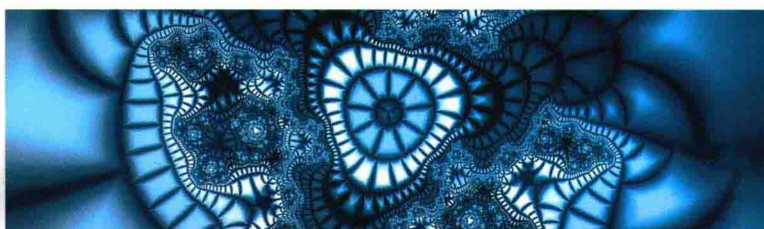


PEARSON

国外信息科学与技术经典图书系列

# Data Structures and Other Objects Using C++

(Fourth Edition)



# 数据结构

## ——C++版(第四版)

Michael Main 著  
Walter Savitch

(英文影印版)



科学出版社

PEARSON

国外信息科学与技术经典图书系:

# 数据结构——C++版 (英文影印版)

(第四版)

Data Structures and Other Objects  
Using C++  
(Fourth Edition)

Michael Main   Walter Savitch 著

科学出版社

北 京

图字: 01-2012-4732

## 内 容 简 介

本书是一本基于 C++ 的思想、介绍数据结构和算法的大学教材, 已经在全球多个国家的大学用作数据结构课程的基础教材。本书以 C++ 语言作为实现语言, 利用面向对象的方法, 从规格说明出发, 使用基础的数据类型来描述程序算法的设计与实现。书中主要内容包括: 软件开发的各个阶段, 抽象数据类型与 C++ 类, 容器类, 指针与动态数组, 链表, 用模板、迭代器和 STL 进行软件开发, 堆栈, 队列, 递归思想, 树, 平衡树, 查找, 排序, 派生类与继承, 图表。通过学习本书, 可使读者具备使用数据类型的能力, 学会利用多种方法来实现数据类型, 以及从不同的实现中进行取舍。

本书可作为计算机、电类专业本科生和非信息技术专业硕士研究生的教材, 也可供工程技术人员参考。

Original edition, entitled Data Structures and Other Objects Using C++, 4E, 978-0-13-212948-0 by Main, Michael; Savitch, Water, published by Pearson Education, Inc, publishing as Pearson, Copyright © 2011 by Pearson Education, Inc.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from Pearson Education, Inc.

China edition published by PEARSON EDUCATION ASIA LTD., and CHINA SCIENCE PUBLISHING & MEDIA LTD. (SCIENCE PRESS) Copyright © 2012

Authorized for sale and distribution in the People's Republic of China exclusively (except Taiwan, Hong Kong SAR and Macau SAR). 本版本仅可在中国地区(除台湾、香港与澳门)销售与发行。

本书封面贴有 Pearson Education(培生教育出版集团)激光防伪标签。无标签者不得销售。

### 图书在版编目(CIP)数据

数据结构: C++ 版 = Data Structures and Other Objects Using C++: 4 版: 英文/(美) 梅因(Main, M.), (美) 萨维特奇(Savitch, W.) 著. —影印本. —北京: 科学出版社, 2012  
(国外信息科学与技术经典图书系列)

ISBN 978-7-03-035024-4

I. ①数… II. ①梅… ②萨… III. ①数据结构-C 英文②C 语言-程序设计-英文 IV. ①TP311.12  
②312

中国版本图书馆 CIP 数据核字(2011)第 133346 号

责任编辑: 潘斯斯 匡 敏 / 责任印制: 闫 磊 / 封面设计: 迷底书装

科学出版社出版

北京东黄城根北街16号

邮政编码: 100717

<http://www.sciencep.com>

保定市中画美凯印刷有限公司印刷

科学出版社发行 各地新华书店经销

\*

2012年6月第 一 版 开本: 787×1092 1/16

2012年6月第一次印刷 印张: 53

字数: 1160 000

定价: 108.00 元

(如有印装质量问题, 我社负责调换)

