

starting out with >>>

# C++

From Control Structures  
through Objects

EIGHTH EDITION



TONY GADDIS

STARTING OUT WITH

**C++**

From Control Structures  
through Objects

**EIGHTH EDITION**

*This page intentionally left blank*

STARTING OUT WITH

**C++**

From Control Structures  
through Objects

**EIGHTH EDITION**

**Tony Gaddis**

Haywood Community College

**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 São Paulo Sydney Hong Kong Seoul Singapore Taipei Tokyo

Editorial Director: Marcia Horton  
Acquisitions Editor: Matt Goldstein  
Program Manager: Kayla Smith-Tarbox  
Director of Marketing: Christy Lesko  
Marketing Coordinator: Kathryn Ferranti  
Marketing Assistant: Jon Bryant  
Senior Managing Editor: Scott Disanno  
Senior Project Manager: Marilyn Lloyd  
Operations Supervisor: Vincent Scelta  
Operations Specialist: Linda Sager  
Art Director, Cover: Jayne Conte  
Text Designer: Joyce Cosentino Wells

Cover Designer: Bruce Kenselaar  
Manager, Visual Research: Karen Sanatar  
Permissions Supervisor: Michael Joyce  
Permission Administrator: Jenell Forschler  
Cover Image: Sergio37\_120/Fotolia  
Media Project Manager: Renata Butera  
Full-Service Project Manager: Jogender Taneja  
Aptara®, Inc.  
Full-Service Vendor: Aptara®, Inc.  
Printer/Binder: Courier Kendallville  
Cover Printer: Lehigh-Phoenix Color/Hagerstown

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

---

Copyright © 2015, 2012, 2009 Pearson Education, Inc., publishing as Addison-Wesley 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, One Lake Street, Upper Saddle River, New Jersey 07458 or you may fax your request to 201 236-3290.

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++ : from control structures through objects/Tony Gaddis.—Eighth edition.  
pages cm

Includes bibliographical references and index.

Online the following appendices are available at [www.pearsonhighered.com/gaddis](http://www.pearsonhighered.com/gaddis): Appendix D: Introduction to flowcharting; Appendix E: Using UML in class design; Appendix F: Namespaces; Appendix G: Writing managed C++ code for the .net framework; Appendix H: Passing command line arguments; Appendix I: Header file and library function reference; Appendix J: Binary numbers and bitwise operations; Appendix K: Multi-source file programs; Appendix L: Stream member functions for formatting; Appendix M: Introduction to Microsoft Visual C++ 2010 express edition; Appendix N: Answers to checkpoints; and Appendix O: Solutions to odd-numbered review questions.

ISBN-13: 978-0-13-376939-5

ISBN-10: 0-13-376939-9

1. C++ (Computer program language) I. Title. II. Title: From control structures through objects.

QA76.73.C153G33 2014b

005.13'3—dc23

2014000213

10 9 8 7 6 5 4 3 2 1

**PEARSON**

ISBN 13: 978-0-13-376939-5

ISBN 10: 0-13-376939-9

# Contents at a Glance

	<b>Preface</b>	<b>xv</b>
<b>CHAPTER 1</b>	<b>Introduction to Computers and Programming</b>	<b>1</b>
<b>CHAPTER 2</b>	<b>Introduction to C++</b>	<b>27</b>
<b>CHAPTER 3</b>	<b>Expressions and Interactivity</b>	<b>83</b>
<b>CHAPTER 4</b>	<b>Making Decisions</b>	<b>149</b>
<b>CHAPTER 5</b>	<b>Loops and Files</b>	<b>227</b>
<b>CHAPTER 6</b>	<b>Functions</b>	<b>299</b>
<b>CHAPTER 7</b>	<b>Arrays</b>	<b>375</b>
<b>CHAPTER 8</b>	<b>Searching and Sorting Arrays</b>	<b>457</b>
<b>CHAPTER 9</b>	<b>Pointers</b>	<b>495</b>
<b>CHAPTER 10</b>	<b>Characters, C-Strings, and More About the <code>string</code> Class</b>	<b>547</b>
<b>CHAPTER 11</b>	<b>Structured Data</b>	<b>599</b>
<b>CHAPTER 12</b>	<b>Advanced File Operations</b>	<b>657</b>
<b>CHAPTER 13</b>	<b>Introduction to Classes</b>	<b>711</b>
<b>CHAPTER 14</b>	<b>More About Classes</b>	<b>811</b>
<b>CHAPTER 15</b>	<b>Inheritance, Polymorphism, and Virtual Functions</b>	<b>891</b>
<b>CHAPTER 16</b>	<b>Exceptions, Templates, and the Standard Template Library (STL)</b>	<b>971</b>
<b>CHAPTER 17</b>	<b>Linked Lists</b>	<b>1025</b>
<b>CHAPTER 18</b>	<b>Stacks and Queues</b>	<b>1063</b>
<b>CHAPTER 19</b>	<b>Recursion</b>	<b>1121</b>
<b>CHAPTER 20</b>	<b>Binary Trees</b>	<b>1155</b>
	<b>Appendix A: Getting Started with Alice</b>	<b>1185</b>
	<b>Appendix B: The ASCII Character Set</b>	<b>1211</b>
	<b>Appendix C: Operator Precedence and Associativity</b>	<b>1213</b>
	<b>Quick References</b>	<b>1215</b>

**Index 1217**

**Credit 1237**

**Online** The following appendices are available at [www.pearsonhighered.com/gaddis](http://www.pearsonhighered.com/gaddis).

