `multimap`, and `unordered_map` Classes 1090
17.5 The `set`, `multiset`, and `unordered_set` Classes 1117
17.6 The `tuple` Class 1124
17.7 Algorithms 1131
17.8 Introduction to Function Objects and Lambda Expressions 1153
Review Questions and Exercises 1160
Programming Challenges 1165

CHAPTER 18 Linked Lists 1169

18.1 Introduction to the Linked List ADT 1169
18.2 Linked List Operations 1171
18.3 A Linked List Template 1190
18.4 Variations of the Linked List 1201
18.5 The STL `list` and `forward_list` Containers 1202
Review Questions and Exercises 1207
Programming Challenges 1209

CHAPTER 19 Stacks and Queues 1213

19.1 Introduction to the Stack ADT 1213
19.2 Dynamic Stacks 1231
19.3 The STL `stack` Container 1241
19.4 Introduction to the Queue ADT 1243
19.5 Dynamic Queues 1256
19.6 The STL `deque` and `queue` Containers 1263
Review Questions and Exercises 1266
Programming Challenges 1268

CHAPTER 20 Recursion 1271

20.1 Introduction to Recursion 1271
20.2 Solving Problems with Recursion 1275
20.3 Focus on Problem Solving and Program Design: The Recursive `gcd` Function 1283
20.4 Focus on Problem Solving and Program Design: Solving Recursively Defined Problems 1284

20.5 Focus on Problem Solving and Program Design: Recursive Linked List Operations 1285
20.6 Focus on Problem Solving and Program Design: A Recursive Binary Search Function 1289
20.7 The Towers of Hanoi 1291
20.8 Focus on Problem Solving and Program Design: The QuickSort Algorithm 1294
20.9 Exhaustive Algorithms 1298
20.10 Recursion and Variadic Function Templates 1301
20.11 Focus on Software Engineering: Recursion versus Iteration 1303
Review Questions and Exercises 1304
Programming Challenges 1305

CHAPTER 21 Binary Trees 1309

21.1 Definition and Applications of Binary Trees 1309
21.2 Binary Search Tree Operations 1312
21.3 Template Considerations for Binary Search Trees 1329
Review Questions and Exercises 1335
Programming Challenges 1336

Appendix A: The ASCII Character Set 1339
Appendix B: Operator Precedence and Associativity 1341
Quick References 1343
Index 1345
Credit 1363

Online The following appendices are available at www.pearsonhighered.com/cs-resources.

Appendix C: Introduction to Flowcharting
Appendix D: Using UML in Class Design
Appendix E: Namespaces
Appendix F: Passing Command Line Arguments
Appendix G: Binary Numbers and Bitwise Operations
Appendix H: STL Algorithms
Appendix I: Multi-Source File Programs
Appendix J: Stream Member Functions for Formatting
Appendix K: Unions
Appendix L: Answers to Checkpoints
Appendix M: Answers to Odd Numbered Review Questions

Case Study 1: C-String Manipulation
Case Study 2: High Adventure Travel Agency—Part 1
Case Study 3: Loan Amortization
Case Study 4: Creating a String Class
Case Study 5: High Adventure Travel Agency—Part 2
Case Study 6: High Adventure Travel Agency—Part 3
Case Study 7: Intersection of Sets
Case Study 8: Sales Commission

LOCATION OF VIDEONOTES IN THE TEXT

Chapter 1	Introduction to Flowcharting, p. 20 Designing a Program with Pseudocode, p. 20 Designing the Account Balance Program, p. 25 Predicting the Result of Problem 33, p. 26
Chapter 2	Using `cout`, p. 32 Variable Definitions, p. 39 Assignment Statements and Simple Math Expressions, p. 64 Solving the Restaurant Bill Problem, p. 83
Chapter 3	Reading Input with `cin`, p. 87 Formatting Numbers with `setprecision`, p. 115 Solving the Stadium Seating Problem, p. 146
Chapter 4	The `if` Statement, p. 158 The `if/else` Statement, p. 170 The `if/else if` Statement, p. 180 Solving the Time Calculator Problem, p. 228
Chapter 5	The `while` Loop, p. 242 The `for` Loop, p. 257 Reading Data from a File, p. 284 Solving the Calories Burned Problem, p. 301
Chapter 6	Functions and Arguments, p. 321 Value-Returning Functions, p. 334 Solving the Markup Problem, p. 377
Chapter 7	Accessing Array Elements with a Loop, p. 392 Passing an Array to a Function, p. 420 Solving the Chips and Salsa Problem, p. 462
Chapter 8	The Binary Search, p. 474 The Selection Sort, p. 490 Solving the Charge Account Validation Modification Problem, p. 508
Chapter 9	Dynamically Allocating an Array, p. 541 Solving the Pointer Rewrite Problem, p. 563
Chapter 10	Writing a C-String-Handling Function, p. 593 More about the `string` Class, p. 599 Solving the Backward String Problem, p. 616

LOCATION OF VIDEONOTES IN THE TEXT *(continued)*

Chapter 11	Creating a Structure, p. 623 Passing a Structure to a Function, p. 639 Solving the Weather Statistics Problem, p. 670
Chapter 12	Passing File Stream Objects to Functions, p. 683 Working with Multiple Files, p. 696 Solving the File Encryption Filter Problem, p. 747
Chapter 13	Writing a Class, p. 758 Defining an Instance of a Class, p. 763 Solving the `Employee` Class Problem, p. 840
Chapter 14	Operator Overloading, p. 869 Class Aggregation, p. 898 Solving the `NumDays` Problem, p. 935
Chapter 15	Redefining a Base Class Function in a Derived Class, p. 970 Polymorphism, p. 981 Solving the `Employee` and `ProductionWorker` Classes Problem, p. 1015
Chapter 16	Throwing an Exception, p. 1024 Handling an Exception, p. 1024 Writing a Function Template, p. 1043 Solving the Exception Project Problem, p. 1063
Chapter 17	The `array` Container, p. 1068 Iterators, p. 1070 The `vector` Container, p. 1076 The `map` Container, p. 1090 The `set` Container, p. 1117 Function Objects and Lambda Expressions, p. 1153 The Course Information Problem, p. 1165
Chapter 18	Appending a Node to a Linked List, p. 1172 Inserting a Node in a Linked List, p. 1179 Deleting a Node from a Linked List, p. 1185 Solving the Member Insertion by Position Problem, p. 1210
Chapter 19	Storing Objects in an STL `stack`, p. 1241 Storing Objects in an STL `queue`, p. 1265 Solving the File Compare Problem, p. 1270
Chapter 20	Reducing a Problem with Recursion, p. 1276 Solving the Recursive Multiplication Problem, p. 1306
Chapter 21	Inserting a Node in a Binary Tree, p. 1314 Deleting a Node from a Binary Tree, p. 1320 Solving the Node Counter Problem, p. 1336

This page intentionally left blank

Preface

Welcome to *Starting Out with C++: From Control Structures through Objects, 10th edition*. This book is intended for use in a two-semester C++ programming sequence, or an accelerated one-semester course. Students new to programming, as well as those with prior course work in other languages, will find this text beneficial. The fundamentals of programming are covered for the novice, while the details, pitfalls, and nuances of the C++ language are explored in depth for both the beginner and more experienced student. The book is written with clear, easy-to-understand language, and it covers all the necessary topics for an introductory programming course. This text is rich in example programs that are concise, practical, and real-world oriented, ensuring that the student not only learns how to implement the features and constructs of C++, but why and when to use them.

