

DATA STRUCTURES

A Pseudocode
Approach
with C++



Richard F. Gilberg • Behrouz A. Forouzan

TP311.112
W 2

Data Structures

A Pseudocode Approach with C++

Richard F. Gilberg

De Anza College

Behrouz A. Forouzan

De Anza College

江苏工业学院图书馆
藏书章



Brooks/Cole

Thomson Learning™

Australia • Canada • Mexico • Singapore • Spain
United Kingdom • United States

Sponsoring Editor: *Kallie Swanson*
Marketing Team: *Samantha Cabaluna and Christina De Veto*
Editorial Assistant: *Grace Fujimoto*
Production Coordinator: *Mary Vezilich*
Production Service: *Carlisle Publishers Services*
Print Buyer: *Vena Dyer*

Cover Designer: *Christine Garrigan*
Design Coordinator: *Roy Neuhaus*
Typesetting: *Carlisle Communications, Ltd.*
Printing and Binding: *R.R. Donnelley/Crawfordsville*

COPYRIGHT © 2001 by Brooks/Cole
A division of Thomson Learning
The Thomson Learning logo is a trademark used herein under license.

For more information about this or any other Brooks/Cole product, contact:

BROOKS/COLE
511 Forest Lodge Road
Pacific Grove, CA 93950 USA
www.brookscole.com
1-800-423-0563 (Thomson Learning Academic Resource Center)

All rights reserved. No part of this work may be reproduced, transcribed or used in any form or by any means—graphic, electronic, or mechanical, including photocopying, recording, taping, Web distribution, or information storage and/or retrieval systems—without the prior written permission of the publisher.

For permission to use material from this work, contact us by

Web: www.thomsonrights.com
fax: 1-800-730-2215
phone: 1-800-730-2214

Printed in the United States of America

10 9 8 7 6 5 4 3 2

Library of Congress Cataloging-in-Publication Data

Gilberg, Richard F.

Data structures: a pseudocode approach with C++/Richard F. Gilberg, Behrouz A. Forouzan.

Richard F. Gilberg.

p. cm.

Includes index.

ISBN 0-534-95216-X

1. C++ (Computer program language) 2. Data structures (Computer science). I.

Forouzan, Behrouz A. II. Title.

QA76.73.C153 G545 2001
005.13'3—dc21

00-025353
CIP

In memory of my mother, Ann

R. F. Gilberg

To my nephew, Ryan Cameron Kioumeh

B. A. Forouzan

Preface

The study of data structures is both exciting and challenging. It is exciting because it presents a wide range of programming techniques that make it possible to solve larger and more complex problems. It is challenging because the complex nature of data structures brings with it many concepts that change the way we approach the design of programs.

Because the study of data structures encompasses an abundant amount of material, you will find that it is not possible to cover all of it in one term. In fact, data structures is such a pervasive subject that you will find it taught in lower-division, upper-division, and graduate programs.

Our primary focus in this text is to present data structures as an introductory subject, taught in a lower-division course. With this focus in mind, we present the material in a simple, straightforward manner with many examples and figures. We also deemphasize the mathematical aspect of data structures, leaving the formal mathematical proofs of the algorithms for later courses.

Pseudocode is an English-like presentation of the steps needed to solve a problem. It is written with a relaxed syntax that allows students to solve a problem at a level that hides the detail while they concentrate on the problem requirements. In other words, it allows students to concentrate on the big picture.

In addition to being an excellent design tool, pseudocode is also language independent. Consequently, students can use the same pseudocode design to implement an algorithm in several different languages. We developed our pseudocode syntax in our data structures classes over a 15-year period. During that time, our students have implemented the pseudocode algorithms in Pascal, C, and C++. In this text, we use C++ for all of our code implementations.

As we discuss the various data structures, we first present the general principles using diagrams to help the student visualize the concept. If the data structure is large and complex enough to require several algorithms, we use a structure chart to present a design solution. Once the design and structure are fully understood, we present a pseudocode algorithm, followed as appropriate by its C++ implementation.