**Appendix D: Introduction to Flowcharting**

**Appendix E: Using UML in Class Design**

**Appendix F: Namespaces**

**Appendix G: Passing Command Line Arguments**

**Appendix H: Header File and Library Function Reference**

**Appendix I: Binary Numbers and Bitwise Operations**

**Appendix J: Multi-Source File Programs**

**Appendix K: Stream Member Functions for Formatting**

**Appendix L: Answers to Checkpoints**

**Appendix M: Solutions to Odd-Numbered Review Questions**

# Contents

## **Preface xv**

### **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

### **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 Variables and Literals 37
- 2.5 Identifiers 41
- 2.6 Integer Data Types 42
- 2.7 The `char` Data Type 48
- 2.8 The C++ `string` Class 52
- 2.9 Floating-Point Data Types 54
- 2.10 The `bool` Data Type 57
- 2.11 Determining the Size of a Data Type 58
- 2.12 Variable Assignments and Initialization 59
- 2.13 Scope 61
- 2.14 Arithmetic Operators 61
- 2.15 Comments 69
- 2.16 Named Constants 71
- 2.17 Programming Style 73



**CHAPTER 3 Expressions and Interactivity 83**

- 3.1 The `cin` Object 83
- 3.2 Mathematical Expressions 89
- 3.3 When You Mix Apples and Oranges: Type Conversion 98
- 3.4 Overflow and Underflow 100
- 3.5 Type Casting 101
- 3.6 Multiple Assignment and Combined Assignment 104
- 3.7 Formatting Output 108
- 3.8 Working with Characters and `string` Objects 118
- 3.9 More Mathematical Library Functions 124
- 3.10 Focus on Debugging: Hand Tracing a Program 130
- 3.11 Focus on Problem Solving: A Case Study 132

**CHAPTER 4 Making Decisions 149**

- 4.1 Relational Operators 149
- 4.2 The `if` Statement 154
- 4.3 Expanding the `if` Statement 162
- 4.4 The `if/else` Statement 166
- 4.5 Nested `if` Statements 169
- 4.6 The `if/else if` Statement 176
- 4.7 Flags 181
- 4.8 Logical Operators 182
- 4.9 Checking Numeric Ranges with Logical Operators 189
- 4.10 Menus 190
- 4.11 Focus on Software Engineering: Validating User Input 193
- 4.12 Comparing Characters and Strings 195
- 4.13 The Conditional Operator 199
- 4.14 The `switch` Statement 202
- 4.15 More About Blocks and Variable Scope 211

**CHAPTER 5 Loops and Files 227**

- 5.1 The Increment and Decrement Operators 227
- 5.2 Introduction to Loops: The `while` Loop 232
- 5.3 Using the `while` Loop for Input Validation 239
- 5.4 Counters 241
- 5.5 The `do-while` Loop 242
- 5.6 The `for` Loop 247
- 5.7 Keeping a Running Total 257
- 5.8 Sentinels 260
- 5.9 Focus on Software Engineering: Deciding Which Loop to Use 261
- 5.10 Nested Loops 262
- 5.11 Using Files for Data Storage 265
- 5.12 Optional Topics: Breaking and Continuing a Loop 284

**CHAPTER 6 Functions 299**

- 6.1 Focus on Software Engineering: Modular Programming 299
- 6.2 Defining and Calling Functions 300
- 6.3 Function Prototypes 309
- 6.4 Sending Data into a Function 311

6.5	Passing Data by Value	316
6.6	Focus on Software Engineering: Using Functions in a Menu-Driven Program	318
6.7	The <code>return</code> Statement	322
6.8	Returning a Value from a Function	324
6.9	Returning a Boolean Value	332
6.10	Local and Global Variables	334
6.11	Static Local Variables	342
6.12	Default Arguments	345
6.13	Using Reference Variables as Parameters	348
6.14	Overloading Functions	354
6.15	The <code>exit()</code> Function	358
6.16	Stubs and Drivers	361
<b>CHAPTER 7</b>	<b>Arrays</b>	<b>375</b>
7.1	Arrays Hold Multiple Values	375
7.2	Accessing Array Elements	377
7.3	No Bounds Checking in C++	384
7.4	Array Initialization	387
7.5	The Range-Based <code>for</code> Loop	392
7.6	Processing Array Contents	396
7.7	Focus on Software Engineering: Using Parallel Arrays	404
7.8	Arrays as Function Arguments	407
7.9	Two-Dimensional Arrays	418
7.10	Arrays with Three or More Dimensions	425
7.11	Focus on Problem Solving and Program Design: A Case Study	427
7.12	If You Plan to Continue in Computer Science: Introduction to the STL <code>vector</code>	429
<b>CHAPTER 8</b>	<b>Searching and Sorting Arrays</b>	<b>457</b>
8.1	Focus on Software Engineering: Introduction to Search Algorithms	457
8.2	Focus on Problem Solving and Program Design: A Case Study	463
8.3	Focus on Software Engineering: Introduction to Sorting Algorithms	470
8.4	Focus on Problem Solving and Program Design: A Case Study	477
8.5	If You Plan to Continue in Computer Science: Sorting and Searching vectors	485
<b>CHAPTER 9</b>	<b>Pointers</b>	<b>495</b>
9.1	Getting the Address of a Variable	495
9.2	Pointer Variables	497
9.3	The Relationship Between Arrays and Pointers	504
9.4	Pointer Arithmetic	508
9.5	Initializing Pointers	510
9.6	Comparing Pointers	511
9.7	Pointers as Function Parameters	513
9.8	Focus on Software Engineering: Dynamic Memory Allocation	522
9.9	Focus on Software Engineering: Returning Pointers from Functions	526
9.10	Using Smart Pointers to Avoid Memory Leaks	533
9.11	Focus on Problem Solving and Program Design: A Case Study	536

**CHAPTER 10 Characters, C-Strings, and More About the string Class 547**

- 10.1 Character Testing 547
- 10.2 Character Case Conversion 551
- 10.3 C-Strings 554
- 10.4 Library Functions for Working with C-Strings 558
- 10.5 C-String/Numeric Conversion Functions 569
- 10.6 Focus on Software Engineering: Writing Your Own C-String-Handling Functions 575
- 10.7 More About the C++ `string` Class 581
- 10.8 Focus on Problem Solving and Program Design: A Case Study 590

