

starting out with >>>

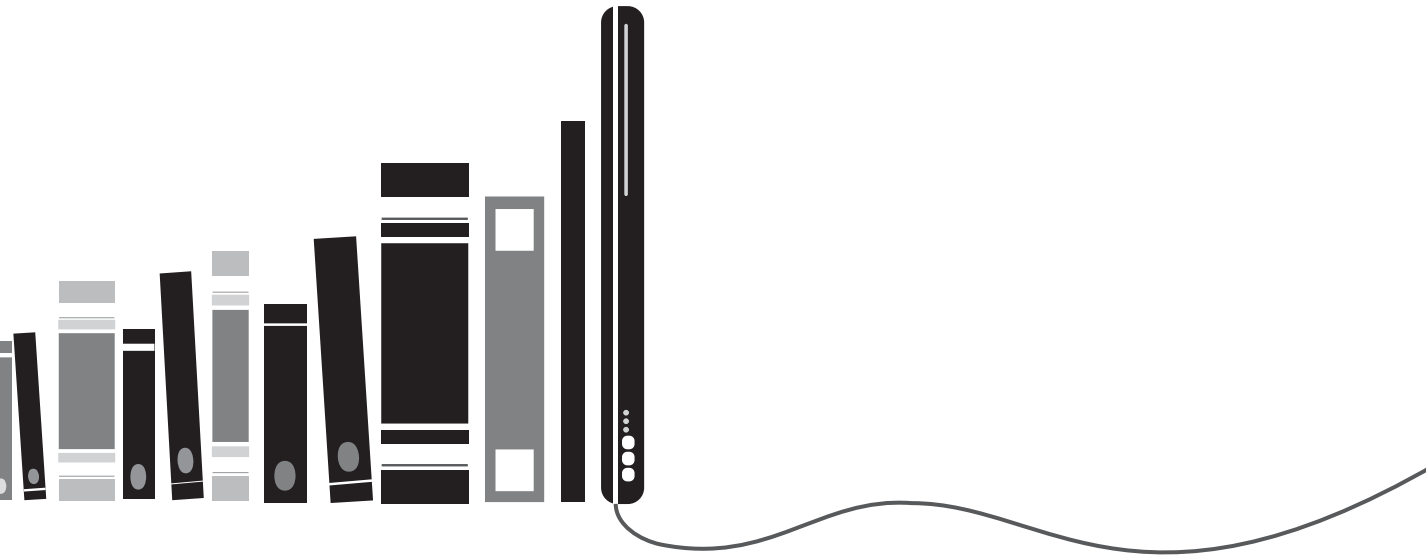
# C++

## EARLY OBJECTS

EIGHTH EDITION



TONY GADDIS · JUDY WALTERS · GODFREY MUGANDA



# get with the programming

Through the power of practice and immediate personalized feedback, MyProgrammingLab improves your performance.

## MyProgrammingLab™

Learn more at [www.myprogramminglab.com](http://www.myprogramminglab.com)

*This page intentionally left blank*

## LOCATION OF VIDEONOTES IN THE TEXT



<b>Chapter 1</b>	Designing a Program with Pseudocode, p. 20 Designing the Account Balance Program, p. 24 Predicting the Output of Problem 30, p. 25 Solving the Candy Bar Sales Problem, p. 26
<b>Chapter 2</b>	Using <code>cout</code> to Display Output, p. 32 Assignment Statements, p. 60 Arithmetic Operators, p. 61 Solving the Restaurant Bill Problem, p. 73
<b>Chapter 3</b>	Using <code>cin</code> to Read Input, p. 77 Evaluating Mathematical Expressions, p. 84 Combined Assignment Operators, p. 105 Solving the Stadium Seating Problem, p. 149
<b>Chapter 4</b>	Using an <code>if</code> Statement, p. 160 Using an <code>if/else</code> Statement, p. 169 Using an <code>if/else if</code> Statement, p. 174 Using Logical Operators, p. 187 Solving the Time Calculator Problem, p. 236
<b>Chapter 5</b>	The <code>while</code> Loop, p. 244 The <code>for</code> Loop, p. 266 Nested Loops, p. 277 Solving the Ocean Levels Problem, p. 318
<b>Chapter 6</b>	Defining and Calling Functions, p. 324 Using Function Arguments, p. 334 Value-Returning Functions, p. 344 Solving the Markup Problem, p. 399
<b>Chapter 7</b>	Creating a Class, p. 412 Creating and Using Class Objects, p. 414 Creating and Using Structures, p. 454 Solving the <code>car</code> Class Problem, p. 498
<b>Chapter 8</b>	Accessing Array Elements, p. 505 Passing an Array to a Function, p. 535 Two-Dimensional Arrays, p. 545 Solving the Chips and Salsa Problem, p. 586
<b>Chapter 9</b>	Performing a Binary Search, p. 598 Sorting a Set of Data, p. 605 Solving the Lottery Winners Problem, p. 634

(continued on next page)

## LOCATION OF VIDEONOTES IN THE TEXT *(continued)*



<b>Chapter 10</b>	Pointer Variables, p. 639 Dynamically Allocating an Array, p. 663 Solving the Days in Current Month Problem, p. 693
<b>Chapter 11</b>	Operator Overloading, p. 722 Aggregation and Composition, p. 752 Overriding Base Class Functions, p. 773 Solving the Number of Days Worked Problem, p. 786
<b>Chapter 12</b>	Converting Strings to Numbers, p. 806 Writing a C-String Handling Function, p. 811 Solving the Case Manipulator Problem, p. 834
<b>Chapter 13</b>	The <code>get</code> Family of Member Functions, p. 853 Rewinding a File, p. 857 Solving the File Encryption Filter Problem, p. 895
<b>Chapter 14</b>	Recursive Binary Search, p. 911 QuickSort, p. 913 Solving the Recursive Multiplication Problem, p. 931
<b>Chapter 15</b>	Polymorphism, p. 939 Composition versus Inheritance, p. 950 Solving the Sequence Sum Problem, p. 968
<b>Chapter 16</b>	Throwing and Handling Exceptions, p. 972 Writing a Function Template, p. 984 Iterators, p. 1002 Solving the Arithmetic Exceptions Problem, p. 1018
<b>Chapter 17</b>	Adding an Element to a Linked List, p. 1029 Removing an Element from a Linked List, p. 1036 Solving the Member Insertion by Position Problem, p. 1067
<b>Chapter 18</b>	Storing Objects in an STL Stack, p. 1081 Storing Objects in an STL Queue, p. 1095 Solving the File Reverser Problem, p. 1107
<b>Chapter 19</b>	Inserting an Element into a Binary Tree, p. 1116 Removing an Element from a Binary Tree, p. 1120 Solving the Tree Size Problem, p. 1136

Eighth  
Edition

Starting Out with

C++  
Early Objects

Tony Gaddis  
Judy Walters  
Godfrey Muganda

PEARSON

Boston Columbus Indianapolis New York San Francisco Upper Saddle River  
Amsterdam Cape Town Dubai London Madrid Milan Munich Paris Montreal Toronto  
Delhi Mexico City Sao Paulo Sydney Hong Kong Seoul Singapore Taipei Tokyo

**Editorial Director, ECS:** Marcia Horton  
**Executive Editor:** Matt Goldstein  
**Editorial Assistant:** Jenah Blitz-Stoehr  
**Director of Marketing:** Christy Lesko  
**Marketing Manager:** Yezan Alayan  
**Senior Senior Marketing Coordinator:** Kathryn Ferranti  
**Director of Production:** Erin Gregg  
**Senior Managing Editor:** Scott Disanno  
**Production Project Manager:** Kayla Smith-Tarbox  
**Manufacturing Buyer:** Lisa McDowell  
**Art Director:** Anthony Gemmellaro

**Cover Designer:** Joyce Wells  
**Manager, Rights and Permissions:** Michael Joyce  
**Text Permission Coordinator:** Jackie Bates, GEX inc.  
**Cover Image:** Svetlana Kuznetsova / Shutterstock  
**Media Project Manager:** Renata Butera  
**Full-Service Project Management:** Mohinder Singh/Aptara®, Inc.  
**Composition:** Aptara®, Inc.  
**Printer/Binder:** Edwards Brothers  
**Cover Printer:** Lehigh-Phoenix Color/Hagerstown  
**Text Font:** Sabon

Credits and acknowledgments borrowed from other sources and reproduced, with permission, in this textbook appear on appropriate page within text.