## Chapter List

CHAPTER 1	THE PHASES OF SOFTWARE DEVELOPMENT	31
CHAPTER 2	ABSTRACT DATA TYPES AND C++ CLASSES	63
CHAPTER 3	CONTAINER CLASSES	126
CHAPTER 4	POINTERS AND DYNAMIC ARRAYS	184
CHAPTER 5	LINKED LISTS	250
CHAPTER 6	SOFTWARE DEVELOPMENT WITH TEMPLATES, ITERATORS, AND THE STL	320
CHAPTER 7	STACKS	382
CHAPTER 8	QUEUES	423
CHAPTER 9	RECURSIVE THINKING	466
CHAPTER 10	TREES	504
CHAPTER 11	BALANCED TREES	569
CHAPTER 12	SEARCHING	613
CHAPTER 13	SORTING	659
CHAPTER 14	DERIVED CLASSES AND INHERITANCE	713
CHAPTER 15	GRAPHS	762
APPENDIXES		811
INDEX		841

# Contents

## CHAPTER 1 THE PHASES OF SOFTWARE DEVELOPMENT

- 1.1 Specification, Design, Implementation 33
  - Design Concept: Decomposing the Problem 34
  - Preconditions and Postconditions 36
  - Using Functions Provided by Other Programmers 38
  - Implementation Issues for the ANSI/ISO C++ Standard 38
  - C++ *Feature*: The Standard Library and the Standard Namespace 39
  - Programming Tip*: Use Declared Constants 41
  - Clarifying the Const Keyword*
    - Part 1: Declared Constants 42
  - Programming Tip*: Use Assert to Check a Precondition 42
  - Programming Tip*: Use EXIT\_SUCCESS in a Main Program 44
  - C++ *Feature*: Exception Handling 44
  - Self-Test Exercises for Section 1.1 44
- 1.2 Running Time Analysis 45
  - The Stair-Counting Problem 45
  - Big-O Notation 51
  - Time Analysis of C++ Functions 53
  - Worst-Case, Average-Case, and Best-Case Analyses 55
  - Self-Test Exercises for Section 1.2 55
- 1.3 Testing and Debugging 56
  - Choosing Test Data 56
  - Boundary Values 57
  - Fully Exercising Code 58
  - Debugging 58
  - Programming Tip*: How to Debug 58
  - Self-Test Exercises for Section 1.3 59
- Chapter Summary 60
- Solutions to Self-Test Exercises 61

## CHAPTER 2 ABSTRACT DATA TYPES AND C++ CLASSES

- 2.1 Classes and Members 64
  - Programming Example*: The Throttle Class 64
  - Clarifying the Const Keyword*
    - Part 2: Constant Member Functions 68
  - Using a Class 69
  - A Small Demonstration Program for the Throttle Class 70
  - Implementing Member Functions 72
  - Member Functions May Activate Other Members 74
  - Programming Tip*: Style for Boolean Variables 74
  - Self-Test Exercises for Section 2.1 75

## 16 Contents

### 2.2 Constructors 75

The Throttle's Constructor 76  
What Happens If You Write a Class with No Constructors? 79  
*Programming Tip: Always Provide Constructors* 79  
Revising the Throttle's Member Functions 79  
Inline Member Functions 79  
*Programming Tip: When to Use an Inline Member Function* 80  
Self-Test Exercises for Section 2.2 81

### 2.3 Using a Namespace, Header File, and Implementation File 81

Creating a Namespace 81  
The Header File 82  
Describing the Value Semantics of a Class Within the Header File 86  
*Programming Tip: Document the Value Semantics* 87  
The Implementation File 87  
Using the Items in a Namespace 89  
*Pitfall: Never Put a Using Statement Actually in a Header File* 90  
Self-Test Exercises for Section 2.3 92