**CHAPTER 11 Structured Data 599**

- 11.1 Abstract Data Types 599
- 11.2 Focus on Software Engineering: Combining Data into Structures 601
- 11.3 Accessing Structure Members 604
- 11.4 Initializing a Structure 608
- 11.5 Arrays of Structures 611
- 11.6 Focus on Software Engineering: Nested Structures 613
- 11.7 Structures as Function Arguments 617
- 11.8 Returning a Structure from a Function 620
- 11.9 Pointers to Structures 623
- 11.10 Focus on Software Engineering: When to Use `.`, When to Use `->`, and When to Use `*` 626
- 11.11 Unions 628
- 11.12 Enumerated Data Types 632

**CHAPTER 12 Advanced File Operations 657**

- 12.1 File Operations 657
- 12.2 File Output Formatting 663
- 12.3 Passing File Stream Objects to Functions 665
- 12.4 More Detailed Error Testing 667
- 12.5 Member Functions for Reading and Writing Files 670
- 12.6 Focus on Software Engineering: Working with Multiple Files 678
- 12.7 Binary Files 680
- 12.8 Creating Records with Structures 685
- 12.9 Random-Access Files 689
- 12.10 Opening a File for Both Input and Output 697

**CHAPTER 13 Introduction to Classes 711**

- 13.1 Procedural and Object-Oriented Programming 711
- 13.2 Introduction to Classes 718
- 13.3 Defining an Instance of a Class 723
- 13.4 Why Have Private Members? 736
- 13.5 Focus on Software Engineering: Separating Class Specification from Implementation 737
- 13.6 Inline Member Functions 743
- 13.7 Constructors 746
- 13.8 Passing Arguments to Constructors 750

- 13.9 Destructors 758
- 13.10 Overloading Constructors 762
- 13.11 Private Member Functions 765
- 13.12 Arrays of Objects 767
- 13.13 Focus on Problem Solving and Program Design: An OOP Case Study 771
- 13.14 Focus on Object-Oriented Programming: Simulating Dice with Objects 778
- 13.15 Focus on Object-Oriented Programming: Creating an Abstract Array  
Data Type 782
- 13.16 Focus on Object-Oriented Design: The Unified Modeling Language (UML) 785
- 13.17 Focus on Object-Oriented Design: Finding the Classes and Their  
Responsibilities 788

## **CHAPTER 14 More About Classes 811**

- 14.1 Instance and Static Members 811
- 14.2 Friends of Classes 819
- 14.3 Memberwise Assignment 824
- 14.4 Copy Constructors 825
- 14.5 Operator Overloading 831
- 14.6 Object Conversion 858
- 14.7 Aggregation 860
- 14.8 Focus on Object-Oriented Design: Class Collaborations 865
- 14.9 Focus on Object-Oriented Programming: Simulating the Game  
of Cho-Han 869

## **CHAPTER 15 Inheritance, Polymorphism, and Virtual Functions 891**

- 15.1 What Is Inheritance? 891
- 15.2 Protected Members and Class Access 900
- 15.3 Constructors and Destructors in Base and Derived Classes 906
- 15.4 Redefining Base Class Functions 918
- 15.5 Class Hierarchies 923
- 15.6 Polymorphism and Virtual Member Functions 929
- 15.7 Abstract Base Classes and Pure Virtual Functions 945
- 15.8 Multiple Inheritance 952

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

- 16.1 Exceptions 971
- 16.2 Function Templates 990
- 16.3 Focus on Software Engineering: Where to Start When Defining Templates 996
- 16.4 Class Templates 996
- 16.5 Introduction to the Standard Template Library (STL) 1005

## **CHAPTER 17 Linked Lists 1025**

- 17.1 Introduction to the Linked List ADT 1025
- 17.2 Linked List Operations 1027
- 17.3 A Linked List Template 1043
- 17.4 Variations of the Linked List 1055
- 17.5 The STL list Container 1056

**CHAPTER 18 Stacks and Queues 1063**

- 18.1 Introduction to the Stack ADT 1063
- 18.2 Dynamic Stacks 1080
- 18.3 The STL `stack` Container 1091
- 18.4 Introduction to the Queue ADT 1093
- 18.5 Dynamic Queues 1105
- 18.6 The STL `deque` and `queue` Containers 1112

**CHAPTER 19 Recursion 1121**

- 19.1 Introduction to Recursion 1121
- 19.2 Solving Problems with Recursion 1125
- 19.3 Focus on Problem Solving and Program Design: The Recursive `gcd` Function 1133
- 19.4 Focus on Problem Solving and Program Design: Solving Recursively Defined Problems 1134
- 19.5 Focus on Problem Solving and Program Design: Recursive Linked List Operations 1135
- 19.6 Focus on Problem Solving and Program Design: A Recursive Binary Search Function 1139
- 19.7 The Towers of Hanoi 1141
- 19.8 Focus on Problem Solving and Program Design: The QuickSort Algorithm 1144
- 19.9 Exhaustive Algorithms 1148
- 19.10 Focus on Software Engineering: Recursion vs. Iteration 1151

**CHAPTER 20 Binary Trees 1155**

- 20.1 Definition and Applications of Binary Trees 1155
- 20.2 Binary Search Tree Operations 1158
- 20.3 Template Considerations for Binary Search Trees 1175

**Appendix A: Getting Started with Alice 1185**

**Appendix B: The ASCII Character Set 1211**

**Appendix C: Operator Precedence and Associativity 1213**

**Quick References 1215**

**Index 1217**

**Credit 1237**

**Online** The following appendices are available at [www.pearsonhighered.com/gaddis](http://www.pearsonhighered.com/gaddis).