**Credits:** Figure 1-1a: Microsoft Powerpoint and Microsoft Word, Microsoft Corporation. 2010.

Reference: The most commonly used method for encoding characters is ASCII...(cont.)The American Standard Code for Information Interchange. American National Standards Institute. 2012.

Reference: “QuickSort is a recursive sorting algorithm that was invented in 1960 by C. A. R. Hoare.” Hoare, C.A.R. “QuickSort”. Oxford University Press. 1960.

Copyright © 2012 by Microsoft Corporation. Used with permission from Microsoft.

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.

The programs and applications presented in this book have been included for their instructional value. They have been tested with care, but are not guaranteed for any particular purpose. The publisher does not offer any warranties or representations, nor does it accept any liabilities with respect to the programs or applications.

---

Copyright © 2014, 2008. Pearson Education, Inc., publishing as Addison-Wesley, 501 Boylston Street, Suite 900, Boston, Massachusetts 02116. 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 likewise. To obtain permission(s) to use material from this work, please submit a written request to Pearson Education, Inc., Permissions Department, 501 Boylston Street, Suite 900, Boston, Massachusetts 02116.

Many of the designations by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed in initial caps or all caps.

#### Library of Congress Cataloging-in-Publication Data

Gaddis, Tony.

Starting out with C++ : early objects / Tony Gaddis, Judy Walters, Godfrey Muganda.—Eighth edition.

pages cm

ISBN-13: 978-0-13-336092-9

ISBN-10: 0-13-336092-X

1. C++ (Computer program language) I. Walters, Judy. II. Muganda, Godfrey. III. Title.

QA76.73.C153G33 2014

005.13'3—dc23

2012045400

10 9 8 7 6 5 4 3 2 1

**PEARSON**

ISBN 10: 0-13-336092-X  
ISBN 13: 978-0-13-336092-9

# Contents

## Preface xiii

### CHAPTER 1 Introduction to Computers and Programming 1

- 1.1 Why Program? 1
- 1.2 Computer Systems: Hardware and Software 3
- 1.3 Programs and Programming Languages 7
- 1.4 What Is a Program Made of? 13
- 1.5 Input, Processing, and Output 17
- 1.6 The Programming Process 18
- 1.7 Tying It All Together: *Hi! It's Me* 23

### 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 36
- 2.4 Standard and Prestandard C++ 37
- 2.5 Variables, Literals, and the Assignment Statement 38
- 2.6 Identifiers 42
- 2.7 Integer Data Types 43
- 2.8 Floating-Point Data Types 49
- 2.9 The `char` Data Type 52
- 2.10 The C++ `string` Class 56
- 2.11 The `bool` Data Type 58
- 2.12 Determining the Size of a Data Type 59
- 2.13 More on Variable Assignments and Initialization 59
- 2.14 Scope 61
- 2.15 Arithmetic Operators 61
- 2.16 Comments 65
- 2.17 Programming Style 67
- 2.18 Tying It All Together: *Smile!* 69



**CHAPTER 3 Expressions and Interactivity 77**

- 3.1 The `cin` Object 77
- 3.2 Mathematical Expressions 84
- 3.3 Data Type Conversion and Type Casting 91
- 3.4 Overflow and Underflow 98
- 3.5 Named Constants 99
- 3.6 Multiple and Combined Assignment 104
- 3.7 Formatting Output 108
- 3.8 Working with Characters and Strings 118
- 3.9 Using C-Strings 125
- 3.10 More Mathematical Library Functions 131
- 3.11 Focus on Debugging: *Hand Tracing a Program* 136
- 3.12 Green Fields Landscaping Case Study—Part 1 138
- 3.13 Tying It All Together: *Word Game* 140

**CHAPTER 4 Making Decisions 155**

- 4.1 Relational Operators 155
- 4.2 The `if` Statement 160
- 4.3 The `if/else` Statement 169
- 4.4 The `if/else if` Statement 174
- 4.5 Menu-Driven Programs 181
- 4.6 Nested `if` Statements 183
- 4.7 Logical Operators 187
- 4.8 Validating User Input 196
- 4.9 More About Blocks and Scope 197
- 4.10 More About Characters and Strings 200
- 4.11 The Conditional Operator 207
- 4.12 The `switch` Statement 210
- 4.13 Enumerated Data Types 219
- 4.14 Focus on Testing and Debugging: *Validating Output Results* 222
- 4.15 Green Fields Landscaping Case Study—Part 2 225
- 4.16 Tying It All Together: *Fortune Teller* 229

**CHAPTER 5 Looping 243**

- 5.1 Introduction to Loops: The `while` Loop 243
- 5.2 Using the `while` Loop for Input Validation 250
- 5.3 The Increment and Decrement Operators 252
- 5.4 Counters 258
- 5.5 The `do-while` Loop 260
- 5.6 The `for` Loop 266
- 5.7 Keeping a Running Total 272
- 5.8 Sentinels 274
- 5.9 Focus on Software Engineering: *Deciding Which Loop to Use* 276
- 5.10 Nested Loops 277
- 5.11 Breaking Out of a Loop 280
- 5.12 Using Files for Data Storage 284
- 5.13 Focus on Testing and Debugging: *Creating Good Test Data* 302
- 5.14 Central Mountain Credit Union Case Study 305
- 5.15 Tying It All Together: *What a Colorful World* 309

**CHAPTER 6 Functions 323**

- 6.1 Modular Programming 323
- 6.2 Defining and Calling Functions 324
- 6.3 Function Prototypes 332
- 6.4 Sending Data into a Function 334
- 6.5 Passing Data by Value 339
- 6.6 The `return` Statement 343
- 6.7 Returning a Value from a Function 344
- 6.8 Returning a Boolean Value 350
- 6.9 Using Functions in a Menu-Driven Program 352
- 6.10 Local and Global Variables 355
- 6.11 Static Local Variables 363
- 6.12 Default Arguments 365
- 6.13 Using Reference Variables as Parameters 369
- 6.14 Overloading Functions 377
- 6.15 The `exit()` Function 382
- 6.16 Stubs and Drivers 384
- 6.17 Little Lotto Case Study 387
- 6.18 Tying It All Together: *Glowing Jack-o-lantern* 392

**CHAPTER 7 Introduction to Classes and Objects 407**

- 7.1 Abstract Data Types 407
- 7.2 Object-Oriented Programming 409
- 7.3 Introduction to Classes 411
- 7.4 Creating and Using Objects 414
- 7.5 Defining Member Functions 416
- 7.6 Constructors 423
- 7.7 Destructors 429
- 7.8 Private Member Functions 432
- 7.9 Passing Objects to Functions 435
- 7.10 Object Composition 442
- 7.11 Focus on Software Engineering: *Separating Class Specification, Implementation, and Client Code* 446
- 7.12 Structures 453
- 7.13 Home Software Company OOP Case Study 467
- 7.14 Introduction to Object-Oriented Analysis and Design 474
- 7.15 Screen Control 484
- 7.16 Tying It All Together: *Yoyo Animation* 489