Revel

If you are using this textbook along with **Revel** for Gaddis *Starting Out with C++, 10e,* please understand that there may be some pedagogical differences between the two. **Revel**, Pearson's fully immersive, all-in-one digital learning environment, is design for interactive online learning. Therefore, some of the learning aids in the textbook had to be removed or re-imagined in order to create a better online learning experience for students.

Changes in the Tenth Edition

This book's pedagogy, organization, and clear writing style remain the same as in the previous edition. Many improvements and updates have been made, which are summarized here:

- **New material on the `if` statement and the `switch` statement with Initialization**

  C++ 17 introduced new forms of the `if` statement and the `switch` statement that include an initialization clause. In this edition, Chapter 4 includes new material on this syntax and shows examples using both.

- **New Random Number Generator**

  Modern C++ provides a new and improved random number generator with an intuitive syntax for getting a random number within a specified range. This edition replaces the previous C-style technique for random number generation with the new, modern C++ approach.

- **Tuples**

  Chapter 17, which covers the Standard Template Library, provides a new section on the `tuple` library. Tuples are explained and numerous examples of using tuples to store and retrieve data are given.

- **New Forms of String and Numeric Literals**

  This edition introduces raw string literals, binary literals, and the use of digit separators in numeric literals.

- **The `filesystem` Library**

  Chapter 12 includes a new section on the `filesystem` library, which was introduced in C++ 17. The `filesystem` library allows you to work with files and directories at the operating system level, performing operations such as copying and deleting files, getting a list of a directory's contents, and recursively traversing a directory tree.

- **Structured Binding Declarations**

  Structured binding declarations, which were introduced in C++ 17, provide a concise syntax for unpacking a collection or data structure and assigning its contents to individual variables. This edition shows how to use structured binding declarations to unpack arrays, structures, and tuples.

- **Defaulted and Deleted Operations**

  Chapter 14 shows how to use the `default` and `delete` key words to explicitly instruct the compiler to either generate or not generate a class's default constructor, default copy constructor, default move constructor, default copy assignment operator, and default destructor.

- **Usage of `typename` Instead of `class` In Templates**

  In the code for function and class templates, this edition uses the `typename` key word instead of the `class` key word for declaring type parameters.

- **The `noexcept` Key Word**

  Chapter 16 in this edition introduces the `noexcept` key word and discusses its use for declaring functions that do not throw an exception.

- **Enhanced Discussion of Deleting Nodes in a Linked List**

  Chapter 18's explanation of deleting a node in a linked list has been expanded with more detail, including a new figure that illustrates the process of unlinking a node, and pseudocode describing the process for deleting a node in either a sorted or an unsorted linked list.

- **Variadic Function Templates**

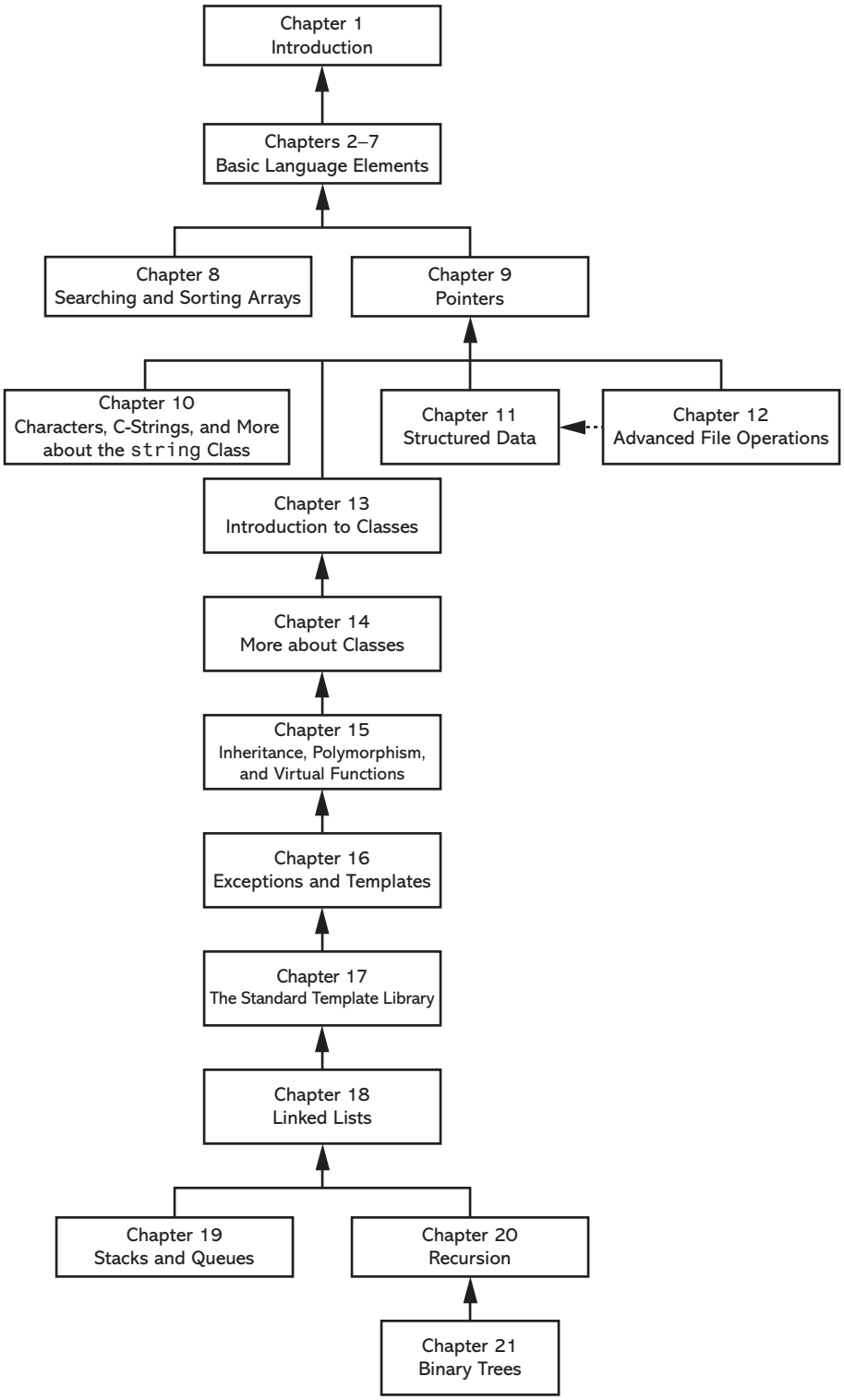
  Chapter 20 presents a new section on variadic function templates, which allow you to write a set of function templates that use recursion to process a variable number of arguments.

Organization of the Text

This text teaches C++ in a step-by-step fashion. Each chapter covers a major set of topics and builds knowledge as the student progresses through the book. Although the chapters can be easily taught in their existing sequence, some flexibility is provided. The diagram shown in Figure P-1 suggests possible sequences of instruction.