**Appendix D: Introduction to Flowcharting**

**Appendix E: Using UML in Class Design**

**Appendix F: Namespaces**

**Appendix G: Passing Command Line Arguments**

**Appendix H: Header File and Library Function Reference**

**Appendix I: Binary Numbers and Bitwise Operations**

**Appendix J: Multi-Source File Programs**

**Appendix K: Stream Member Functions for Formatting**

**Appendix L: Answers to Checkpoints**

**Appendix M: Solutions to Odd-Numbered Review Questions**

## LOCATION OF VIDEONOTES IN THE TEXT



<b>Chapter 1</b>	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
<b>Chapter 2</b>	Using <code>cout</code> , p. 31 Variable Definitions, p. 37 Assignment Statements and Simple Math Expressions, p. 62 Solving the Restaurant Bill Problem, p. 80
<b>Chapter 3</b>	Reading Input with <code>cin</code> , p. 83 Formatting Numbers with <code>setprecision</code> , p. 111 Solving the Stadium Seating Problem, p. 142
<b>Chapter 4</b>	The <code>if</code> Statement, p. 154 The <code>if/else</code> statement, p. 166 The <code>if/else if</code> Statement, p. 176 Solving the Time Calculator Problem, p. 221
<b>Chapter 5</b>	The <code>while</code> Loop, p. 232 The <code>for</code> Loop, p. 247 Reading Data from a File, p. 274 Solving the Calories Burned Problem, p. 293
<b>Chapter 6</b>	Functions and Arguments, p. 311 Value-Returning Functions, p. 324 Solving the Markup Problem, p. 366
<b>Chapter 7</b>	Accessing Array Elements With a Loop, p. 380 Passing an Array to a Function, p. 407 Solving the Chips and Salsa Problem, p. 448
<b>Chapter 8</b>	The Binary Search, p. 460 The Selection Sort, p. 474 Solving the Charge Account Validation Modification Problem, p. 492
<b>Chapter 9</b>	Dynamically Allocating an Array, p. 523 Solving the Pointer Rewrite Problem, p. 545
<b>Chapter 10</b>	Writing a C-String-Handling Function, p. 575 More About the <code>string</code> Class, p. 581 Solving the Backward String Problem, p. 594

(continued on the next page)

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



<b>Chapter 11</b>	Creating a Structure, p. 601 Passing a Structure to a Function, p. 617 Solving the Weather Statistics Problem, p. 652
<b>Chapter 12</b>	Passing File Stream Objects to Functions, p. 665 Working with Multiple Files, p. 678 Solving the File Encryption Filter Problem, p. 708
<b>Chapter 13</b>	Writing a Class, p. 718 Defining an Instance of a Class, p. 723 Solving the <code>Employee</code> Class Problem, p. 802
<b>Chapter 14</b>	Operator Overloading, p. 831 Class Aggregation, p. 860 Solving the <code>NumDays</code> Problem, p. 885
<b>Chapter 15</b>	Redefining a Base Class Function in a Derived Class, p. 918 Polymorphism, p. 929 Solving the <code>Employee</code> and <code>Production-Worker</code> Classes Problem, p. 963
<b>Chapter 16</b>	Throwing an Exception, p. 972 Handling an Exception, p. 972 Writing a Function Template, p. 990 Storing Objects in a <code>vector</code> , p. 1010 Solving the Exception Project Problem, p. 1024
<b>Chapter 17</b>	Appending a Node to a Linked List, p. 1028 Inserting a Node in a Linked List, p. 1035 Deleting a Node from a Linked List, p. 1039 Solving the Member Insertion by Position Problem, p. 1061
<b>Chapter 18</b>	Storing Objects in an STL <code>stack</code> , p. 1091 Storing Objects in an STL <code>queue</code> , p. 1114 Solving the File Compare Problem, p. 1119
<b>Chapter 19</b>	Reducing a Problem with Recursion, p. 1126 Solving the Recursive Multiplication Problem, p. 1153
<b>Chapter 20</b>	Inserting a Node in a Binary Tree, p. 1160 Deleting a Node from a Binary Tree, p. 1166 Solving the Node Counter Problem, p. 1182

# Preface

Welcome to *Starting Out with C++: From Control Structures through Objects, 8th 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.

## Changes in the Eighth Edition

C++11 is the latest standard version of the C++ language. In previous years, while the standard was being developed, it was known as C++0x. In August 2011, it was approved by the International Standards Organization (ISO), and the name of the standard was officially changed to C++11. Most of the popular compilers now support the C++11 standard.

The new C++11 standard was the primary motivation behind this edition. Although this edition introduces many of the new language features, a C++11 compiler is not strictly required to use the book. As you progress through the book, you will see C++11 icons in the margins, next to the new features that are introduced. Programs appearing in sections that are not marked with this icon will still compile using an older compiler.

Here is a summary of the new C++11 topics that are introduced in this edition:

- The `auto` key word is introduced as a way to simplify complex variable definitions. The `auto` key word causes the compiler to infer a variable's data type from its initialization value.
- The `long long int` and `unsigned long long int` data types, and the `LL` literal suffix are introduced.
- Chapter 5 shows how to pass a `string` object directly to a file stream object's `open` member function, without the need to call the `c_str()` member function. (A discussion of the `c_str()` function still exists for anyone using a legacy compiler.)



- The range-based `for` loop is introduced in Chapter 7. This new looping mechanism automatically iterates over each element of an array, `vector`, or other collection, without the need of a counter variable or a subscript.
- Chapter 7 shows how a `vector` can be initialized with an initialization list.
- The `nullptr` key word is introduced as the standard way of representing a null pointer.
- Smart pointers are introduced in Chapter 9, with an example of dynamic memory allocation using `unique_ptr`.
- Chapter 10 discusses the new, overloaded `to_string` functions for converting numeric values to `string` objects.
- The `string` class's new `back()` and `front()` member functions are included in Chapter 10's overview of the `string` class.
- Strongly typed enums are discussed in Chapter 11.
- Chapter 13 shows how to use the smart pointer `unique_ptr` to dynamically allocate an object.
- Chapter 15 discusses the `override` key word and demonstrates how it can help prevent subtle overriding errors. The `final` key word is discussed as a way of preventing a virtual member function from being overridden.