**CHAPTER 8 Arrays 503**

- 8.1 Arrays Hold Multiple Values 503
- 8.2 Accessing Array Elements 505
- 8.3 Inputting and Displaying Array Contents 507
- 8.4 Array Initialization 514
- 8.5 Processing Array Contents 520
- 8.6 Using Parallel Arrays 531
- 8.7 The `typedef` Statement 535
- 8.8 Arrays as Function Arguments 535
- 8.9 Two-Dimensional Arrays 545
- 8.10 Arrays with Three or More Dimensions 553

- 8.11 Vectors 556
- 8.12 Arrays of Objects 568
- 8.13 National Commerce Bank Case Study 578
- 8.14 Tying It All Together: *Rock, Paper, Scissors* 580

## **CHAPTER 9 Searching, Sorting, and Algorithm Analysis 595**

- 9.1 Introduction to Search Algorithms 595
- 9.2 Searching an Array of Objects 602
- 9.3 Introduction to Sorting Algorithms 605
- 9.4 Sorting an Array of Objects 614
- 9.5 Sorting and Searching Vectors 617
- 9.6 Introduction to Analysis of Algorithms 619
- 9.7 Case Studies 627
- 9.8 Tying It All Together: *Secret Messages* 628

## **CHAPTER 10 Pointers 637**

- 10.1 Pointers and the Address Operator 637
- 10.2 Pointer Variables 639
- 10.3 The Relationship Between Arrays and Pointers 643
- 10.4 Pointer Arithmetic 647
- 10.5 Initializing Pointers 648
- 10.6 Comparing Pointers 650
- 10.7 Pointers as Function Parameters 653
- 10.8 Pointers to Constants and Constant Pointers 657
- 10.9 Focus on Software Engineering: *Dynamic Memory Allocation* 661
- 10.10 Focus on Software Engineering: *Returning Pointers from Functions* 666
- 10.11 Pointers to Class Objects and Structures 670
- 10.12 Focus on Software Engineering: *Selecting Members of Objects* 676
- 10.13 United Cause Relief Agency Case Study 678
- 10.14 Tying It All Together: *Pardon Me, Do You Have the Time?* 686

## **CHAPTER 11 More About Classes and Object-Oriented Programming 695**

- 11.1 The `this` Pointer and Constant Member Functions 695
- 11.2 Static Members 700
- 11.3 Friends of Classes 707
- 11.4 Memberwise Assignment 712
- 11.5 Copy Constructors 713
- 11.6 Operator Overloading 722
- 11.7 Type Conversion Operators 746
- 11.8 Convert Constructors 749
- 11.9 Aggregation and Composition 752
- 11.10 Inheritance 758
- 11.11 Protected Members and Class Access 763
- 11.12 Constructors, Destructors, and Inheritance 768
- 11.13 Overriding Base Class Functions 773
- 11.14 Tying It All Together: *Putting Data on the World Wide Web* 775

**CHAPTER 12 More on C-Strings and the `string` Class 789**

- 12.1 C-Strings 789
- 12.2 Library Functions for Working with C-Strings 794
- 12.3 Conversions Between Numbers and Strings 805
- 12.4 Writing Your Own C-String Handling Functions 811
- 12.5 More About the C++ `string` Class 816
- 12.6 Creating Your Own String Class 820
- 12.7 Advanced Software Enterprises Case Study 827
- 12.8 Tying It All Together: *Program Execution Environments* 828

**CHAPTER 13 Advanced File and I/O Operations 837**

- 13.1 Input and Output Streams 837
- 13.2 More Detailed Error Testing 845
- 13.3 Member Functions for Reading and Writing Files 848
- 13.4 Binary Files 861
- 13.5 Creating Records with Structures 865
- 13.6 Random-Access Files 870
- 13.7 Opening a File for Both Input and Output 876
- 13.8 Online Friendship Connections Case Study: *Object Serialization* 881
- 13.9 Tying It All Together: *File Merging and Color-Coded HTML* 887

**CHAPTER 14 Recursion 899**

- 14.1 Introduction to Recursion 899
- 14.2 The Recursive Factorial Function 906
- 14.3 The Recursive gcd Function 908
- 14.4 Solving Recursively Defined Problems 909
- 14.5 A Recursive Binary Search Function 911
- 14.6 Focus on Problem Solving and Program Design: *The QuickSort Algorithm* 913
- 14.7 The Towers of Hanoi 917
- 14.8 Focus on Problem Solving: *Exhaustive and Enumeration Algorithms* 920
- 14.9 Focus on Software Engineering: *Recursion versus Iteration* 924
- 14.10 Tying It All Together: *Infix and Prefix Expressions* 925

**CHAPTER 15 Polymorphism and Virtual Functions 933**

- 15.1 Type Compatibility in Inheritance Hierarchies 933
- 15.2 Polymorphism and Virtual Member Functions 939
- 15.3 Abstract Base Classes and Pure Virtual Functions 944
- 15.4 Focus on Object-Oriented Programming: *Composition versus Inheritance* 950
- 15.5 Secure Encryption Systems, Inc., Case Study 955
- 15.6 Tying It All Together: *Let's Move It* 959

**CHAPTER 16 Exceptions, Templates, and the Standard Template Library (STL) 971**

- 16.1 Exceptions 971
- 16.2 Function Templates 983
- 16.3 Class Templates 991
- 16.4 Class Templates and Inheritance 996
- 16.5 Introduction to the Standard Template Library 1000
- 16.6 Tying It All Together: *Word Transformers Game* 1013

**CHAPTER 17 Linked Lists 1021**

- 17.1 Introduction to the Linked List ADT 1021
- 17.2 Linked List Operations 1027
- 17.3 A Linked List Template 1039
- 17.4 Recursive Linked List Operations 1043
- 17.5 Variations of the Linked List 1052
- 17.6 The STL `list` Container 1052
- 17.7 Reliable Software Systems, Inc., Case Study 1054
- 17.8 Tying It All Together: *More on Graphics and Animation* 1058

**CHAPTER 18 Stacks and Queues 1069**

- 18.1 Introduction to the Stack ADT 1069
- 18.2 Dynamic Stacks 1077
- 18.3 The STL `stack` Container 1080
- 18.4 Introduction to the Queue ADT 1082
- 18.5 Dynamic Queues 1090
- 18.6 The STL `deque` and `queue` Containers 1094
- 18.7 Focus on Problem Solving and Program Design: *Eliminating Recursion* 1096
- 18.8 Tying It All Together: *Converting Postfix Expressions to Infix* 1101

**CHAPTER 19 Binary Trees 1109**

- 19.1 Definition and Applications of Binary Trees 1109
- 19.2 Binary Search Tree Operations 1113
- 19.3 Template Considerations for Binary Search Trees 1129
- 19.4 Tying It All Together: *Genealogy Trees* 1129

**Appendix A: The ASCII Character Set 1139****Appendix B: Operator Precedence and Associativity 1143****Appendix C: Answers to Checkpoints 1145****Appendix D: Answers to Odd-Numbered Review Questions 1185****Index 1207****Additional Appendices**

The following appendices are located on the book's companion web site.

**Appendix E: A Brief Introduction to Object-Oriented Programming****Appendix F: Using UML in Class Design****Appendix G: Multi-Source File Programs****Appendix H: Multiple and Virtual Inheritance****Appendix I: Header File and Library Function Reference****Appendix J: Namespaces****Appendix K: C++ Casts and Run-Time Type Identification****Appendix L: Passing Command Line Arguments****Appendix M: Binary Numbers and Bitwise Operations****Appendix N: Introduction to Flowcharting**

# Preface

Welcome to *Starting Out with C++: Early Objects*, 8th Edition. This book is intended for use in a two-term or three-term C++ programming sequence, or an accelerated one-term course. Students new to programming, as well 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.