Chapter 1 covers fundamental hardware, software, and programming concepts. You may choose to skip this chapter if the class is already familiar with those topics. Chapters 2 through 7 cover basic C++ syntax, data types, expressions, selection structures, repetition structures, functions, and arrays. Each of these chapters builds on the previous chapter and should be covered in the order presented.

Figure P-1 Chapter dependency chart



After Chapter 7 has been covered, you may proceed to Chapter 8, or jump to Chapter 9.

After Chapter 9 has been covered, Chapter 10, 11, 12 or 13 may be covered. (If you jump to Chapter 12 at this point, you will need to postpone Sections 12.8, 12.9, and 12.10 until Chapter 11 has been covered.) After Chapter 13, you may cover Chapters 14 through 18 in sequence. Next, you can proceed to either Chapter 19 or Chapter 20. Finally, Chapter 21 may be covered.

This text's approach starts with a firm foundation in structured, procedural programming before delving fully into object-oriented programming and advanced data structures.

Brief Overview of Each Chapter

Chapter 1: Introduction to Computers and Programming

This chapter provides an introduction to the field of computer science and covers the fundamentals of programming, problem solving, and software design. The components of programs, such as key words, variables, operators, and punctuation, are covered. The tools of the trade, such as pseudocode, flow charts, and hierarchy charts, are also presented.

Chapter 2: Introduction to C++

This chapter gets the student started in C++ by introducing data types, identifiers, variable declarations, constants, comments, program output, simple arithmetic operations, and C-strings. Programming style conventions are introduced and good programming style is modeled here, as it is throughout the text.

Chapter 3: Expressions and Interactivity

In this chapter, the student learns to write programs that input and handle numeric, character, and string data. The use of arithmetic operators and the creation of mathematical expressions are covered in greater detail, with emphasis on operator precedence. Debugging is introduced, with a section on hand tracing a program. Sections are also included on simple output formatting, on data type conversion and type casting, and on using library functions that work with numbers.

Chapter 4: Making Decisions

Here, the student learns about relational operators, relational expressions, and how to control the flow of a program with the `if`, `if/else`, and `if/else if` statements. The conditional operator and the `switch` statement are also covered. Crucial applications of these constructs are covered, such as menu-driven programs and the validation of input.

Chapter 5: Loops and Files

This chapter covers repetition control structures. The `while` loop, `do-while` loop, and `for` loop are taught, along with common uses for these devices. Counters, accumulators, running totals, sentinels, and other application-related topics are discussed. Sequential file I/O is also introduced. The student learns to read and write text files, and use loops to process the data in a file.

Chapter 6: Functions

In this chapter, the student learns how and why to modularize programs, using both `void` and value returning functions. Argument passing is covered, with emphasis on when arguments should be passed by value versus when they need to be passed by reference. Scope of

variables is covered, and sections are provided on local versus global variables and on static local variables. Overloaded functions are also introduced and demonstrated.

Chapter 7: Arrays and Vectors

In this chapter, the student learns to create and work with single and multi-dimensional arrays. Many examples of array processing are provided including examples illustrating how to find the sum, average, highest, and lowest values in an array, and how to sum the rows, columns, and all elements of a two-dimensional array. Programming techniques using parallel arrays are also demonstrated, and the student is shown how to use a data file as an input source to populate an array. STL `vectors` are introduced and compared to arrays.

Chapter 8: Searching and Sorting Arrays

Here, the student learns the basics of sorting arrays and searching for data stored in them. The chapter covers the Bubble Sort, Selection Sort, Linear Search, and Binary Search algorithms. There is also a section on sorting and searching STL `vector` objects.

Chapter 9: Pointers

This chapter explains how to use pointers. Pointers are compared to and contrasted with reference variables. Other topics include pointer arithmetic, initialization of pointers, relational comparison of pointers, pointers and arrays, pointers and functions, dynamic memory allocation, and more.

Chapter 10: Characters, C-Strings, and More about the `string` Class

This chapter discusses various ways to process text at a detailed level. Library functions for testing and manipulating characters are introduced. C-strings are discussed, and the technique of storing C-strings in `char` arrays is covered. An extensive discussion of the `string` class methods is also given.

Chapter 11: Structured Data

The student is introduced to abstract data types and taught how to create them using structures, unions, and enumerated data types. Discussions and examples include using pointers to structures, passing structures to functions, and returning structures from functions.

Chapter 12: Advanced File Operations

This chapter covers sequential access, random access, text, and binary files. The various modes for opening files are discussed, as well as the many methods for reading and writing file contents. Advanced output formatting is also covered. The chapter includes a discussion of operating system paths and introduces the standard `filesystem` library for accessing files and directories at the operating system level.

Chapter 13: Introduction to Classes

The student now shifts focus to the object-oriented paradigm. This chapter covers the fundamental concepts of classes. Member variables and functions are discussed. The student learns about private and public access specifications, and reasons to use each. The topics of constructors, overloaded constructors, and destructors are also presented. The chapter presents a section modeling classes with UML, and how to find the classes in a particular problem.

Chapter 14: More about Classes

This chapter continues the study of classes. Static members, friends, memberwise assignment, and copy constructors are discussed. The chapter also includes in-depth sections on operator overloading, object conversion, and object aggregation. There is also a section on class collaborations and the use of CRC cards.

Chapter 15: Inheritance, Polymorphism, and Virtual Functions

The study of classes continues in this chapter with the subjects of inheritance, polymorphism, and virtual member functions. The topics covered include base and derived class constructors and destructors, virtual member functions, base class pointers, static and dynamic binding, multiple inheritance, and class hierarchies.

Chapter 16: Exceptions and Templates

The student learns to develop enhanced error trapping techniques using exceptions. Discussion then turns to function and class templates as a method for reusing code.

Chapter 17: The Standard Template Library

This chapter discusses the containers, iterators, and algorithms in the Standard Template Library (STL). The specific containers covered are the `array`, `vector`, `map`, `multimap`, `unordered_map`, `set`, `multiset`, `unordered_set`, and `tuple` classes. The student then learns about sorting, searching, permutation, and set algorithms. The chapter concludes with a discussion of function objects (functors) and lambda functions.

Chapter 18: Linked Lists

This chapter introduces concepts and techniques needed to work with lists. A linked list ADT is developed and the student is taught to code operations such as creating a linked list, appending a node, traversing the list, searching for a node, inserting a node, deleting a node, and destroying a list. A linked list class template is also demonstrated.

Chapter 19: Stacks and Queues

In this chapter, the student learns to create and use static and dynamic stacks and queues. The operations of stacks and queues are defined, and templates for each ADT are demonstrated.

Chapter 20: Recursion

This chapter discusses recursion and its use in problem solving. A visual trace of recursive calls is provided, and recursive applications are discussed. Many recursive algorithms are presented, including recursive functions for finding factorials, finding a greatest common denominator (GCD), performing a binary search, and sorting (QuickSort). The classic Towers of Hanoi example is also presented. For students who need more challenge, there is a section on exhaustive algorithms. The chapter concludes with a discussion of variadic function templates, which use recursion to process a variable number of arguments.

Chapter 21: Binary Trees

This chapter covers the binary tree ADT and demonstrates many binary tree operations. The student learns to traverse a tree, insert an element, delete an element, replace an element, test for an element, and destroy a tree.