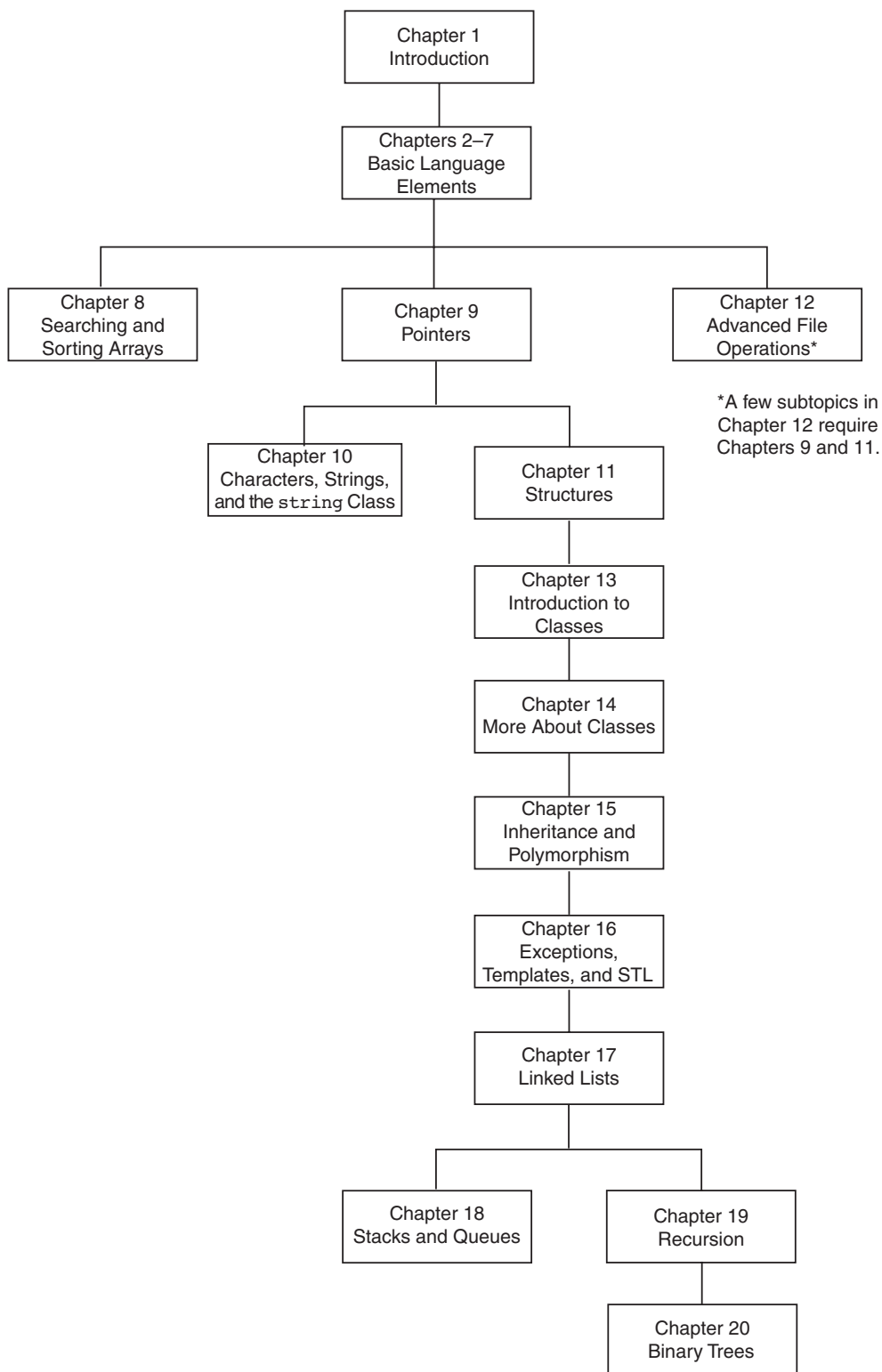
In addition to the C++11 topics, the following general improvements were made:

- Several new programming problems have been added to the text, and many of the existing programming problems have been modified to make them unique from previous editions.
- The discussion of early, historic computers in Chapter 1 is expanded.
- The discussion of literal values in Chapter 2 is improved.
- The introduction of the `char` data type in Chapter 2 is reorganized to use character literals in variable assignments before using ASCII values in variable assignments.
- The discussion of random numbers in Chapter 3 is expanded and improved, with the addition of a new *In the Spotlight* section.
- A new *Focus on Object-Oriented Programming* section has been added to Chapter 13, showing how to write a class that simulates dice.
- A new *Focus on Object-Oriented Programming* section has been added to Chapter 14, showing an object-oriented program that simulates the game of Cho-Han. The program uses objects for the dealer, two players, and a pair of dice.

## 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.

Figure P-1



Chapter 1 covers fundamental hardware, software, and programming concepts. You may choose to skip this chapter if the class has already mastered 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.

After Chapter 7 has been covered, you may proceed to Chapter 8, or jump to either Chapter 9 or Chapter 12. (If you jump to Chapter 12 at this point, you will need to postpone sections 12.7, 12.8, and 12.10 until Chapters 9 and 11 have been covered.)

After Chapter 9 has been covered, either of Chapters 10 or 11 may be covered. After Chapter 11, you may cover Chapters 13 through 17 in sequence. Next you can proceed to either Chapter 18 or Chapter 19. Finally, Chapter 20 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. An optional section explains the difference between ANSI standard and pre-standard C++ programs.

### **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**

In this chapter the student learns to create and work with single and multidimensional 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: Sorting and Searching 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.