## What's New in the Eighth Edition

This book's pedagogy and clear writing style remain the same as in the previous edition. However, many improvements have been made to make it even more student-friendly and to keep it state of the art for introductory programming using the C++ programming language.

- **Updated Material**  
Material has been updated throughout the book to reflect changes in technology, operating systems, and software development environments, as well as to improve clarity and incorporate best practices in object-oriented programming.
- **New Material**  
New material has been added on a number of topics including expanded coverage on using files. Chapter 5 now brings together, adds to, and better organizes the material on files formerly found in Chapters 3, 4, and 5.
- **Reorganized Chapters**  
Several chapters have been reorganized to improve student learning. Chapter 2, Introduction to C++, now covers integer and floating-point data types before introducing characters and strings. Chapter 5, Looping, now discusses how looping structures are used before introducing the mechanics of creating them. Chapter 7, Introduction to Classes and Objects, now revisits a class students already know and have been using, the `string` class, before introducing how to create and use their own classes and objects.

- **Greater Focus on Object-Oriented Programming**  
Many examples throughout the text have been rewritten to incorporate appropriate use of classes and objects.
- **Improved Sample Programs**  
Sample programs have been revised where appropriate to incorporate current best programming practices. For example, throughout the book functions receiving objects or arrays whose values should not be changed now use the `const` keyword to protect them.
- **Improved Diagrams**  
Many diagrams have been improved and new diagrams added to better illustrate important concepts.
- **New Programming Challenges**  
New Programming Challenges have been added in many chapters, including a number of Challenges that ask students to develop object-oriented solutions and to create solutions that reuse, modify, and build on previously written code.
- **Answers in the Book**  
Answers to all the Checkpoint questions throughout the book and to the odd-numbered review questions at the end of every chapter are now conveniently located at the back of the book in Appendices C and D.

## Organization of the Text

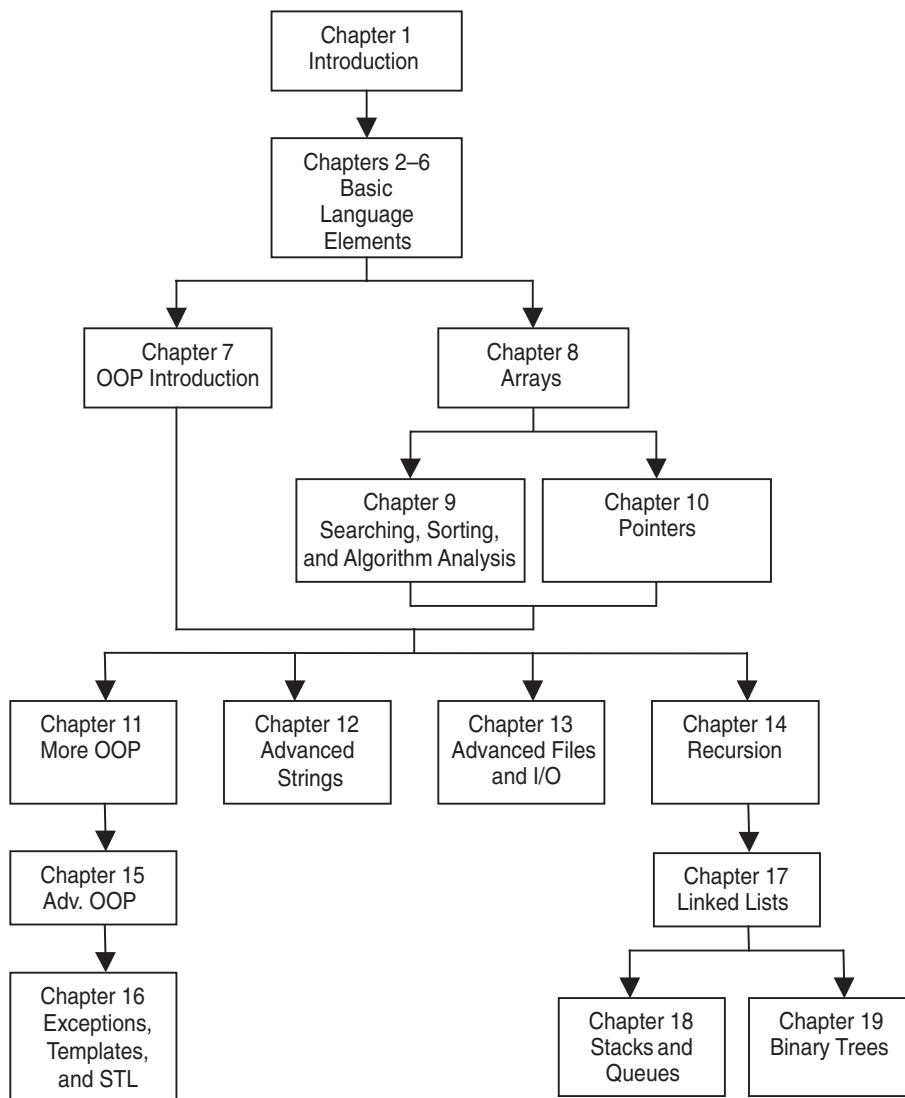
This text teaches C++ in a step-by-step fashion. Each chapter covers a major set of topics and builds knowledge as the student progresses through the book. Although the chapters can be easily taught in their existing sequence, flexibility is provided. The following dependency diagram (Figure P-1) suggests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

Chapter 1 covers fundamental hardware, software, and programming concepts. The instructor may choose to skip this chapter if the class has already mastered those topics. Chapters 2 through 6 cover basic C++ syntax, data types, expressions, selection structures, repetition structures, and functions. Each of these chapters builds on the previous chapter and should be covered in the order presented.

Chapter 7 introduces object-oriented programming. It can be covered any time after Chapter 6, but before Chapter 11. Instructors who prefer to introduce arrays before classes can cover Chapter 8 before Chapter 7. In this case it is only necessary to postpone Section 8.12 (Arrays of Objects) until Chapter 7 has been covered.

As Figure P-1 illustrates, in the second half of the book Chapters 11, 12, 13, and 14 can be covered in any order. Chapters 11, 15, and 16, however, should be done in sequence. Instructors who wish to introduce data structures at an earlier point in the course, without having first covered advanced C++ and OOP features, can cover Chapter 17 (Linked Lists), followed by Chapters 18 and 19 (Stacks & Queues and Binary Trees), any time after Chapter 14 (Recursion). In this case it is necessary to simply omit the sections in Chapters 17–19 that deal with templates and the Standard Template Library.

Figure P-1





## 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 hardware, software, operating systems, programming, problem solving, and software engineering. The components of programs, such as key words, variables, operators, and punctuation are covered. The tools of the trade, such as hierarchy charts and pseudocode, are also presented. The *Tying It All Together* section shows students how to use the `cout` statement to create a personalized output message. Programming Challenges at the end of the chapter help students see how the same basic input, processing, and output structure can be used to create multiple programs.

### Chapter 2: Introduction to C++

This chapter gets the student started in C++ by introducing the basic parts of a C++ program, data types, the use of variables and literals, assignment statements, simple arithmetic operations, program output, and comments. The C++ `string` class is presented and `string` objects are used from this point on in the book as the primary method of handling strings. Programming style conventions are introduced, and good programming style is modeled here, as it is throughout the text. An optional section explains the difference between ANSI standard and prestandard C++ programs. The *Tying It All Together* section lets the student play with simple text-based graphics.