Appendix A: The ASCII Character Set

A list of the ASCII and Extended ASCII characters and their codes.

Appendix B: Operator Precedence and Associativity

A chart showing the C++ operators and their precedence.

The following appendices are available online at www.pearsonhighered.com/cs-resources.

Appendix C: Introduction to Flowcharting

A brief introduction to flowcharting. This tutorial discusses sequence, decision, case, repetition, and module structures.

Appendix D: Using UML in Class Design

This appendix shows the student how to use the Unified Modeling Language to design classes. Notation for showing access specification, data types, parameters, return values, overloaded functions, composition, and inheritance are included.

Appendix E: Namespaces

This appendix explains namespaces and their purpose. Examples showing how to define a namespace and access its members are given.

Appendix F: Passing Command Line Arguments

Teaches the student how to write a C++ program that accepts arguments from the command line. This appendix will be useful to students working in a command line environment, such as Unix, Linux, or the Windows command prompt.

Appendix G: Binary Numbers and Bitwise Operations

A guide to the C++ bitwise operators, as well as a tutorial on the internal storage of integers

Appendix H: STL Algorithms

This appendix gives a summary of each of the function templates provided by the Standard Template Library (STL), and defined in the `<algorithm>` header file.

Appendix I: Multi-Source File Programs

Provides a tutorial on creating programs that consist of multiple source files. Function header files, class specification files, and class implementation files are discussed.

Appendix J: Stream Member Functions for Formatting

Covers stream member functions for formatting such as `setf`

Appendix K: Unions

This appendix introduces unions. It describes the purpose of unions and the difference between a union and a `struct`, demonstrates how to declare a union and define a union variable, and shows example programs that use unions.

Appendix L: Answers to Checkpoints

Students may test their own progress by comparing their answers to the Checkpoint exercises against this appendix. The answers to all Checkpoints are included.

Appendix M: Answers to Odd Numbered Review Questions

Another tool that students can use to gauge their progress.

Features of the Text

Concept Statements Each major section of the text starts with a concept statement. This statement summarizes the ideas of the section.

Example Programs The text has hundreds of complete example programs, each designed to highlight the topic currently being studied. In most cases, these are practical, real-world examples. Source code for these programs is provided so that students can run the programs themselves.

Program Output After each example program, there is a sample of its screen output. This immediately shows the student how the program should function.

In the Spotlight Each of these sections provides a programming problem and a detailed, step-by-step analysis showing the student how to solve it.

VideoNotes Videos that provide explanations of specific topics and show the student how to solve various programming problems are available for viewing at www.pearsonhighered.com/cs-resources. Icons appear throughout the text alerting the student to specific videos.

Checkpoints Checkpoints are questions placed throughout each chapter as a self-test study aid. These questions allow students to check how well they have learned a new topic. Answers for all Checkpoint questions can be downloaded from the book's companion Web site at www. pearsonhighered.com/cs-resources.

Notes Notes appear at appropriate places throughout the text. They are short explanations of interesting or often misunderstood points relevant to the topic at hand.

Warnings Warnings are notes that caution the student about certain C++ features, programming techniques, or practices that can lead to malfunctioning programs or lost data.

Case Studies Case studies that simulate real-world applications appear in many chapters throughout the text. These case studies are designed to highlight the major topics of the chapter in which they appear.

Review Questions and Exercises Each chapter presents a thorough and diverse set of review questions, such as fill-in-the-blank and short answer, that check the student's mastery of the basic material presented in the chapter. These are followed by exercises requiring problem solving and analysis, such as the *Algorithm Workbench, Predict the Output,* and *Find the Errors sections*. Answers to the odd-numbered review questions and review exercises can be downloaded from the book's companion Web site at www.pearsonhighered.com/cs-resources.

Programming Challenges Each chapter offers a pool of programming exercises designed to solidify the student's knowledge of the topics currently being studied. In most cases, the assignments present real-world problems to be solved. When applicable, these exercises include input validation rules.

Group Projects There are several group programming projects throughout the text, intended to be constructed by a team of students. One student might build the program's user interface, while another student writes the mathematical code, and another designs and implements a class the program uses. This process is similar to the way many professional programs are written and encourages team work within the classroom.

Modern C++ Throughout the text, new Modern C++ language features are introduced.

Supplements

Student Online Resources

Many student resources are available for this book from the publisher. The following items are available on the Gaddis Series Companion Web site at www.pearsonhighered.com/cs-resources:

- The source code for each example program in the book
- Access to the book's VideoNotes
- A full set of appendices, including answers to the Checkpoint questions and answers to the odd-numbered review questions
- A collection of valuable Case Studies

Instructor Resources

The following supplements are available to qualified instructors only:

- Answers to all Review Questions in the text
- Solutions for all Programming Challenges in the text
- PowerPoint presentation slides for every chapter

- Computerized test bank
- Answers to all Student Lab Manual questions
- Solutions for all Student Lab Manual programs

Visit the Gaddis Series Companion Web site at www.pearsonhighered.com/cs-resources to access the Instructor Resources. If you do not already have access to the Pearson IRC, please contact your Pearson representative at Pearson.com/RepLocator.com

Which Gaddis C++ book is right for you?

The Starting Out with C++ Series includes three books, one of which is sure to fit your course:

- *Starting Out with C++: From Control Structures through Objects*
- *Starting Out with C++: Early Objects*
- *Starting Out with C++: Brief Version*

The following chart will help you determine which book is right for your course.

■ FROM CONTROL STRUCTURES THROUGH OBJECTS ■ BRIEF VERSION	■ EARLY OBJECTS
LATE INTRODUCTION OF OBJECTS Classes are introduced in Chapter 13 of the standard text and the brief text, after control structures, functions, arrays, and pointers. Advanced OOP topics, such as inheritance and polymorphism, are covered in the following two chapters.	**EARLIER INTRODUCTION OF OBJECTS** Classes are introduced in Chapter 7, after control structures and functions, but before arrays and pointers. Their use is then integrated into the remainder of the text. Advanced OOP topics, such as inheritance and polymorphism, are covered in Chapters 11 and 15.
INTRODUCTION OF DATA STRUCTURES AND RECURSION Linked lists, stacks and queues, and binary trees are introduced in the final chapters of the standard text. Recursion is covered after stacks and queues, but before binary trees. These topics are not covered in the brief text, though it does have appendices dealing with linked lists and recursion.	**INTRODUCTION OF DATA STRUCTURES AND RECURSION** Linked lists, stacks and queues, and binary trees are introduced in the final chapters of the text, after the chapter on recursion.

Acknowledgments

There have been many helping hands in the development and publication of this text. We would like to thank the following faculty reviewers for their helpful suggestions and expertise.