## **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, Templates, and the Standard Template Library (STL)**

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. Finally, the student is introduced to the containers, iterators, and algorithms offered by the Standard Template Library (STL).

## **Chapter 17: 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 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.

## **Chapter 19: 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.

## **Chapter 20: 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: Getting Started with Alice**

This appendix gives a quick introduction to Alice. Alice is free software that can be used to teach fundamental programming concepts using 3D graphics.

## **Appendix B: ASCII Character Set**

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

## **Appendix C: Operator Precedence and Associativity**

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

The following appendices are available online at [www.pearsonhighered.com/gaddis](http://www.pearsonhighered.com/gaddis).

## **Appendix D: Introduction to Flowcharting**

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

## **Appendix E: 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 F: Namespaces**

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

## **Appendix G: 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 H: Header File and Library Function Reference**

This appendix provides a reference for the C++ library functions and header files discussed in the book.

## **Appendix I: Binary Numbers and Bitwise Operations**

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

### **Appendix J: 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 K: Stream Member Functions for Formatting**

Covers stream member functions for formatting such as `setf`.

### **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: Solutions 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** A series of online videos, developed specifically for this book, is available for viewing at [www.pearsonhighered.com/gaddis](http://www.pearsonhighered.com/gaddis). 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 can be downloaded from the book's Companion Web site at [www.pearsonhighered.com/gaddis](http://www.pearsonhighered.com/gaddis). This allows students to 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 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/gaddis](http://www.pearsonhighered.com/gaddis).

**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.

**Software Development Project: Serendipity Booksellers**

Available for download from the book's Companion Web site at [www.pearsonhighered.com/gaddis](http://www.pearsonhighered.com/gaddis). This is an ongoing project that instructors can optionally assign to teams of students. It systematically develops a "real-world" software package: a point-of-sale program for the fictitious Serendipity Booksellers organization. The Serendipity assignment for each chapter adds more functionality to the software, using constructs and techniques covered in that chapter. When complete, the program will act as a cash register, manage an inventory database, and produce a variety of reports.

**C++ Quick Reference Guide**

For easy access, a quick reference guide to the C++ language is printed on the last two pages of Appendix C in the book.

**C++11**

Throughout the text, new C++11 language features are introduced. Look for the C++11 icon to find these new features.



## 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/gaddis](http://www.pearsonhighered.com/gaddis):

- The source code for each example program in the book
- Access to the book's companion 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
- The complete Serendipity Booksellers Project

### Integrated Development Environment (IDE) Resource Kits

Professors who adopt this text can order it for students with a kit containing five popular C++ IDEs (Microsoft® Visual Studio Express Edition, Dev C++, NetBeans, Eclipse, and CodeLite) and access to a Web site containing written and video tutorials for getting started in each IDE. For ordering information, please contact your campus Pearson Education representative or visit [www.pearsonhighered.com/cs](http://www.pearsonhighered.com/cs).

### Online 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 students to figure out what went wrong—and why. For instructors, a comprehensive gradebook tracks correct and incorrect answers and stores the code inputted 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 [www.myprogramminglab.com](http://www.myprogramminglab.com).

### 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 Pearson Instructor Resource Center ([www.pearsonhighered.com/irc](http://www.pearsonhighered.com/irc)) for information on how to access instructor resources.

### Textbook Web site

Student and instructor resources, including links to download Microsoft® Visual Studio Express and other popular IDEs, for all the books in the Gaddis *Starting Out With* series can be accessed at the following URL:

<http://www.pearsonhighered.com/gaddis>

### Get this book the way you want it!

This book is part of Pearson Education’s custom database for Computer Science textbooks. Use our online PubSelect system to select just the chapters you need from this, and other, Pearson Education CS textbooks. You can edit the sequence to exactly match your course organization and teaching approach. Visit [www.pearsoncustom.com/cs](http://www.pearsoncustom.com/cs) for details.

### 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.

<b>■ FROM CONTROL STRUCTURES THROUGH OBJECTS</b> <b>■ BRIEF VERSION</b>	<b>■ EARLY OBJECTS</b>
<p><b>LATE INTRODUCTION OF OBJECTS</b> Classes are introduced in Chapter 13 of the standard text and Chapter 11 of 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.</p> <p><b>INTRODUCTION OF DATA STRUCTURES AND RECURSION</b> 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.</p>	<p><b>EARLIER INTRODUCTION OF OBJECTS</b> 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.</p> <p><b>INTRODUCTION OF DATA STRUCTURES AND RECURSION</b> Linked lists, stacks and queues, and binary trees are introduced in the final chapters of the text, after the chapter on recursion.</p>

## 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 for the 8th Edition

Robert Burn  
*Diablo Valley College*

Michael Dixon  
*Sacramento City College*

Qiang Duan  
*Penn State University—Abington*

Daniel Edwards  
*Oklahoma College*

Xisheng Fang  
*Oklahoma College*

Ken Hang  
*Green River Community College*

Kay Johnson  
*Community College of Rhode Island*

Michelle Levine  
*Broward College*

Cindy Lindstrom  
*Lakeland College*

Susan Reeder  
*Seattle University*

Sandra Roberts  
*Snead College*

Lopa Roychoudhuri  
*Angelo State University*

Richard Snyder  
*Lehigh Carbon Community College*

Donald Southwell  
*Delta College*

Chadd Williams  
*Pacific University*

### Reviewers for Previous Editions

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*

Charles Cadenhead  
*Richland Community College*

Randall Campbell  
*Morningside College*

Wayne Caruolo  
*Red Rocks Community College*

Cathi Chambley-Miller  
*Aiken Technical College*

C.C. Chao  
*Jacksonville State University*

Joseph Chao  
*Bowling Green State University*

Royce Curtis  
*Western Wisconsin Technical College*

Joseph DeLiberio  
*Arizona State University*

Jeanne Douglas  
*University of Vermont*

Michael Dowell  
*Augusta State U*

William E. Duncan  
*Louisiana State University*

Judy Etchison  
*Southern Methodist University*

Dennis Fairclough  
*Utah Valley State 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*