### 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, with emphasis on operator precedence. Debugging is introduced, with a section on hand tracing a program. Sections are also included on using random numbers, on simple output formatting, on data type conversion and type casting, and on using library functions that work with numbers. For those who wish to cover them, there is also a section on C-strings. The *Tying It All Together* section shows students how to create a simple interactive word game.

### Chapter 4: Making Decisions

Here the student learns about relational expressions and how to control the flow of a program with `if`, `if/else`, and `if/else if` statements. Logical operators, the conditional operator, and the `switch` statement are also covered. Applications of these constructs, such as menu-driven programs, are illustrated. This chapter also continues the theme of debugging with a section on validating output results. The *Tying It All Together* section uses random numbers and branching statements to create a fortune telling game.

### Chapter 5: Looping

This chapter introduces, C++'s repetitive control mechanisms. The `while` loop, `do-while` loop, and `for` loop are presented, along with a variety of methods to control them. These include using counters, user input, end sentinels, and end-of-file testing. Applications utilizing loops, such as keeping a running total and performing data validation, are also covered. An extensive new section on working with files has been added, and the emphasis on testing and debugging continues, with a section on creating good test data. The chapter's *Tying It All Together* section introduces students to Windows commands to create colorful output and uses a loop to create a multi-colored display.

## **Chapter 6: Functions**

In this chapter the student learns how and why to modularize programs, using both `void` and value-returning functions. Parameter 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. The *Tying It All Together* section includes a modular, menu-driven program that emphasizes the versatility of functions, illustrating how their behavior can be controlled by the arguments sent to them.

## **Chapter 7: Introduction to Classes and Objects**

In this chapter the text begins to focus on the object-oriented paradigm. Students have used provided C++ classes since the beginning of the text, but now they learn how to define their own classes and to create and use objects of these classes. Careful attention is paid to illustrating which functions belong in a class versus which functions belong in a client program that uses the class. Good object-oriented practices are discussed and modeled, such as protecting member data through carefully constructed accessor and mutator functions and hiding class implementation details from client programs. Once students are comfortable working with classes and objects, the chapter provides a brief introduction to the topic of object-oriented analysis and design. The chapter also introduces structures and uses them in the *Tying It All Together* section, where students learn to use screen control techniques to create an animation that simulates the motion of a yoyo.

## **Chapter 8: Arrays**

In this chapter the student learns to create and work with single and multidimensional arrays. Many examples of array processing are provided, including functions to compute the sum, average, highest and lowest values in an array. Students also learn to create tables using two-dimensional arrays, and to analyze the array data by row or by column. 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. A section on arrays of objects and structures is located at the end of the chapter, so it can be covered now or saved for later if the instructor wishes to cover this chapter before Chapter 7. The *Tying It All Together* section uses arrays to create a game of *Rock, Paper, Scissors* between a human player and the computer.

## **Chapter 9: Searching, Sorting, and Algorithm Analysis**

Here the student learns the basics of searching for information stored in arrays and of sorting arrays, including arrays of objects. The chapter covers the Linear Search, Binary Search, Bubble Sort, and Selection Sort algorithms and has an optional section on sorting and searching STL vectors. A brief introduction to algorithm analysis is included, and students are shown how to determine which of two algorithms is more efficient. This chapter's *Tying It All Together* section uses both a table lookup and a searching algorithm to encode and decode secret messages.

## **Chapter 10: Pointers**

This chapter explains how to use pointers. Topics include pointer arithmetic, initialization of pointers, comparison of pointers, pointers and arrays, pointers and functions, dynamic memory allocation, and more. The *Tying It All Together* section demonstrates the use of pointers to access library data structures and functions that return calendar and wall clock time.

### **Chapter 11: More About Classes and Object-Oriented Programming**

This chapter continues the study of classes and object-oriented programming. It covers object aggregation and composition, as well as inheritance, and illustrates the difference between is-a and has-a relations. Constant member functions, static members, friends, memberwise assignment, copy constructors, object type conversion operators, convert constructors, and operator overloading are also included. The *Tying It All Together* section brings together the concepts of inheritance and convert constructors to build a program that formats the contents of an array to form an HTML table for display on a Web site.

### **Chapter 12: More on C-Strings and the `string` Class**

This chapter covers standard library functions for working with characters and C-strings, covering topics such as passing C-strings to functions and using the C++ `sstream` classes to convert between numeric and string forms of numbers. Additional material about the C++ `string` class and its member functions and operators is presented, with a program illustrating how to write your own string class. The *Tying It All Together* section shows students how to access string-based program environments to obtain information about the computer and the network on which the program is running.

### **Chapter 13: Advanced File and I/O Operations**

This chapter introduces more advanced topics for working with sequential access text files and introduces random access and binary files. Various modes for opening files are discussed, as well as the many methods for reading and writing their contents. The *Tying It All Together* program applies many of the techniques covered in the chapter to merge two text files into an HTML document for display on the Web, with different colors used to illustrate which file each piece of data came from.

### **Chapter 14: Recursion**

In this chapter recursion is defined and demonstrated. A visual trace of recursive calls is provided, and recursive applications are discussed. Many recursive algorithms are presented, including recursive functions for computing factorials, finding a greatest common denominator (GCD), performing a binary search, sorting using QuickSort, and solving the famous Towers of Hanoi problem. For students who need more challenge, there is a section on exhaustive and enumeration algorithms. The *Tying It All Together* section uses recursion to evaluate prefix expressions.

### **Chapter 15: Polymorphism and Virtual Functions**

The study of classes and object-oriented programming continues in this chapter with the introduction of more advanced concepts such as polymorphism and virtual functions. Information is also presented on abstract base classes, pure virtual functions, type compatibility within an inheritance hierarchy, and virtual inheritance. The *Tying It All Together* section illustrates the use of inheritance and polymorphism to display and animate graphical images.

**Chapter 16: Exceptions, Templates, and the Standard Template Library (STL)**

Here the student learns to develop enhanced error trapping techniques using exceptions. Discussion then turns to using function and class templates to create generic code. Finally, the student is introduced to the containers, iterators, and algorithms offered by the Standard Template Library (STL). The *Tying It All Together* section uses various containers in the Standard Template Library to create an educational children's game.

**Chapter 17: Linked Lists**

This chapter introduces concepts and techniques needed to work with lists. A linked list ADT is developed, and the student learns how to create and destroy a list, as well as to write functions to insert, append, and delete nodes, to traverse the list, and to search for a specific node. A linked list class template is also demonstrated. The *Tying It All Together* section brings together many of the most important concepts of OOP by using objects, inheritance, and polymorphism in conjunction with the STL list class to animate a collection of images.

**Chapter 18: 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. The static array-based stack uses exception-handling to handle stack overflow and underflow, providing a realistic and natural example of defining, throwing, and catching exceptions. The *Tying It All Together* section discusses strategies for evaluating postfix expressions and uses a stack to convert a postfix expression to infix.

**Chapter 19: Binary Trees**

This chapter covers the binary tree ADT and demonstrates many binary tree operations. The student learns to traverse a tree, insert, delete, and replace elements, search for a particular element, and destroy a tree. The *Tying It All Together* section introduces a tree structure versatile enough to create genealogy trees.

## Appendices in the Book

**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 list of the C++ operators with their precedence and associativity.

**Appendix C: Answers to Checkpoints** A tool students can use to assess their understanding by comparing their answers to the Checkpoint exercises found throughout the book. The answers to all Checkpoint exercises are included.

**Appendix D: Answers to Odd-Numbered Review Questions** Another tool students can use to gauge their understanding and progress.

## Additional Appendices on the Book's Companion Website

**Appendix E: A Brief Introduction to Object-Oriented Programming** An introduction to the concepts and terminology of object-oriented programming.