Abstract Data Types

The second major feature of this text is its use of abstract data types (ADTs) implemented as C++ classes. To make ADTs data independent, we use template classes. All ADTs accept either one (data) or two (data and key) arguments. In this way any data type, including derived types and structures, can be used with all ADTs. Conversely, each ADT can be used with any data type as long as the required operators are pre-defined for that type. We introduce the concept immediately in Chapter 1 and use it extensively throughout the text.

Not every data structure should be implemented as an ADT class. However, where appropriate, we develop a complete C++ implementation for the student's study and use. Specifically, students will find ADT class implementations for Lists (Chapter 3), Stacks (Chapter 4), Queues (Chapter 5), AVL Trees (Chapter 8), B-Trees (Chapter 10), and Graphs (Chapter 12). The code for all of the ADTs is available on the Instructor's Materials page at the Brooks/Cole Web site www.brookscole.com

Structure and Style

One of our basic educational tenets is that good habits are formed early. The corollary is that bad habits are hard to break. Therefore, we consistently emphasize the principles of structured programming and software engineering. Every algorithm and program in the book uses a consistent style. As the algorithms and programs are analyzed, style and standards are further explained. While we acknowledge that there are many good styles, our experience has shown that if students are exposed to a good style and implement it, they will be better able to adapt to other good styles. On the other hand, unlearning sloppy short-cut habits is very difficult.

Visual Approach

A brief scan of the book will demonstrate that our approach is primarily visual. There are over 345 figures, 35 tables, 140 algorithms, 180 programs, and numerous code examples. Although this amount of material tends to create a large book, these materials make it much easier for students to follow the concepts.

Pedagogical End Materials

End of chapter materials reinforce what the student has learned. The important topics in the chapter are summarized in bulleted lists. Following the summary are three practice sets.

Exercises are multiple choice and short answer questions covering the material in the chapter. The answers to the odd numbered questions are included in the back of the book.

Problems are short assignments that ask the student to develop a pseudocode algorithm or write a short program to be run on a computer. These problems can usually be developed in 2 to 3 hours. The instructor's manual contains complete solutions for all exercises and problems.

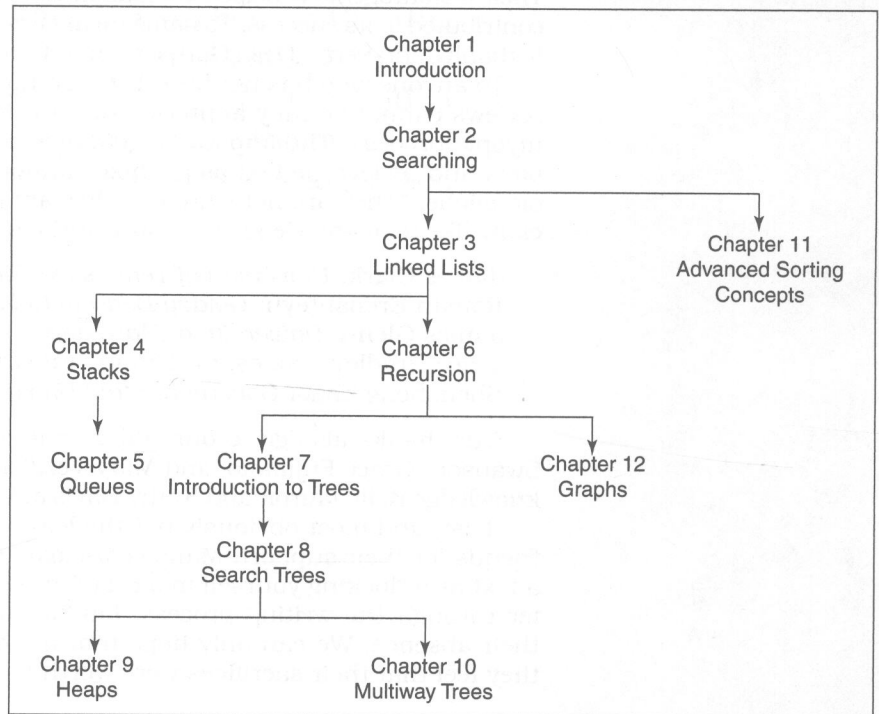
Projects are longer, major assignments that may take an average student 6 to 9 hours or more to develop.