Ranette Halverson, Ph.D.  
*Midwestern State University*

Carol Hannahs  
*University of Kentucky*

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*

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*

Jennifer Li  
*Ohlone College*

Norman H. Liebling  
*San Jacinto College*

Zhu-qu Lu  
*University of Maine, Presque Isle*

Heidar Malki  
*University of Houston*

Debbie Mathews  
*J. Sargeant Reynolds Community College*

Rick Matzen  
*Northeastern State University*

Robert McDonald  
*East Stroudsburg University*

James McGuffee  
*Austin Community College*

- 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*
- 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*
- 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*
- 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*
- Vida Winans  
*Illinois Institute of Technology*

I would like to thank my family for their love and support in all of my many projects. I am extremely fortunate to have Matt Goldstein as my editor. I am also fortunate to have Kathryn Ferranti as marketing coordinator. She does a great job getting my books out to the academic community. I had a great production team led by Marilyn Lloyd and Kayla Smith-Tarbox. Thanks to you all!

## **About the Author**

Tony Gaddis is the principal author of the *Starting Out with* series of textbooks. He has nearly 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.

**BREAKTHROUGH**  
To improving results



# 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)

# 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 Procedural and Object-Oriented Programming |

## 1.1

### Why Program?

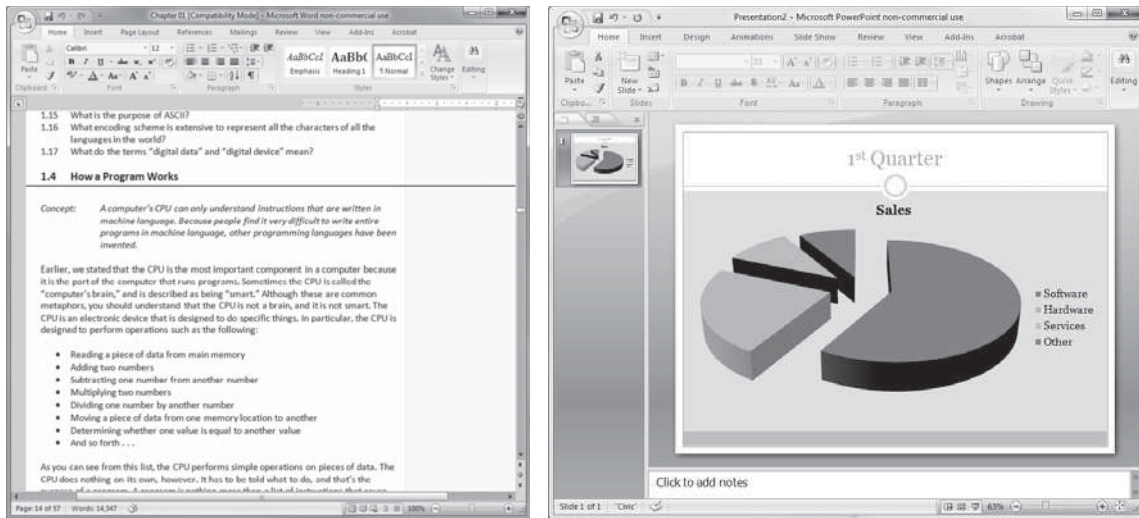
**CONCEPT:** Computers can do many different jobs because they are programmable.

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 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 computer games. And don't forget that smart phones, iPods®, 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 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 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.



**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”
- 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 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.

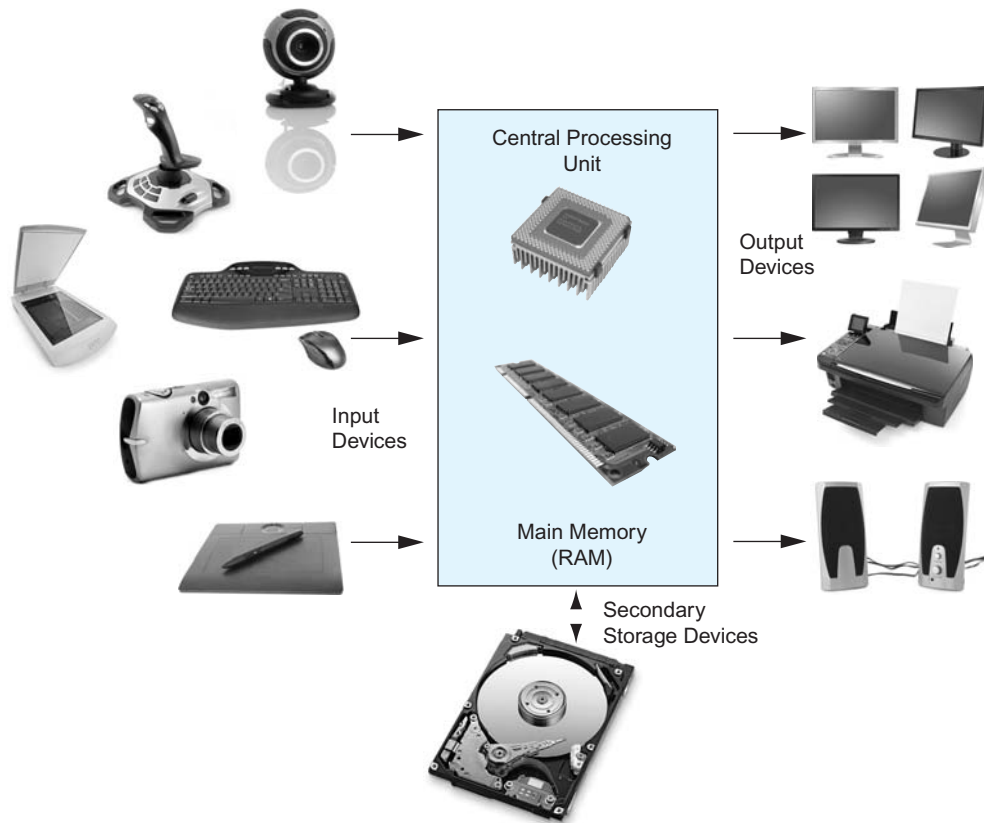
## Hardware

*Hardware* refers to the physical components 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:

- 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**



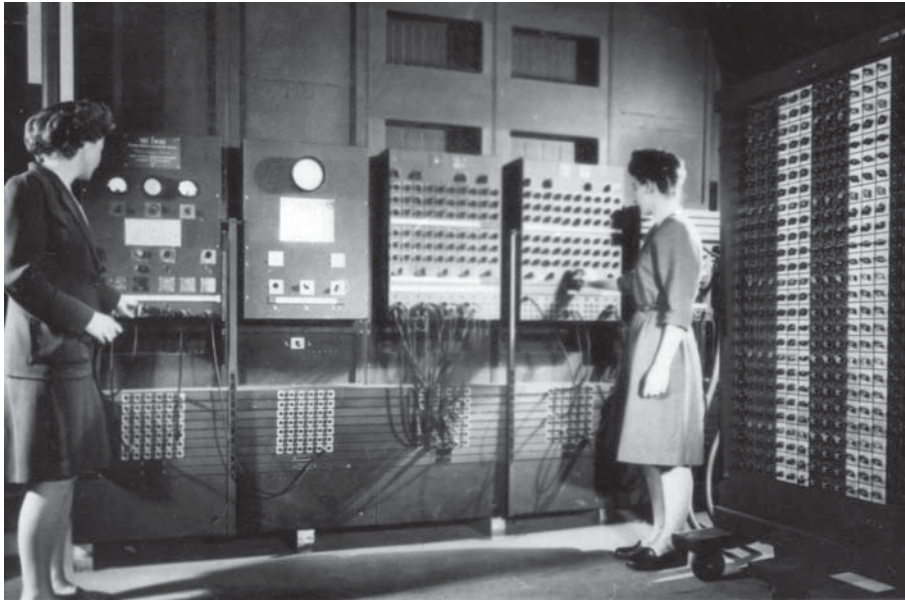
### The CPU

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**

---

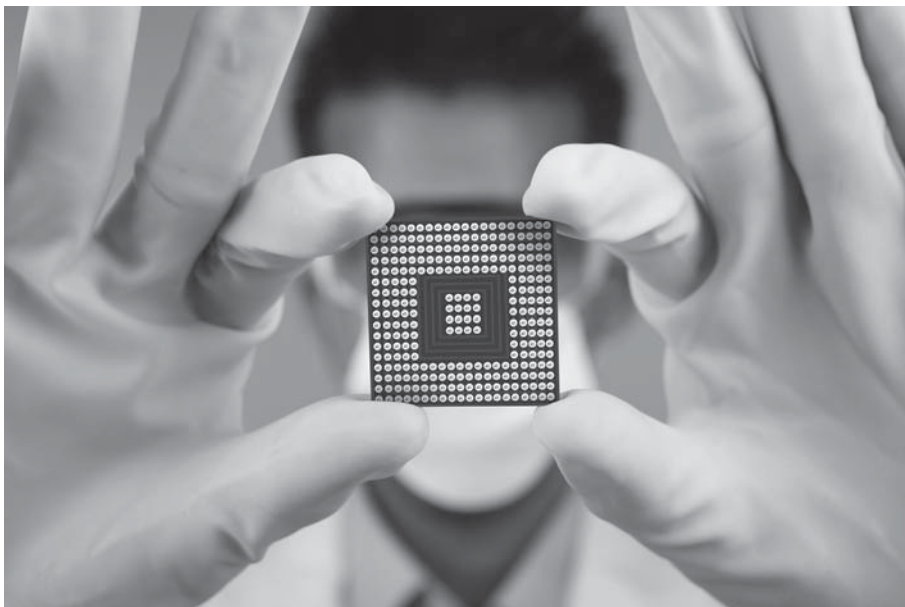


the photo are working with the historic ENIAC computer. The *ENIAC*, considered by many to be the world's first programmable electronic computer, was built in 1945 to calculate artillery ballistic tables for the U.S. Army. This machine, which was primarily one big CPU, was 8 feet tall, 100 feet long, 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 electro-mechanical CPUs in early computers, microprocessors are also much more powerful.

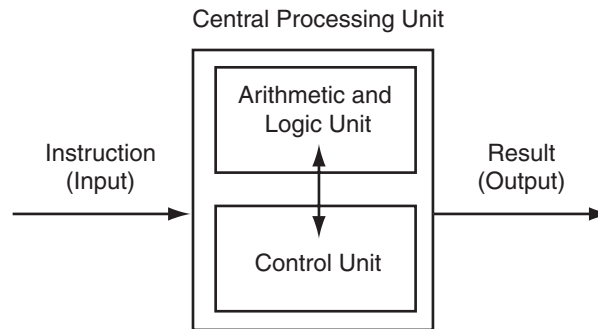
**Figure 1-4**

---



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 1-5.

**Figure 1-5**



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:

<i>Fetch</i>	The CPU's control unit fetches, from main memory, the next instruction in the sequence of program instructions.
<i>Decode</i>	The instruction is encoded in the form of a number. The control unit decodes the instruction and generates an electronic signal.
<i>Execute</i>	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 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 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. 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**

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 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 drives. *USB drives* are small devices that plug into the computer's USB (universal serial bus) port and appear to the system as a disk drive. USB 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 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 mediums 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 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 monitors, printers, and speakers. Disk drives, USB drives, and CD/DVD recorders can also be considered output devices because the CPU sends them information to be saved.

## Software

If a computer is to function, software is not optional. 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*. 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* and 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, 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++.

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)