**Appendix F: Using UML in Class Design** A brief introduction to the Unified Modeling Language (UML) class diagrams with examples of their use.

**Appendix G: Multi-Source File Programs** A tutorial on how to create, compile, and link programs with multiple source files. Includes the use of function header files, class specification files, and class implementation files.

**Appendix H: Multiple and Virtual Inheritance** A self-contained discussion of the C++ concepts of multiple and virtual inheritance for anyone already familiar with single inheritance.

**Appendix I: Header File and Library Function Reference** A reference for the C++ library functions and header files used in the book.

**Appendix J: Namespaces** An explanation of namespaces and their purpose, with examples provided on how to define a namespace and access its members.





**Appendix K: C++ Casts and Run-Time Type Identification** An introduction to different ways of doing type casting in C++ and to run-time type identification.

**Appendix L: Passing Command Line Arguments** An introduction to writing C++ programs that accept command-line arguments. This appendix will be useful to students working in a command-line environment, such as UNIX or Linux.

**Appendix M: Binary Numbers and Bitwise Operations** A guide to the binary number system and the C++ bitwise operators, as well as a tutorial on the internal storage of integers.

**Appendix N: Introduction to Flowcharting** A tutorial that introduces flowcharting and its symbols. It includes handling sequence, selection, case, repetition, and calls to other modules. Sample flowcharts for several of the book's example programs are presented.

## Features of the Text

- Concept Statements** Each major section of the text starts with a concept statement. This statement summarizes the key idea of the section.
- Example Programs** The text has over 350 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.
- Tying It All Together** This special section, found at the end of every chapter, shows the student how to do something clever and fun with the material covered in that chapter.
-  **VideoNotes** A series of online videos, developed specifically for this book, are available for viewing at <http://www.pearsonhighered.com/gaddis/>. VideoNote icons appear throughout the text, alerting the student to videos about specific topics.
-  **Checkpoints** Checkpoints are questions placed throughout each chapter as a self-test study aid. Answers for all Checkpoint questions are provided in Appendix C at the back of the book so students can check how well they have learned a new topic.
-  **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 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, with complete code provided for each one. Additional case studies are provided on the book's companion website. 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. Each chapter ends with a *Soft Skills* exercise that focuses on communication and group process skills. Answers to the odd-numbered review questions and review exercises are provided in Appendix D at the back of the book.
- 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.

**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 teamwork within the classroom.

**C++ Quick Reference Guide**

For easy access, a quick reference guide to the C++ language is printed on the inside back cover.

**Supplements****Student Resources**

The following items are available on the Gaddis Series resource page at [www.pearsonhighered.com/gaddis](http://www.pearsonhighered.com/gaddis):

- Complete source code for every program included in the book
- Additional case studies, complete with source code
- Serendipity Booksellers ongoing software development project
- A full set of appendices (including several tutorials) that accompany the book
- Access to the book's companion VideoNotes
- Links to download numerous programming environments and IDEs, including MinGW C++ Compiler and wxDev-C++ IDE

**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
- A computerized test bank
- A collection of lab materials
- Source code files

Visit the Pearson Education Instructor Resource Center (<http://www.pearsonhighered.com/irc>) for information on how to access them.

**Practice and Assessment with MyProgrammingLab**

*MyProgrammingLab* helps students fully grasp the logic, semantics, and syntax of programming. Through practice exercises and immediate, personalized feedback, *MyProgrammingLab* improves the programming competence of beginning students who often struggle with the basic concepts and paradigms of popular high-level programming languages. A self-study and homework tool, a *MyProgrammingLab* course consists of hundreds of small practice exercises organized around the structure of this textbook. For students, the system automatically detects errors in the logic and syntax of their code submissions and offers targeted hints that enable them to figure out what went wrong. For instructors, a comprehensive gradebook tracks correct and incorrect answers and stores the code input by students for review.

*MyProgrammingLab* is offered to users of this book in partnership with Turing's Craft, the makers of the CodeLab interactive programming exercise system. For a full demonstration, to see feedback from instructors and students, or to get started using *MyProgrammingLab* in your course, visit [MyProgrammingLab.com](http://MyProgrammingLab.com).

### **Integrated Development Environment (IDE) Resource Kits**

Instructors who adopt this text for their students can also order an accompanying kit that contains the following popular C++ development environments:

- Microsoft<sup>®</sup> Visual Studio 2010 Express Edition
- Dev C++
- NetBeans
- Eclipse
- CodeLite

The kit also provides access to a website containing written and video tutorials for getting started in each IDE. For ordering information, please contact your Pearson Education Representative or visit [www.pearsonhighered.com/cs](http://www.pearsonhighered.com/cs).



## 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.

### Reviewers of the Eighth Edition or Its Previous Versions

Ahmad Abuhejleh <i>University of Wisconsin, River Falls</i>	Larry Farrer <i>Guilford Technical Community College</i>
David Akins <i>El Camino College</i>	Richard Flint <i>North Central College</i>
Steve Allan <i>Utah State University</i>	Sheila Foster <i>California State University Long Beach</i>
Ijaz A. Awan <i>Savannah State University</i>	David E. Fox <i>American River College</i>
John Bierbauer <i>North Central College</i>	Cindy Fry <i>Baylor University</i>
Don Biggerstaff <i>Fayetteville Technical Community College</i>	Peter Gacs <i>Boston University</i>
Paul Bladek <i>Spokane Falls Community College</i>	Cristi Gale <i>Sterling College</i>
Chuck Boehm <i>Dean Foods, Inc.</i>	James Gifford <i>University of Wisconsin, Stevens Point</i>
Bill Brown <i>Pikes Peak Community College</i>	Leon Gleiberman <i>Touro College</i>
Richard Cacace <i>Pensacola Junior College</i>	Simon Gray <i>Ashland University—Ohio</i>
Randy Campbell <i>Morningside College</i>	Margaret E. Guertin <i>Tufts University</i>
Stephen P. Carl <i>Wright State University</i>	Jamshid Haghighi <i>Guilford Technical Community College</i>
Wayne Caruolo <i>Red Rocks Community College</i>	Ranette H. Halverson <i>Midwestern State University, Wichita Falls, TX</i>
Thomas Cheatham <i>Middle Tennessee State University</i>	Dennis Heckman <i>Portland Community College</i>
James Chegwidden <i>Tarrant County College</i>	Ric Heishman <i>Northern Virginia Community College</i>
John Cigas <i>Rockhurst University</i>	Patricia Hines <i>Brookdale Community College</i>
John Cross <i>Indiana University of Pennsylvania</i>	Mike Holland <i>Northern Virginia Community College</i>
Fred M. D'Angelo <i>Pima Community College</i>	Lister Wayne Horn <i>Pensacola Junior College</i>
Joseph DeLibero <i>Arizona State University</i>	Richard Hull <i>Lenoir-Rhyne College</i>
Dennis Fairclough <i>Utah Valley State College</i>	Norman Jacobson <i>University of California, Irvine</i>