### 2.4 Classes and Parameters 93

*Programming Example: The Point Class* 93  
Default Arguments 95  
*Programming Tip: A Default Constructor Can Be Provided by Using Default Arguments* 96  
Parameters 97  
*Pitfall: Using a Wrong Argument Type for a Reference Parameter* 100  
*Clarifying the Const Keyword*  
Part 3: Const Reference Parameters 102  
*Programming Tip: Use const Consistently* 103  
When the Type of a Function's Return Value Is a Class 103  
Self-Test Exercises for Section 2.4 104

### 2.5 Operator Overloading 104

Overloading Binary Comparison Operators 105  
Overloading Binary Arithmetic Operators 106  
Overloading Output and Input Operators 107  
Friend Functions 110  
*Programming Tip: When to Use a Friend Function* 111  
The Point Class—Putting Things Together 112  
Summary of Operator Overloading 115  
Self-Test Exercises for Section 2.5 115

### 2.6 The Standard Template Library and the Pair Class 116

Chapter Summary 117

Solutions to Self-Test Exercises 118

Programming Projects 120

## CHAPTER 3 CONTAINER CLASSES

3.1	The Bag Class	127
	The Bag Class—Specification	128
	C++ Feature: Typedef Statements Within a Class Definition	129
	C++ Feature: The <code>std::size_t</code> Data Type	130
	Clarifying the <i>Const</i> Keyword	
	Part 4: Static Member Constants	134
	Older Compilers Do Not Support Initialization of Static Member Constants	135
	The Bag Class—Documentation	135
	Documenting the Value Semantics	137
	The Bag Class—Demonstration Program	137
	The Bag Class—Design	139
	Pitfall: The <code>value_type</code> Must Have a Default Constructor	140
	The Invariant of a Class	140
	The Bag Class—Implementation	141
	Pitfall: Needing to Use the Full Type Name <code>bag::size_type</code>	142
	Programming Tip: Make Assertions Meaningful	142
	C++ Feature: The Copy Function from the C++ Standard Library	146
	The Bag Class—Putting the Pieces Together	147
	Programming Tip: Document the Class Invariant in the Implementation File	147
	The Bag Class—Testing	151
	Pitfall: An Object Can Be an Argument to Its Own Member Function	151
	The Bag Class—Analysis	152
	Self-Test Exercises for Section 3.1	153
3.2	Programming Project: The Sequence Class	154
	The Sequence Class—Specification	154
	The Sequence Class—Documentation	157
	The Sequence Class—Design	157
	The Sequence Class—Pseudocode for the Implementation	160
	Self-Test Exercises for Section 3.2	162
3.3	Interactive Test Programs	163
	C++ Feature: Converting Input to Uppercase Letters	164
	C++ Feature: The Switch Statement	168
	Self-Test Exercises for Section 3.3	168
3.4	The STL Multiset Class and Its Iterator	169
	The Multiset Template Class	169
	Some Multiset Members	170
	Iterators and the [...] Pattern	170
	Pitfall: Do Not Access an Iterator's Item After Reaching <code>end()</code>	172
	Testing Iterators for Equality	173
	Other Multiset Operations	173
	Invalid Iterators	174
	Clarifying the <i>Const</i> Keyword	
	Part 5: Const Iterators	174
	Pitfall: Changing a Container Object Can Invalidate Its Iterators	174
	Self-Test Exercises for Section 3.4	175
	Chapter Summary	176
	Solutions to Self-Test Exercises	176
	Programming Projects	179