Ahmad Abuhejleh
University of Wisconsin–River Falls

David Akins
El Camino College

Steve Allan
Utah State University

Vicki Allan
Utah State University

Karen M. Arlien
Bismark State College

Mary Astone
Troy University

Ijaz A. Awan
Savannah State University

Robert Baird
Salt Lake Community College

Don Biggerstaff
Fayetteville Technical Community College

Michael Bolton
Northeastern Oklahoma State University

Bill Brown
Pikes Peak Community College

Robert Burn
Diablo Valley College

Charles Cadenhead
Richland Community College

Randall Campbell
Morningside College

Wayne Caruolo
Red Rocks Community College

Cathi Chambley-Miller
Aiken Technical College

Chia-Chin Chang
Lakeland College

C.C. Chao
Jacksonville State University

Joseph Chao
Bowling Green State University

Royce Curtis
Western Wisconsin Technical College

Joseph DeLibero
Arizona State University

Michael Dixon
Sacramento City College

Jeanne Douglas
University of Vermont

Michael Dowell
Augusta State University

Qiang Duan
Penn State University—Abington

William E. Duncan
Louisiana State University

Daniel Edwards
Ohlone College

Judy Etchison
Southern Methodist University

Dennis Fairclough
Utah Valley State College

Xisheng Fang
Ohlone College

Mark Fienup
University of Northern Iowa

Richard Flint
North Central College

Ann Ford Tyson
Florida State University

Jeanette Gibbons
South Dakota State University

James Gifford
University of Wisconsin–Stevens Point

Leon Gleiberman
Touro College

Barbara Guillott
Louisiana State University

Pranshu Gupta
DeSales University

Ranette Halverson, Ph.D.
Midwestern State University

Ken Hang
Green River Community College

Carol Hannahs
University of Kentucky

Charles Hardnett
Gwinnett Technical College

Dennis Heckman
Portland Community College

Ric Heishman
George Mason University

Michael Hennessy
University of Oregon

Ilga Higbee
Black Hawk College

Patricia Hines
Brookdale Community College

Mike Holland
Northern Virginia Community College

Mary Hovik
Lehigh Carbon Community College

Richard Hull
Lenoir-Rhyne College

Kay Johnson
Community College of Rhode Island

Chris Kardaras
North Central College

Willard Keeling
Blue Ridge Community College

A.J. Krygeris
Houston Community College

Sheila Lancaster
Gadsden State Community College

Ray Larson
Inver Hills Community College

Michelle Levine
Broward College

Jennifer Li
Ohlone College

Norman H. Liebling
San Jacinto College

Cindy Lindstrom
Lakeland College

Zhu-qu Lu
University of Maine, Presque Isle

Heidar Malki
University of Houston

Debbie Mathews
J. Sargeant Reynolds Community College

Svetlana Marzelli
Atlantic Cape Community College

Rick Matzen
Northeastern State University

Robert McDonald
East Stroudsburg University

James McGuffee
Austin Community College

Jie Meichsner
St. Cloud State University

Dean Mellas
Cerritos College

Lisa Milkowski
Milwaukee School of Engineering

Marguerite Nedreberg
Youngstown State University

Lynne O'Hanlon
Los Angeles Pierce College

Frank Paiano
Southwestern Community College

Theresa Park
Texas State Technical College

Mark Parker
Shoreline Community College

Ron Del Porto
Penn State Erie, The Behrend College

Tino Posillico
SUNY Farmingdale

Frederick Pratter
Eastern Oregon University

Susan L. Quick
Penn State University

Alberto Ramon
Diablo Valley College

Bazlur Rasheed
Sault College of Applied Arts and Technology

Farshad Ravanshad
Bergen Community College

Susan Reeder
Seattle University

Sandra Roberts
Snead College

Lopa Roychoudhuri
Angelo State University

Lisa Rudnitsky
Baruch College

Dolly Samson
Weber State University

Ruth Sapir
SUNY Farmingdale

Jason Schatz
City College of San Francisco

Dr. Sung Shin
South Dakota State University

Bari Siddique
University of Texas at Brownsville

William Slater
Collin County Community College

Shep Smithline
University of Minnesota

Richard Snyder
Lehigh Carbon Community College

Donald Southwell
Delta College

Caroline St. Claire
North Central College

Kirk Stephens
Southwestern Community College

Cherie Stevens
South Florida Community College

Dale Suggs
Campbell University

Mark Swanson
Red Wing Technical College

Ann Sudell Thorn
Del Mar College

Martha Tillman
College of San Mateo

Ralph Tomlinson
Iowa State University

David Topham
Ohlone College

Robert Tureman
Paul D. Camp Community College

Arisa K. Ude
Richland College

Peter van der Goes
Rose State College

Stewart Venit
California State University, Los Angeles

Judy Walters
North Central College

John H. Whipple
Northampton Community College

Aurelia Williams
Norfolk State University

Chadd Williams
Pacific University

Vida Winans
Illinois Institute of Technology

I would like to thank the faculty, staff, and administration at Haywood Community College for the opportunity to build a career teaching the subjects that I love. I would also like to thank my family and friends for their support in all of my projects.

It is a great honor to be published by Pearson, and I am extremely fortunate to have Tracy Johnson as my Content Manager. She and her colleagues Holly Stark, Erin Sullivan, Alicia Wilson, Rachel Reeve, Scott Disanno, Bob Engelhardt, Timothy Gardiner, and Carol Snyder have worked tirelessly to produce and promote this book. Thanks to you all!

About the Author

Tony Gaddis is the principal author of the *Starting Out with* series of textbooks. He has two decades of experience teaching computer science courses, primarily at Haywood Community College. Tony is a highly acclaimed instructor who was previously selected as the North Carolina Community College Teacher of the Year and has received the Teaching Excellence award from the National Institute for Staff and Organizational Development. The *Starting Out with* series includes introductory textbooks covering Programming Logic and Design, Alice, C++, Java™, Microsoft® Visual Basic®, Microsoft® Visual C#, Python, and App Inventor, all published by Pearson.

This page intentionally left blank

CHAPTER 1

Introduction to Computers and Programming

TOPICS

1.1 Why Program?
1.2 Computer Systems: Hardware and Software
1.3 Programs and Programming Languages
1.4 What Is a Program Made of?
1.5 Input, Processing, and Output
1.6 The Programming Process
1.7 Procedural and Object-Oriented Programming

1.1 Why Program?

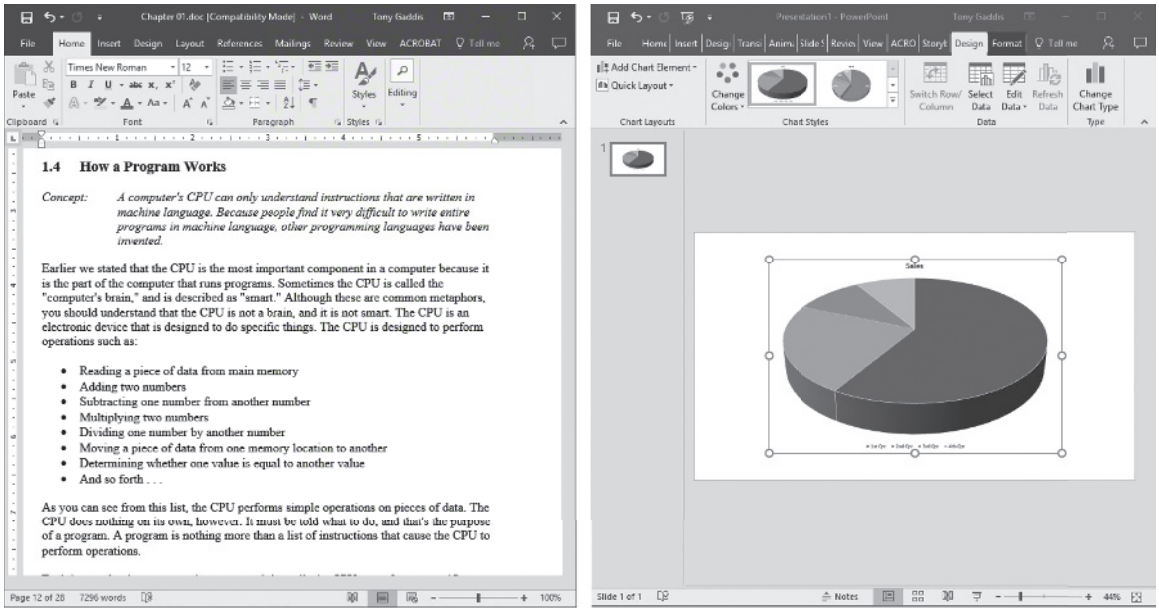
CONCEPT: Computers can do many different jobs because they are programmable.