- Eric Jiang  
*San Diego State University*
- Yinping Jiao  
*South Texas College*
- Neven Jurkovic  
*Palo Alto College*
- David Kaeli  
*Northeastern University*
- Chris Kardaras  
*North Central College*
- Eugene Katzen  
*Montgomery College—Rockville*
- Willard Keeling  
*Blue Ridge Community College*
- A. J. Krygeris  
*Houston Community College*
- Ray Larson  
*Inver Hills Community College*
- Stephen Leach  
*Florida State University*
- Parkay Louie  
*Houston Community College*
- Zhu-qu Lu  
*University of Maine, Presque Isle*
- Tucjer Maney  
*George Mason University*
- Bill Martin  
*Central Piedmont Community College*
- Svetlana Marzelli  
*Atlantic Cape Community College*
- Debbie Mathews  
*J. Sargeant Reynolds*
- Ron McCarty  
*Penn State Erie, The Behrend College*
- Robert McDonald  
*East Stroudsburg University*
- James McGuffee  
*Austin Community College*
- M. Dee Medley  
*Augusta State University*
- Cathi Chambley-Miller  
*Aiken Technical College*
- Sandeep Mitra  
*SUNY Brockport*
- Frank Paiano  
*Southwestern Community College*
- Theresa Park  
*Texas State Technical College*
- Mark Parker  
*Shoreline Community College*
- Robert Plantz  
*Sonoma State University*
- Tino Posillico  
*SUNY Farmingdale*
- Mahmoud K. Quweider  
*University of Texas at Brownsville*
- M. Padmaja Rao  
*Francis Marion University*
- Timothy Reeves  
*San Juan College*
- Ronald Robison  
*Arkansas Tech University*
- Caroline St. Clair  
*North Central College*
- Dolly Samson  
*Weber State University*
- Kate Sanders  
*Rhode Island College*
- Lalchand Shimpi  
*Saint Augustine's College*
- Sung Shin  
*South Dakota State University*
- Barbara A. Smith  
*University of Dayton*
- Garth Sorenson  
*Snow College*
- Donald Southwell  
*Delta College*
- Daniel Spiegel  
*Kutztown University*
- Ray Springston  
*University of Texas at Arlington*
- Kirk Stephens  
*Southwestern Community College*
- Cherie Stevens  
*South Florida Community College*
- Joe Struss  
*Des Moines Area Community College*
- Hong Sung  
*University of Central Oklahoma*
- Sam Y. Sung  
*South Texas College*
- Mark Swanson  
*Red Wing Technical College*
- Martha Tillman  
*College of San Mateo*

Delores Tull  
*Itawamba Community College*

Rober Tureman  
*Paul D. Camp Community College*

Jane Turk  
*LaSalle University*

Sylvia Unwin  
*Bellevue Community College*

Stewart Venit  
*California State University, Los Angeles*

David Walter  
*Virginia State University*

Doug White  
*University of Northern Colorado*

Chris Wild  
*Old Dominion University*

Catherine Wyman  
*DeVry Institute of Technology, Phoenix*

Sherali Zeadally  
*University of the District of Columbia*

Chaim Ziegler  
*Brooklyn College*

The authors would like to thank their students at Haywood Community College and North Central College for inspiring them to write student-friendly books. They would also like to thank their families for their tremendous support throughout this project, as well as North Central College for providing Prof. Walters and Muganda with the sabbatical term during which they worked on this book. An especially big thanks goes to our terrific editorial, production, and marketing team at Addison-Wesley. In particular we want to thank our editor Matt Goldstein and our production project manager Kayla Smith-Tarbox, who have been instrumental in guiding the production of this book. We also want to thank our project manager, Mohinder Singh, who helped everything run smoothly, and our meticulous and knowledgeable copyeditor, Linthoingambi Khaidem, who dedicated many hours to making this book the best book it could be. You are great people to work with!

## About the Authors

Tony Gaddis is the principal author of the Starting Out With . . . series of textbooks. He is a highly acclaimed instructor with twenty years of experience teaching computer science courses at Haywood Community College. Tony 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 books covering C++, Java™, Microsoft® Visual Basic®, Microsoft® C#, Python, Programming Logic and Design, and Alice, all published by Pearson/Addison-Wesley.

Judy Walters is an Associate Professor of Computer Science at North Central College in Naperville, Illinois, where she teaches courses in both Computer Science and Interactive Media Studies. She is also very involved with International Programs at her college and has spent two semesters teaching in Costa Rica, where she hopes to retire some day.

Godfrey Muganda is an Associate Professor of Computer Science at North Central College. He teaches a wide variety of courses at both the undergraduate and graduate levels, including courses in Algorithms, Computer Organization, Web Applications, and Web Services. His primary research interests are in the area of Fuzzy Sets and Systems.

# Introduction to Computers and Programming

## TOPICS

- |                                             |                                               |
|---------------------------------------------|-----------------------------------------------|
| 1.1 Why Program?                            | 1.4 What Is a Program Made of?                |
| 1.2 Computer Systems: Hardware and Software | 1.5 Input, Processing, and Output             |
| 1.3 Programs and Programming Languages      | 1.6 The Programming Process                   |
|                                             | 1.7 Tying It All Together: <i>Hi! It's Me</i> |

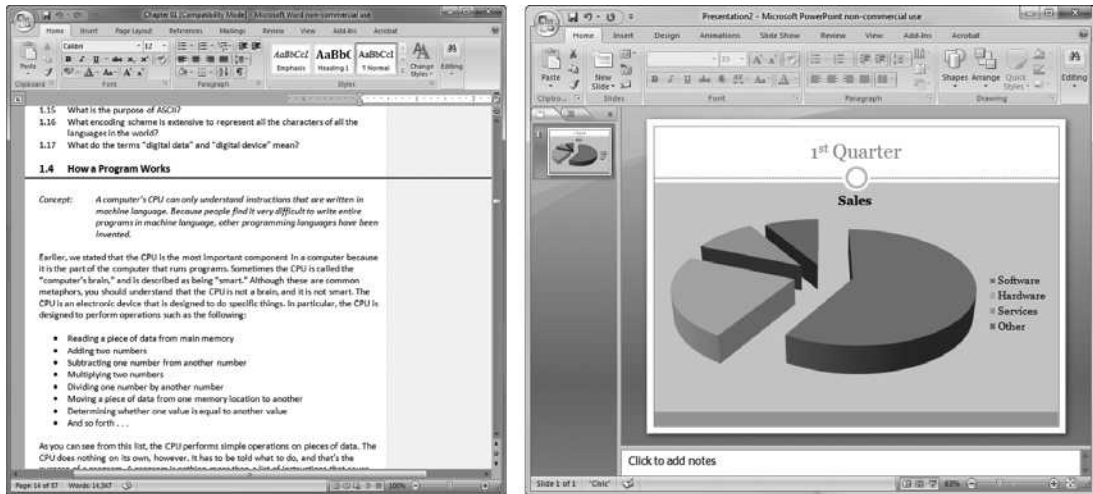
## 1.1

### 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 analyze data, make presentations, conduct business transactions, communicate with customers and coworkers, control machines in manufacturing facilities, and do many other things. At home, people use computers for tasks such as paying bills, shopping online, social networking, and playing games. And don't forget that smart phones, iPods<sup>®</sup>, 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 that computers are not designed to do just one job, but to do 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.

**Figure 1-1** A Word Processing Program and a Presentation Program

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 that 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, academics, entertainment, and almost every other field.

Computer programming is both an art and a science. It is an art because every aspect of a program should be designed with care and judgment. 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"
- Manuals and other forms of written documentation

There is also a scientific, or engineering side to programming. Because programs rarely work right the first time they are written, a lot of experimentation, correction, and redesigning is required. This demands patience and persistence of 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 makes 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.