**CHAPTER 4 POINTERS AND DYNAMIC ARRAYS**

- 4.1 Pointers and Dynamic Memory 185
  - Pointer Variables 186
  - Using the Assignment Operator with Pointers 188
  - Dynamic Variables and the new Operator 189
  - Using new to Allocate Dynamic Arrays 190
  - The Heap and the bad\_alloc Exception 193
  - The delete Operator 193
  - Programming Tip:* Define Pointer Types 194
  - Self-Test Exercises for Section 4.1 195
- 4.2 Pointers and Arrays as Parameters 196
  - Clarifying the Const Keyword*
  - Part 6: Const Parameters That Are Pointers or Arrays 201
  - Self-Test Exercises for Section 4.2 203
- 4.3 The Bag Class with a Dynamic Array 206
  - Pointer Member Variables 206
  - Member Functions Allocate Dynamic Memory as Needed 207
  - Programming Tip:* Provide Documentation about Possible Dynamic Memory Failure 211
  - Value Semantics 211
  - The Destructor 214
  - The Revised Bag Class—Class Definition 215
  - The Revised Bag Class—Implementation 217
  - Programming Tip:* How to Check for Self-Assignment 218
  - Programming Tip:* How to Allocate Memory in a Member Function 221
  - The Revised Bag Class—Putting the Pieces Together 222
  - Self-Test Exercises for Section 4.3 224
- 4.4 Prescription for a Dynamic Class 225
  - Four Rules 225
  - Special Importance of the Copy Constructor 225
  - Pitfall:* Using Dynamic Memory Requires a Destructor, a Copy Constructor, and an Overloaded Assignment Operator 226
  - Self-Test Exercises for Section 4.4 227



4.5	The STL String Class and a Project	227
	Null-Terminated Strings	227
	Initializing a String Variable	228
	The Empty String	228
	Reading and Writing String Variables	229
	<i>Pitfall</i> : Using = and == with Strings	229
	The strcpy Function	229
	The strcat Function	230
	<i>Pitfall</i> : Dangers of strcpy, strcat, and Reading Strings	230
	The strlen Function	231
	The strcmp Function	231
	The String Class—Specification	231
	Constructor for the String Class	233
	Overloading the Operator []	234
	Some Further Overloading	234
	Other Operations for the String Class	235
	The String Class—Design	235
	The String Class—Implementation	236
	Demonstration Program for the String Class	238
	Chaining the Output Operator	240
	Declaring Constant Objects	240
	Constructor-Generated Conversions	240
	Using Overloaded Operations in Expressions	241
	Our String Class Versus the C++ Library String Class	241
	Self-Test Exercises for Section 4.5	241
4.6	Programming Project: The Polynomial	242
	Chapter Summary	246
	Solutions to Self-Test Exercises	246
	Programming Projects	248

## CHAPTER 5 LINKED LISTS

5.1	A Fundamental Node Class for Linked Lists	251
	Declaring a Class for Nodes	251
	Using a Typedef Statement with Linked-List Nodes	252
	Head Pointers, Tail Pointers	253
	The Null Pointer	254
	The Meaning of a Null Head Pointer or Tail Pointer	254
	The Node Constructor	254
	The Node Member Functions	255
	The Member Selection Operator	256
	<i>Clarifying the Const Keyword</i>	
	Part 7: The Const Keyword with a Pointer to a Node, and the Need for Two Versions of Some Member Functions	257
	<i>Programming Tip</i> : A Rule for a Node's Constant Member Functions	258
	<i>Pitfall</i> : Dereferencing the Null Pointer	260
	Self-Test Exercises for Section 5.1	260