Think about some of the different ways that people use computers. In school, students use computers for tasks such as writing papers, searching for articles, sending e-mail, and participating in online classes. At work, people use computers to conduct business transactions, communicate with customers and coworkers, analyze data, make presentations, control machines in manufacturing facilities, and many many other tasks. At home, people use computers for tasks such as paying bills, shopping online, social networking, and playing computer games. And don't forget that smartphones, MP3 players, DVRs, car navigation systems, and many other devices are computers as well. The uses of computers are almost limitless in our everyday lives.

Computers can do such a wide variety of things because they can be programmed. This means computers are not designed to do just one job, but any job that their programs tell them to do. A *program* is a set of instructions that a computer follows to perform a task. For example, Figure 1-1 shows screens using Microsoft Word and PowerPoint, two commonly used programs.

Programs are commonly referred to as *software*. Software is essential to a computer because without software, a computer can do nothing. All of the software we use to make our computers useful is created by individuals known as programmers or software developers. A *programmer*, or *software developer*, is a person with the training and skills necessary to design, create, and test computer programs. Computer programming is an exciting and rewarding career. Today, you will find programmers working in business, medicine, government, law enforcement, agriculture, academia, entertainment, and almost every other field.

Figure 1-1 A word processing program and a presentation program



Computer programming is both an art and a science. It is an art because every aspect of a program should be carefully designed. Listed below are a few of the things that must be designed for any real-world computer program:

- The logical flow of the instructions
- The mathematical procedures
- The appearance of the screens
- The way information is presented to the user
- The program's "user-friendliness"
- Documentation, help files, tutorials, and so on

There is also a scientific, or engineering, side to programming. Because programs rarely work right the first time they are written, a lot of testing, correction, and redesigning is required. This demands patience and persistence from the programmer. Writing software demands discipline as well. Programmers must learn special languages like C++ because computers do not understand English or other human languages. Languages such as C++ have strict rules that must be carefully followed.

Both the artistic and scientific nature of programming make writing computer software like designing a car: Both cars and programs should be functional, efficient, powerful, easy to use, and pleasing to look at.

1.2 Computer Systems: Hardware and Software

CONCEPT: **All computer systems consist of similar hardware devices and software components. This section provides an overview of standard computer hardware and software organization.**

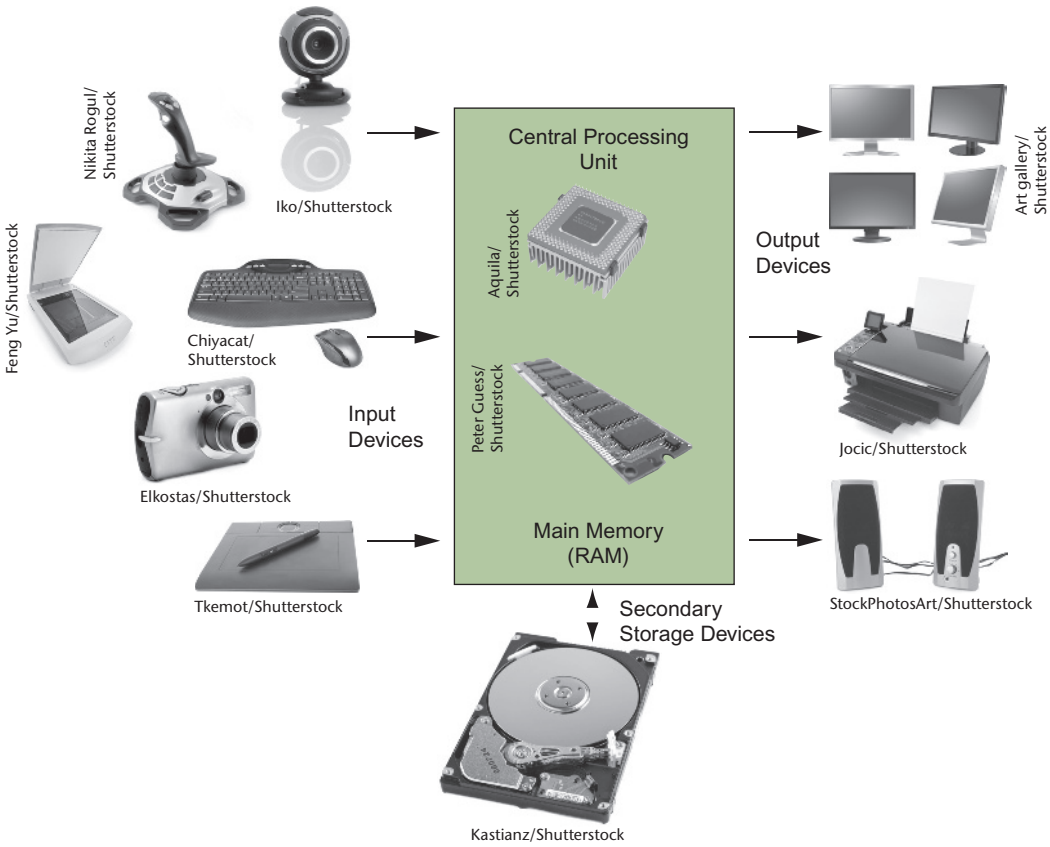
Hardware

Hardware refers to the physical components of which a computer is made. A computer, as we generally think of it, is not an individual device, but a system of devices. Like the instruments in a symphony orchestra, each device plays its own part. A typical computer system consists of the following major components:

- The central processing unit (CPU)
- Main memory
- Secondary storage devices
- Input devices
- Output devices

The organization of a computer system is depicted in Figure 1-2.