Organization And Order Of Topics

We have tried to build flexibility into the text so that the material may be covered in the order that best suits the needs of a particular class. Although we use the materials in the order presented in the text, there are other possible sequences (shown in the figure on this page). We recommend that you assign Chapter 1 as general reading. It contains basic information on pseudocode, abstract data types, and algorithmics students will need for the rest of the text.

The first two sections of Chapter 2 review sequential and binary search concepts. The third section, hashed list searches, may be new material. If you have covered search algorithms in your programming class, you may save this chapter for later. On the other hand, if your students have not studied searching algorithms, then you will need to cover at least the first section. Many of the algorithms in the following chapters require an understanding of sequential and ordered list searching. In many texts, sorting is covered with searching. Because our sorting chapter includes the recursive implementation of quick sort and heap sort (which requires an understanding of trees and heaps), we place it at the end of the text. With the exception of these two sorts, however, it could be covered before Chapter 3.

Chapter 3 introduces linear lists and the basic linked list data structures. It also introduces the first complete ADT class. For these reasons, Chapter 3 should be covered before the remaining chapters in the text.



Possible subject sequences

The stack concept (Chapter 4) is basic to an understanding of recursion (Chapter 6), and recursion is in turn required to understand trees (Chapters 7, 8, and 10) and heaps (Chapter 9). Likewise, queues (Chapter 5) are used in breadth-first traversals in Chapters 7 and 12.

Chapter 9, Heaps, is a stand-alone chapter. Its only outside reference is the heap sort in Chapter 11.

We end the text with graphs in Chapter 12. Like many other data structure subjects, a complete course could be devoted to graphs. In this chapter, we review some basic graph concepts. Although this material could be covered anytime after Chapter 3, you will find that it contains some of the most difficult algorithms in the text. For this reason, we recommend that you present Chapter 12 at the end of the term, when your students will be much better prepared to handle the material.

Acknowledgments

No text of this scope can be developed without the support of many people. This is especially true for this text. The basic algorithms were field-tested by our students at De Anza College. Our first acknowledgment, therefore, has to be to the hundreds of students who by using and commenting on the text made a vital contribution. We especially thank our student, Scott Demouthe, who not only proofed the text, but verified every exercise and problem at the ends of the chapters.

We would also like to acknowledge the support of the De Anza staff. Their encouragement helped us launch the project, and their comments contributed to its success. To name them all is impossible, but we especially thank John Perry, Delia Garbacea, and George Rice.

To anyone who has not been through the process, the value of peer reviews cannot be fully appreciated. Writing a text rapidly becomes a myopic process. The important guidance of reviewers who can stand back and review the text as a whole cannot be measured. To twist an old cliché, "They are not valuable, they are priceless." We would especially like to acknowledge the contributions of the following reviewers:

James Clark, *University of Tennessee, Martin*

Roman Erenshateyn, *Goldey-Beacom College*

James Glenn, *University of Maryland*

Tracy Bradley Maples, *California State University—Long Beach*

Shensheng Zhao, *Governors State University*

Our thanks also go to our editors and staff at Brooks/Cole, Kallie Swanson, Grace Fujimoto, and Mary Vezilich. We would also like to acknowledge Kelli Jauron and Kathy Davis at Carlisle Publishers Services.

Last, and most obviously not the least, we thank our families and friends for their support. Many years ago an author described writing a text as a "locking yourself in a room" process. While the authors suffer through the writing process, families and friends suffer through their absence. We can only hope that as they view the final product, they feel that their sacrifices were worth it.