5.2	A Linked-List Toolkit	261
	Linked-List Toolkit—Header File	262
	Computing the Length of a Linked List	262
	<i>Programming Tip:</i> How to Traverse a Linked List	265
	<i>Pitfall:</i> Forgetting to Test the Empty List	266
	Parameters for Linked Lists	266
	Inserting a New Node at the Head of a Linked List	268
	Inserting a New Node That Is Not at the Head	270
	<i>Pitfall:</i> Unintended Calls to delete and new	273
	Searching for an Item in a Linked List	275
	Finding a Node by Its Position in a Linked List	276
	Copying a Linked List	277
	Removing a Node at the Head of a Linked List	280
	Removing a Node That Is Not at the Head	281
	Clearing a Linked List	282
	Linked-List Toolkit—Putting the Pieces Together	283
	Using the Linked-List Toolkit	284
	Self-Test Exercises for Section 5.2	288
5.3	The Bag Class with a Linked List	289
	Our Third Bag—Specification	289
	Our Third Bag—Class Definition	289
	How to Make the Bag value_type Match the Node value_type	290
	Following the Rules for Dynamic Memory Usage in a Class	293
	The Third Bag Class—Implementation	294
	<i>Pitfall:</i> The Assignment Operator Causes Trouble with Linked Lists	295
	<i>Programming Tip:</i> How to Choose Between Approaches	297
	The Third Bag Class—Putting the Pieces Together	301
	Self-Test Exercises for Section 5.3	302
5.4	Programming Project: The Sequence Class with a Linked List	305
	The Revised Sequence Class—Design Suggestions	305
	The Revised Sequence Class—Value Semantics	306
	Self-Test Exercises for Section 5.4	307
5.5	Dynamic Arrays vs. Linked Lists vs. Doubly Linked Lists	307
	Making the Decision	309
	Self-Test Exercises for Section 5.5	309
5.6	STL Vectors vs. STL Lists vs. STL Deques	310
	Self-Test Exercises for Section 5.6	312
	Chapter Summary	313
	Solutions to Self-Test Exercises	313
	Programming Projects	317

## CHAPTER 6 SOFTWARE DEVELOPMENT WITH TEMPLATES, ITERATORS, AND THE STL

- 6.1 Template Functions 321
  - Syntax for a Template Function 323
  - Programming Tip:* Capitalize the Name of a Template Parameter 323
  - Using a Template Function 324
  - Pitfall:* Failed Unification Errors 324
  - A Template Function to Swap Two Values 326
  - Programming Tip:* Swap, Max, and Min Functions 326
  - Parameter Matching for Template Functions 326
  - A Template Function to Find the Biggest Item in an Array 327
  - Pitfall:* Mismatches for Template Function Arguments 329
  - A Template Function to Insert an Item into a Sorted Array 329
  - Self-Test Exercises for Section 6.1 331
- 6.2 Template Classes 331
  - Syntax for a Template Class 331
  - Programming Tip:* Use the Name Item and the typename Keyword 333
  - Pitfall:* Do Not Place Using Directives in a Template Implementation 334
  - More About the Template Implementation File 334
  - Parameter Matching for Member Functions of Template Classes 339
  - Using the Template Class 339
  - Details of the Story-Writing Program 342
  - Self-Test Exercises for Section 6.2 342
- 6.3 The STL's Algorithms and Use of Iterators 343
  - STL Algorithms 343
  - Standard Categories of Iterators 344
  - Iterators for Arrays 346
  - Self-Test Exercises for Section 6.3 347
- 6.4 The Node Template Class 347
  - Functions That Return a Reference Type 348
  - What Happens When a Reference Return Value Is Copied Elsewhere 350
  - The Data Member Function Now Requires Two Versions 350
  - Header and Implementation Files for the New Node 351
  - Self-Test Exercises for Section 6.4 351
- 6.5 An Iterator for Linked Lists 358
  - The Node Iterator 358
  - The Node Iterator Is Derived from std::iterator 360
  - Pitfall:* std::iterator Might Not Exist 361
  - The Node Iterator's Private Member Variable 361
  - Node Iterator—Constructor 361
  - Node Iterator—the \* Operator 361
  - Node Iterator—Two Versions of the ++ Operator 362
  - Programming Tip:* ++p Is More Efficient Than p++ 364
  - Iterators for Constant Collections 364
  - Programming Tip:* When to Use a Const Iterator 366
  - Self-Test Exercises for Section 6.5 366

6.6	Linked-List Version of the Bag Template Class with an Iterator	367
	How to Provide an Iterator for a Container Class That You Write	367
	The Bag Iterator	368
	Why the Iterator Is Defined Inside the Bag	369
	Self-Test Exercises for Section 6.6	369
	Chapter Summary and Summary of the Five Bags	377
	Solutions to Self-Test Exercises	378
	Programming Projects	380