Figure 1-2 Typical devices in a computer system

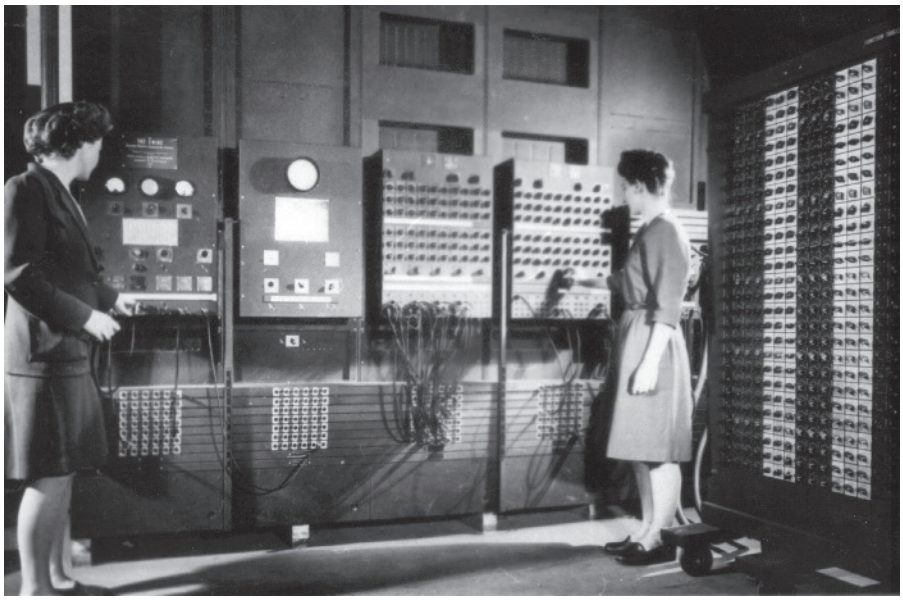


The CPU

When a computer is performing the tasks that a program tells it to do, we say that the computer is *running* or *executing* the program. The *central processing unit*, or *CPU*, is the part of a computer that actually runs programs. The CPU is the most important component in a computer because without it, the computer could not run software.

In the earliest computers, CPUs were huge devices made of electrical and mechanical components such as vacuum tubes and switches. Figure 1-3 shows such a device. The two women in

Figure 1-3 The ENIAC computer

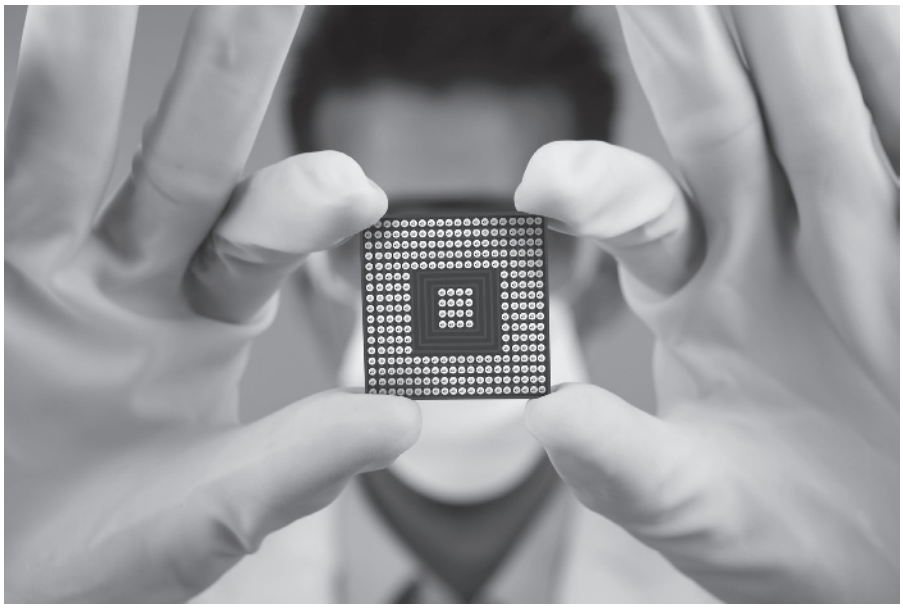


U.S. Army Center of Military History

the photo are working with the historic ENIAC computer. The *ENIAC,* considered by many to be the world's first programmable electronic computer, was built in 1945 to calculate artillery ballistic tables for the U.S. Army. This machine, which was primarily one big CPU, was 8 feet tall, 100 feet long, and weighed 30 tons.

Today, CPUs are small chips known as *microprocessors*. Figure 1-4 shows a photo of a lab technician holding a modern-day microprocessor. In addition to being much smaller than the old electromechanical CPUs in early computers, microprocessors are also much more powerful.

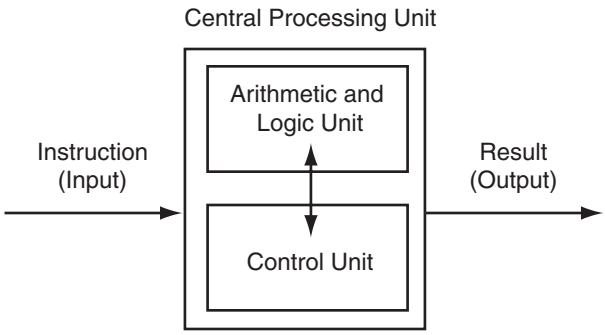
Figure 1-4 A microprocessor



Creativa/Shutterstock

The CPU's job is to fetch instructions, follow the instructions, and produce some result. Internally, the central processing unit consists of two parts: the *control unit* and the *arithmetic and logic unit (ALU)*. The control unit coordinates all of the CPU's operations. It is responsible for determining where to get the next instruction and regulating the other major components of the computer with control signals. The arithmetic and logic unit, as its name suggests, is designed to perform mathematical operations. The organization of the CPU is shown in Figure 1-5.

Figure 1-5 Organization of a CPU



A program is a sequence of instructions stored in the computer's memory. When a computer is running a program, the CPU is engaged in a process known formally as the *fetch/decode/execute cycle*. The steps in the fetch/decode/execute cycle are as follows:

Fetch	The CPU's control unit fetches, from main memory, the next instruction in the sequence of program instructions.
Decode	The instruction is encoded in the form of a number. The control unit decodes the instruction and generates an electronic signal.
Execute	The signal is routed to the appropriate component of the computer (such as the ALU, a disk drive, or some other device). The signal causes the component to perform an operation.

These steps are repeated as long as there are instructions to perform.

Main Memory

You can think of *main memory* as the computer's work area. This is where the computer stores a program while the program is running, as well as the data with which the program is working. For example, suppose you are using a word processing program to write an essay for one of your classes. While you do this, both the word processing program and the essay are stored in main memory.

Main memory is commonly known as *random-access memory* or *RAM*. It is called this because the CPU is able to quickly access data stored at any random location in RAM. RAM is usually a *volatile* type of memory that is used only for temporary storage while a program is running. When the computer is turned off, the contents of RAM are erased. Inside your computer, RAM is stored in small chips.

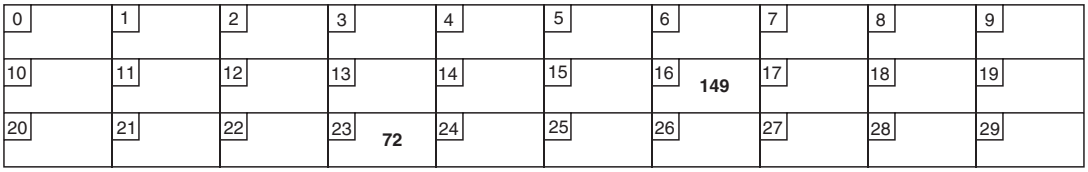
A computer's memory is divided into tiny storage locations known as bytes. One *byte* is enough memory to store only a letter of the alphabet or a small number. In order to do

anything meaningful, a computer must have lots of bytes. Most computers today have billions of bytes of memory.