Richard F. Gilberg
Behrouz A. Forouzan

ABSTRACT DATA TYPES

Singly-Linked Lists

Program	3-9	Node template declaration (131)
Program	3-10	List ADT class declaration (132)
Program	3-11	Create linked list (133)
Program	3-12	Add node (134)
Program	3-13	Insert node (135)
Program	3-14	Remove node (136)
Program	3-15	Delete node (137)
Program	3-16	Search interface function (138)
Program	3-17	Search list (138)
Program	3-18	Empty list (140)
Program	3-19	Full list (140)
Program	3-20	List count (141)
Program	3-21	Traverse list (141)
Program	3-22	Destroy list (143)

Stack ADT-Linked List Implementation

Program	4-6	Stack ADT definitions (196)
Program	4-7	Create stack (197)
Program	4-8	Push stack (198)
Program	4-9	Pop stack (199)
Program	4-10	Retrieve stack top (200)
Program	4-11	Empty stack (200)
Program	4-12	Full stack (201)
Program	4-13	Stack count (201)
Program	4-14	Destroy stack (202)

Stack ADT-Array Implementation

Program	4-15	Data structure for stack array (204)
Program	4-16	Create stack constructor (205)
Program	4-17	Push stack array (206)
Program	4-18	Pop stack array (206)
Program	4-19	Retrieve stack array top (207)
Program	4-20	Empty stack (208)
Program	4-21	Full stack (208)
Program	4-22	Stack count (209)
Program	4-23	Destroy stack (209)

Queue ADT-Linked List Implementation

Program	5-5	Queue ADT data structures (246)
Program	5-6	Create queue (247)
Program	5-7	Enqueue (248)
Program	5-8	Dequeue (249)
Program	5-9	Queue front (250)
Program	5-10	Queue rear (250)
Program	5-11	Empty queue (251)
Program	5-12	Full queue (251)
Program	5-13	Queue count (252)
Program	5-14	Destroy queue (252)

Queue ADT-Array Implementation

Program	5-15	Data structure for queue array (256)
Program	5-16	Create queue: array implementation (256)
Program	5-17	Enqueue: array implementation (258)
Program	5-18	Dequeue: array implementation (259)
Program	5-19	Queue front: array implementation (259)
Program	5-20	Queue rear: array implementation (260)
Program	5-21	Queue full: array implementation (261)
Program	5-22	Destroy queue: array implementation (261)

AVL Tree ADT

Program	8-5	AVL class definition (379)
Program	8-6	AVL tree constructor (380)
Program	8-7	Application AVL insert function (381)
Program	8-8	Recursive insert function (382)
Program	8-9	AVL tree insert left balance (384)
Program	8-10	AVL tree rotate left and right (386)
Program	8-11	AVL tree delete application interface (387)
Program	8-12	AVL tree recursive delete (388)
Program	8-13	AVL tree delete right balance (390)
Program	8-14	AVL retrieve data (393)
Program	8-15	AVL tree recursive retrieve function (393)
Program	8-16	Traverse AVL tree (395)
Program	8-17	AVL empty tree (396)
Program	8-18	AVL full tree (396)
Program	8-19	AVL tree count (397)
Program	8-20	Destroy tree (397)
Program	8-21	Recursive destroy tree (398)

(continued in the back of the book)

Contents

1 Introduction 1

1-1 Pseudocode 2

Algorithm Header 2
Purpose, Conditions, and Return 3
Statement Numbers 4
Variables 4
Algorithm Analysis 5
Statement Constructs 5
Pseudocode Example 6

1-2 The Abstract Data Type 7

Atomic and Composite Data 8
Data Structure 8
Abstract Data Type 9

1-3 A Model for an Abstract Data Type 11

ADT Operations 11
ADT Data Structure 12
ADT Class Templates 13

1-4 Algorithm Efficiency 14

Linear Loops 15
Logarithmic Loops 16
Nested Loops 17
Big-O Notation 19
Standard Measures of Efficiency 20
Big-O Analysis Examples 21