## CHAPTER 7 STACKS

7.1	The STL Stack Class	383
	The Standard Library Stack Class	384
	<i>Programming Example:</i> Reversing a Word	385
	Self-Test Exercises for Section 7.1	386
7.2	Stack Applications	387
	<i>Programming Example:</i> Balanced Parentheses	387
	<i>Programming Example:</i> Evaluating Arithmetic Expressions	389
	Evaluating Arithmetic Expressions—Specification	389
	Evaluating Arithmetic Expressions—Design	390
	Evaluating Arithmetic Expressions—Implementation	396
	Functions Used in the Calculator Program	397
	Evaluating Arithmetic Expressions—Testing and Analysis	397
	Evaluating Arithmetic Expressions—Enhancements	398
	Self-Test Exercises for Section 7.2	398
7.3	Implementations of the Stack Class	399
	Array Implementation of a Stack	399
	Linked-List Implementation of a Stack	403
	The Koenig Lookup	404
	Self-Test Exercises for Section 7.3	404
7.4	More Complex Stack Applications	407
	Evaluating Postfix Expressions	407
	Translating Infix to Postfix Notation	409
	Using Precedence Rules in the Infix Expression	411
	Correctness of the Conversion from Infix to Postfix	413
	Self-Test Exercises for Section 7.4	417
	Chapter Summary	417
	Solutions to Self-Test Exercises	417
	Programming Projects	419

## CHAPTER 8 QUEUES

8.1	The STL Queue	424
	The Standard Library Queue Class	425
	Uses for Queues	425
	Self-Test Exercises for Section 8.1	427

8.2	Queue Applications	428
	<i>Programming Example: Recognizing Palindromes</i>	428
	Self-Test Exercises for Middle of Section 8.2	430
	<i>Programming Example: Car Wash Simulation</i>	431
	Car Wash Simulation—Specification	431
	Car Wash Simulation—Design	432
	Car Wash Simulation—Implementing the Car Wash Classes	435
	Car Wash Simulation—Implementing the Simulation Function	440
	Self-Test Exercises for End of Section 8.2	441
8.3	Implementations of the Queue Class	443
	Array Implementation of a Queue	443
	<i>Programming Tip: Use Small Helper Functions to Improve Clarity</i>	446
	Discussion of the Circular Array Implementation of a Queue	448
	Linked-List Implementation of a Queue	450
	Implementation Details	451
	<i>Programming Tip: Make Note of “Don’t Care” Situations</i>	453
	<i>Pitfall: Which End Is Which</i>	453
	Self-Test Exercises for Section 8.3	456
8.4	Implementing the STL Deque Class	456
	Calling the Destructor and Constructor for the Deque’s value_type Items	459
	Other Variations on Stacks and Queues	460
	Self-Test Exercises for Section 8.4	460
8.5	Reference Return Values for the Stack, Queue, and Other Classes	460
	Chapter Summary	460
	Solutions to Self-Test Exercises	462
	Programming Projects	463

## CHAPTER 9 RECURSIVE THINKING

9.1	Recursive Functions	467
	A First Example of Recursive Thinking	467
	Tracing Recursive Calls	469
	<i>Programming Example: An Extension of write_vertical</i>	471
	A Closer Look at Recursion	472
	General Form of a Successful Recursive Function	475
	Self-Test Exercises for Section 9.1	476
9.2	Studies of Recursion: Fractals and Mazes	477
	<i>Programming Example: Generating Random Fractals</i>	477
	A Function for Generating Random Fractals—Specification	478
	Design and Implementation of the Fractal Function	480
	How the Random Fractals Are Displayed	481
	<i>Programming Example: Traversing a Maze</i>	483
	Traversing a Maze—Specification	483
	Traversing a Maze—Design	485
	Traversing a Maze—Implementation	486
	The Recursive Pattern of Exhaustive Search with Backtracking	488
	<i>Programming Example: The Teddy Bear Game</i>	489
	<i>Pitfall: Forgetting to Use the Return Value from a Recursive Call</i>	489
	Self-Test Exercises for Section 9.2	490