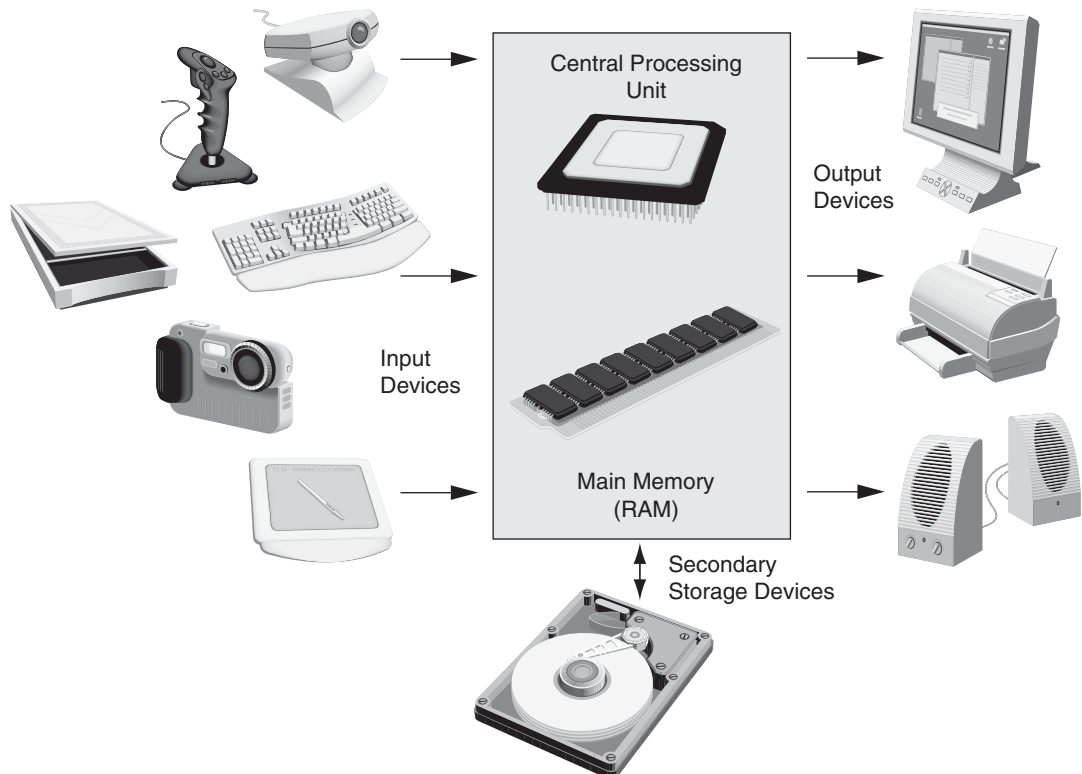
### Hardware

*Hardware* refers to the physical components that a computer is made of. 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:

1. The central processing unit (CPU)
2. Main memory (random-access memory, or RAM)
3. Secondary storage devices
4. Input devices
5. Output devices

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

**Figure 1-2**



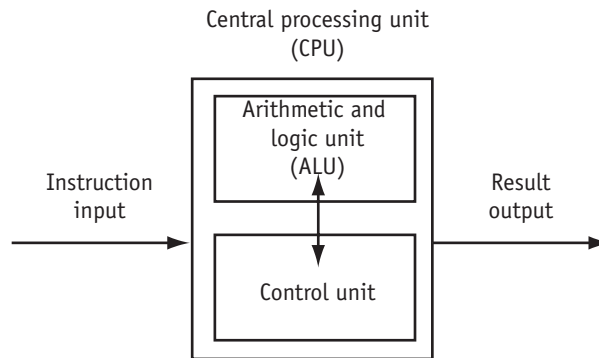
## The CPU

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 that weighed tons. They were made of electrical and mechanical components such as vacuum tubes and switches. Today, CPUs are small chips known as *microprocessors* that can be held in the palm of your hand. In addition to being much smaller than the old electromechanical CPUs in early computers, today's microprocessors are also much more powerful.

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 computer'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 1-3.

**Figure 1-3**



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 that the program is working with. 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 this memory. 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 cells known as *bytes*. One byte is enough memory to store just a single letter of the alphabet or a small number. In order to do anything meaningful, a computer has to have lots of bytes. Most computers today have millions, or even 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, so that the data stored there can be located. Figure 1-4 shows a group of memory cells with their addresses. The number 149 is stored in the cell with the address 16, and the number 72 is stored at address 23.

**Figure 1-4**

0	1	2	3	4	5	6	7	8	9	
10	11	12	13	14	15	16	149	17	18	19
20	21	22	23	72	24	25	26	27	28	29

## 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. Frequently used programs are stored in secondary memory and loaded into main memory as needed. Important information, such as word processing documents, payroll data, and inventory figures, is saved to secondary storage as well.

The most common type of secondary storage device is the disk drive. A *disk drive* stores data by magnetically encoding it onto a circular disk. Most computers have a disk drive mounted inside their case. External disk drives, which connect to one of the computer's communication ports, are also available. External disk drives can be used to create backup copies of important data or to move data to another computer.



In addition to external disk drives, many types of devices have been created for copying data and for moving it to other computers. For many years floppy disk drives were popular. A *floppy disk drive* records data onto a small, flexible (“floppy”) disk, which can be removed from the drive. The use of floppy disk drives has declined dramatically in recent years, in favor of superior devices such as USB flash drives. *USB flash drives* are small devices that plug into the computer’s USB (universal serial bus) port and appear to the system as a disk drive. These drives, which use *flash memory* to store data, are inexpensive, reliable, and small enough to be carried in your pocket.

Optical devices such as the *CD* (compact disc) and the *DVD* (digital versatile disc) are also popular for data storage. Data is not recorded magnetically on an optical disc, but rather is encoded as a series of pits on the disc surface. CD and DVD drives use a laser to detect the pits and thus read the encoded data. Optical discs hold large amounts of data, and because recordable CD and DVD drives are now commonplace, they are good media for creating backup copies of data.

### **Input Devices**

Input is any information 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, scanner, digital camera, and microphone. Disk drives, CD/DVD drives, and USB flash 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 computer screens, printers, and speakers. Output sent to a computer screen is sometimes called *soft copy*, while output sent to a printer is called *hard copy*. Disk drives, USB flash drives, and CD/DVD recorders can also be considered output devices because the CPU sends information to them so it can be saved.

## **Software**

If a computer is to function, software is needed. Everything that 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 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*, or *application programs*. 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 controls 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 might 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 ordered sequentially. Step 1 should be performed before step 2, and so forth. It is important that these instructions be performed in their proper sequence.

Although a person 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*. A machine language program consists of a sequence of *binary numbers* (numbers consisting of only 1s and 0s), which the CPU interprets as commands. Here is an example of what a machine language instruction might look like:

```
101101000000101
```

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* and then wanted to run it on a computer *B* that 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, and 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++.



**NOTE:** The line numbers 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
```

(program continues)

**Program 1-1** (continued)

```
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;
19
20 // Display the pay.
21 cout << "You have earned $" << pay << endl;
22 return 0;
23 }
```

**Program Output with Example Input Shown in Bold**

```
How many hours did you work? 10 [Enter]
How much do you get paid per hour? 15 [Enter]
You have earned $150
```

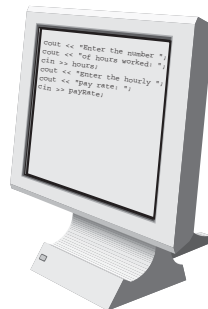
The “Program Output with Example Input Shown in Bold” shows what the program will display on the screen when it is running. In the example, the user enters 10 for the number of hours worked and 15 for the hourly pay. The program displays the earnings, which are \$150.

## Programming Languages

In a broad sense, there are two categories of programming languages: low-level and high-level. A *low-level language* is close to the level of the computer, which means it resembles the numeric machine language of the computer more than the natural language of humans. The easiest languages for people to learn are *high-level languages*. They are called “high-level” because they are closer to the level of human-readability than computer-readability. Figure 1-5 illustrates the concept of language levels.

**Figure 1-5**

High level (Easily read by humans)



Low level (machine language)  
10100010 11101011