Each byte is divided into eight smaller storage locations known as bits. The term *bit* stands for *binary digit*. Computer scientists usually think of bits as tiny switches that can be either on or off. Bits aren't actual "switches," however, at least not in the conventional sense. In most computer systems, bits are tiny electrical components that can hold either a positive or a negative charge. Computer scientists think of a positive charge as a switch in the *on* position, and a negative charge as a switch in the *off* position.

Each byte is assigned a unique number known as an *address*. The addresses are ordered from lowest to highest. A byte is identified by its address in much the same way a post office box is identified by an address. Figure 1-6 shows a group of memory cells with their addresses. In the illustration, sample data is stored in memory. The number 149 is stored in the cell with the address 16, and the number 72 is stored at address 23.

Figure 1-6 Memory



Secondary Storage

Secondary storage is a type of memory that can hold data for long periods of time, even when there is no power to the computer. Programs are normally stored in secondary memory and loaded into main memory as needed. Important data such as word processing documents, payroll data, and inventory records is saved to secondary storage as well.

The most common type of secondary storage device is the disk drive. A traditional *disk drive* stores data by magnetically encoding it onto a circular disk. *Solid-state drives*, which store data in solid-state memory, are increasingly becoming popular. A solid-state drive has no moving parts and operates faster than a traditional disk drive. Most computers have some sort of secondary storage device, either a traditional disk drive or a solid-state drive, mounted inside their case. External storage devices can be used to create backup copies of important data or to move data to another computer. For example, *USB (Universal Serial Bus) drives* and *SD (Secure Digital) memory cards* are small devices that appear in the system as disk drives. They are inexpensive, reliable, and small enough to be carried in your pocket.

Input Devices

Input is any data the computer collects from the outside world. The device that collects the information and sends it to the computer is called an *input device*. Common input devices are the keyboard, mouse, touchscreen, scanner, digital camera, and

microphone. Disk drives, CD/DVD drives, and USB drives can also be considered input devices because programs and information are retrieved from them and loaded into the computer's memory.

Output Devices

Output is any information the computer sends to the outside world. It might be a sales report, a list of names, or a graphic image. The information is sent to an *output device*, which formats and presents it. Common output devices are screens, printers, and speakers. Storage devices can also be considered output devices because the CPU sends them data to be saved.

Software

If a computer is to function, software is not optional. Everything a computer does, from the time you turn the power switch on until you shut the system down, is under the control of software. There are two general categories of software: system software and application software. Most computer programs clearly fit into one of these two categories. Let's take a closer look at each.

System Software

The programs that control and manage the basic operations of a computer are generally referred to as *system software*. System software typically includes the following types of programs:

- ***Operating Systems***
  An *operating system* is the most fundamental set of programs on a computer. The operating system controls the internal operations of the computer's hardware, manages all the devices connected to the computer, allows data to be saved to and retrieved from storage devices, and allows other programs to run on the computer.
- ***Utility Programs***
  A *utility program* performs a specialized task that enhances the computer's operation or safeguards data. Examples of utility programs are virus scanners, file-compression programs, and data-backup programs.
- ***Software Development Tools***
  The software tools that programmers use to create, modify, and test software are referred to as *software development tools*. Compilers and integrated development environments, which we will discuss later in this chapter, are examples of programs that fall into this category.

Application Software

Programs that make a computer useful for everyday tasks are known as *application software*. These are the programs that people normally spend most of their time running on their computers. Figure 1-1, at the beginning of this chapter, shows screens from two commonly used applications—Microsoft Word, a word processing program, and Microsoft PowerPoint, a presentation program. Some other examples of application software are spreadsheet programs, e-mail programs, web browsers, and game programs.

Checkpoint

1.1 Why is the computer used by so many different people, in so many different professions?

1.2 List the five major hardware components of a computer system.

1.3 Internally, the CPU consists of what two units?

1.4 Describe the steps in the fetch/decode/execute cycle.

1.5 What is a memory address? What is its purpose?

1.6 Explain why computers have both main memory and secondary storage.

1.7 What are the two general categories of software?

1.8 What fundamental set of programs control the internal operations of the computer's hardware?

1.9 What do you call a program that performs a specialized task, such as a virus scanner, a file-compression program, or a data-backup program?

1.10 Word processing programs, spreadsheet programs, e-mail programs, web browsers, and game programs belong to what category of software?

1.3 Programs and Programming Languages

CONCEPT: **A program is a set of instructions a computer follows in order to perform a task. A programming language is a special language used to write computer programs.**

What Is a Program?

Computers are designed to follow instructions. A computer program is a set of instructions that tells the computer how to solve a problem or perform a task. For example, suppose we want the computer to calculate someone's gross pay. Here is a list of things the computer should do:

1. Display a message on the screen asking "How many hours did you work?"
2. Wait for the user to enter the number of hours worked. Once the user enters a number, store it in memory.
3. Display a message on the screen asking "How much do you get paid per hour?"
4. Wait for the user to enter an hourly pay rate. Once the user enters a number, store it in memory.
5. Multiply the number of hours by the amount paid per hour, and store the result in memory.
6. Display a message on the screen that tells the amount of money earned. The message must include the result of the calculation performed in Step 5.

Collectively, these instructions are called an *algorithm*. An algorithm is a set of well-defined steps for performing a task or solving a problem. Notice these steps are sequentially ordered. Step 1 should be performed before Step 2, and so forth. It is important that these instructions be performed in their proper sequence.

Although you and I might easily understand the instructions in the pay-calculating algorithm, it is not ready to be executed on a computer. A computer's CPU can only process

instructions that are written in *machine language*. If you were to look at a machine language program, you would see a stream of *binary numbers* (numbers consisting of only 1s and 0s). The binary numbers form machine language instructions, which the CPU interprets as commands. Here is an example of what a machine language instruction might look like:

```
1011010000000101
```

As you can imagine, the process of encoding an algorithm in machine language is very tedious and difficult. In addition, each different type of CPU has its own machine language. If you wrote a machine language program for computer *A* then wanted to run it on computer *B*, which has a different type of CPU, you would have to rewrite the program in computer B's machine language.

Programming languages, which use words instead of numbers, were invented to ease the task of programming. A program can be written in a programming language, such as C++, which is much easier to understand than machine language. Programmers save their programs in text files, then use special software to convert their programs to machine language.

Program 1-1 shows how the pay-calculating algorithm might be written in C++.

The "Program Output with Example Input" shows what the program will display on the screen when it is running. In the example, the user enters 10 for the number of hours worked and 15 for the hourly pay rate. The program displays the earnings, which are $150.

NOTE: The line numbers that are shown in Program 1-1 are *not* part of the program. This book shows line numbers in all program listings to help point out specific parts of the program.

Program 1-1

```
 1 // This program calculates the user's pay.
 2 #include <iostream>
 3 using namespace std;
 4
 5 int main()
 6 {
 7     double hours, rate, pay;
 8
 9     // Get the number of hours worked.
10     cout << "How many hours did you work? ";
11     cin >> hours;
12
13     // Get the hourly pay rate.
14     cout << "How much do you get paid per hour? ";
15     cin >> rate;
16
17     // Calculate the pay.
18     pay = hours * rate;
```

(program continues)