9.3	Reasoning About Recursion	491
	How to Ensure That There Is No Infinite Recursion	493
	Inductive Reasoning About the Correctness of a Recursive Function	496
	Self-Test Exercises for Section 9.3	497
	Chapter Summary	498
	Solutions to Self-Test Exercises	498
	Programming Projects	500

## CHAPTER 10 TREES

10.1	Introduction to Trees	505
	Binary Trees	505
	Binary Taxonomy Trees	508
	General Trees	509
	Self-Test Exercises for Section 10.1	510
10.2	Tree Representations	510
	Array Representation of Complete Binary Trees	510
	Representing a Binary Tree with a Class for Nodes	513
	Self-Test Exercises for Section 10.2	515
10.3	Binary Tree Nodes	515
	<i>Pitfall:</i> Not Connecting All the Links	518
	<i>Programming Example:</i> Animal Guessing	519
	Animal Guessing Program—Design and Implementation	521
	Animal Guessing Program—Improvements	526
	Self-Test Exercises for Section 10.3	530
10.4	Tree Traversals	530
	Traversals of Binary Trees	530
	Printing the Data from a Tree's Node	535
	The Problem with Our Traversals	536
	A Parameter Can Be a Function	537
	A Template Version of the Apply Function	539
	More Generality for the Apply Template Function	540
	Template Functions for Tree Traversals	541
	Self-Test Exercises for Section 10.4	542
10.5	Binary Search Trees	548
	The Binary Search Tree Storage Rules	548
	Our Sixth Bag—Class Definition	552
	Our Sixth Bag—Implementation of Some Simple Functions	552
	Counting the Occurrences of an Item in a Binary Search Tree	553
	Inserting a New Item into a Binary Search Tree	554
	Removing an Item from a Binary Search Tree	555
	The Union Operators for Binary Search Trees	559
	Time Analysis and an Iterator	561
	Self-Test Exercises for Section 10.5	561
	Chapter Summary	561
	Solutions to Self-Test Exercises	562
	Programming Projects	564

## CHAPTER 11 BALANCED TREES

11.1	Heaps	570
	The Heap Storage Rules	570
	The Priority Queue ADT with Heaps	571
	Adding an Entry to a Heap	572
	Removing an Entry from a Heap	573
11.2	The STL Priority Queue and Heap Algorithms	576
	Self-Test Exercises for Sections 11.1 and 11.2	577
11.3	B-Trees	577
	The Problem of Unbalanced Trees	577
	The B-Tree Rules	578
	An Example B-Tree	579
	The Set ADT with B-Trees	580
	Searching for an Item in a B-Tree	585
	Inserting an Item into a B-Tree	587
	The Loose Insertion into a B-Tree	587
	A Private Member Function to Fix an Excess in a Child	590
	Back to the Insert Member Function	591
	Employing Top-Down Design	593
	Removing an Item from a B-Tree	593
	The Loose Erase from a B-Tree	594
	A Private Member Function to Fix a Shortage in a Child	596
	Removing the Biggest Item from a B-Tree	599
	<i>Programming Tip:</i> Write and Test Small Pieces	599
	<i>Programming Tip:</i> Consider Using the STL Vector	600
	External B-Trees	600
	Self-Test Exercises for Section 11.2	601
11.4	Trees, Logs, and Time Analysis	602
	Time Analysis for Binary Search Trees	603
	Time Analysis for Heaps	603
	Logarithms	605
	Logarithmic Algorithms	606
	Self-Test Exercises for Section 11.3	607
11.5	The STL Map and Multimap Classes	607
	Map and Multimap Implementations	608
	Chapter Summary	609
	Solutions to Self-Test Exercises	609
	Programming Projects	612

## CHAPTER 12 SEARCHING

12.1	Serial Search and Binary Search	614
	Serial Search	614
	Serial Search—Analysis	614
	Binary Search	616
	Binary Search—Design	617
	<i>Pitfall:</i> Common Indexing Errors in Binary Search Implementations	619
	Binary Search—Analysis	620
	Standard Library Search Functions	624
	Functions for Sorted Ranges	624
	Functions for Unsorted Ranges	626
	The STL search Function	626
	Self-Test Exercises for Section 12.1	628
12.2	Open-Address Hashing	628
	Introduction to Hashing	628
	The Table Class—Specification	631
	The Table Class—Design	633
	<i>Programming Tip:</i> Using <code>size_t</code> Can Indicate a Value's Purpose	636
	The Table ADT—Implementation	636
	C++ <i>Feature:</i> Inline Functions in the Implementation File	642
	Choosing a Hash Function to Reduce Collisions	642
	Double Hashing to Reduce Clustering	643
	Self-Test Exercises for Section 12.2	644
12.3	Chained Hashing	645
	Self-Test Exercises for Section 12.3	647
12.4	Time Analysis of Hashing	647
	The Load Factor of a Hash Table	647
	Self-Test Exercises for Section 12.4	650
12.5	Programming Project: A Table Class with STL Vectors	650
	A New Table Class	650
	Using Vectors in the New Table	651
	Template Parameters That Are Constants	651
	Template Parameters That Are Functions	651
	Implementing the New Table Class	652
	Self-Test Exercises for Section 12.5	653
12.6	Hash Tables in the TR1 Library Extensions	654
	Chapter Summary	654
	Solutions to Self-Test Exercises	655
	Programming Projects	658



## CHAPTER 13 SORTING

- 13.1 Quadratic Sorting Algorithms 660
  - Selectionsort—Specification 660
  - Selectionsort—Design 660
  - Selectionsort—Implementation 662
  - Selectionsort—Analysis 664
  - Programming Tip: Rough Estimates Suffice for Big-O* 666
  - Insertionsort 666
  - Insertionsort—Analysis 670
  - Self-Test Exercises for Section 13.1 672
- 13.2 Recursive Sorting Algorithms 672
  - Divide-and-Conquer Using Recursion 672
  - C++ Feature: Specifying a Subarray with Pointer Arithmetic* 673
  - Mergesort 675
  - The merge Function 676
  - Dynamic Memory Usage in Mergesort 681
  - Mergesort—Analysis 681
  - Mergesort for Files 683
  - Quicksort 683
  - The partition Function 685
  - Quicksort—Analysis 689
  - Quicksort—Choosing a Good Pivot Element 691
  - Self-Test Exercises for Section 13.2 691
- 13.3 An  $O(n \log n)$  Algorithm Using a Heap 692
  - Heapsort 692
  - Making the Heap 697
  - Reheapification Downward 700
  - Heapsort—Analysis 701
  - Self-Test Exercises for Section 13.3 702
- 13.4 Sorting and Binary Search in the STL 702
  - The Original C qsort Function 702
  - The STL sort Function 703
  - Heapsort in the STL 704
  - Binary Search Functions in the STL 704
  - The Comparison Parameter for STL Sorting Functions 705
  - Writing Your Own sort Function That Uses Iterators 706
- Chapter Summary 707
- Solutions to Self-Test Exercises 708
- Programming Projects 709

## CHAPTER 14 DERIVED CLASSES AND INHERITANCE

- 14.1 Derived Classes 714
  - How to Declare a Derived Class 716
  - The Automatic Constructors of a Derived Class 717
  - Using a Derived Class 718
  - The Automatic Assignment Operator for a Derived Class 720
  - The Automatic Destructor of a Derived Class 720
  - Overriding Inherited Member Functions 721
  - Programming Tip: Make the Overriding Function Call the Original* 722
  - Self-Test Exercises for Section 14.1 722