1-5 Summary 24

1-6 Practice Sets 25

Exercises 25
Problems 27
Projects 27

2 Searching 29

2-1 List Searches 30

Sequential Search 30

Variations on Sequential Searches 33

Binary Search 37

Binary Search Algorithm 40

Analyzing Search Algorithms 41

2-2 C++ Search Algorithms 43

Sequential Search in C++ 43

Binary Search in C++ 44

Search Example 46

2-3 Hashed List Searches 49

Basic Concepts 50

Hashing Methods 52

Hashing Algorithm 57

2-4 Collision Resolution 59

Open Addressing 61

Linked List Resolution 65

Bucket Hashing 66

Combination Approaches 67

Hash List Example 68

2-5 Summary 72

2-6 Practice Sets 74

Exercises 74

Problems 75

Projects 75

3 Linked Lists 77

3-1 Linear List Concepts 78

Insertion 79

Deletion 80

Retrieval 80

Traversal 80

3-2 Linked List Concepts 81

Nodes 81

Linked List Data Structure 82

Pointers to Linked Lists 84

- 3-3 Linked List Algorithms 84**
 - Create List 84
 - Insert Node 85
 - Delete Node 90
 - Search List 93
 - Unordered List Search 96
 - Retrieve Node 97
 - Empty List 97
 - Full List 98
 - List Count 98
 - Traverse List 99
 - Destroy List 101
- 3-4 Processing a Linked List 102**
 - Add Node 104
 - Remove Node 105
 - Print List 106
 - Testing Insert and Delete Logic 107
- 3-5 List Applications 108**
 - Append Lists 108
 - Array of Lists 110
- 3-6 Complex Linked List Structures 113**
 - Circularly Linked Lists 113
 - Doubly Linked Lists 113
 - Multilinked Lists 119
 - Multilinked List Insert 121
 - Multilinked List Delete 122
- 3-7 Building a Linked List—C++ Implementation 122**
 - Data Structure 122
 - Application Functions 123
- 3-8 List Abstract Data Type—Linked List Implementation 129**
 - List ADT Declaration 131
- 3-9 Summary 143**
- 3-10 Practice Sets 144**
 - Exercises 144
 - Problems 147
 - Projects 148
- 4 Stacks 156**
 - 4-1 Basic Stack Operations 157**
 - Push 157
 - Pop 157
 - Stack Top 158
 - 4-2 Stack Linked List Implementation 160**
 - Data Structure 160
 - Stack Algorithms 161
 - 4-3 Stack Applications 168**
 - Reversing Data 169
 - Reverse a List 169
 - Convert Decimal to Binary 170
 - Parsing 171
 - Postponement 173
 - Backtracking 181
 - 4-4 Eight Queens Problem—C++ Implementation 188**
 - Main Line Logic 189
 - Get Board Size 190
 - 4-5 Stack Abstract Data Type Implementation 195**
 - Data Structure 195
 - Stack ADT Implementation 196
 - 4-6 Stack ADT—Array Implementation 202**
 - Array Data Structure 203
 - Create Stack Array 204
 - Push Stack Array 205
 - Pop Stack Array 206
 - Stack Top Array 207
 - Empty Stack Array 208
 - Full Stack Array 208
 - Stack Count Array 208
 - Destroy Stack Array 209
 - 4-7 Summary 209**
 - 4-8 Practice Sets 210**
 - Exercises 210
 - Problems 211
 - Projects 213
- 5 Queues 217**
 - 5-1 Queue Operations 218**
 - Enqueue 218
 - Dequeue 218
 - Queue Front 219
 - Queue Rear 220
 - Queue Example 220
 - 5-2 Queue Linked List Design 220**
 - Data Structure 220
 - Queue Algorithms 222

Create Queue	224
Enqueue	224
Dequeue	225
Retrieving Queue Data	227
Empty Queue	228
Full Queue	228
Queue Count	228
Destroy Queue	229
5-3 Queuing Theory	229
5-4 Queue Applications	231
Queue Simulation	231
Categorizing Data	239
5-5 Categorizing Data—C++ Implementation	241
Main Line Logic	241
Fill Queues	242
Print Queues	244
Print One Queue	244
5-6 Queue ADT—Linked List Implementation	246
Queue Structure	246
Queue ADT Implementation	247
5-7 Queue ADT—Array Implementation	253
Array Queues Implementation	254
5-8 Summary	261
5-9 Practice Sets	262
Exercises	262
Problems	265
Projects	266
6 Recursion	271
6-1 Factorial—A Case Study	272
Recursion Defined	272
Iterative Solution	273
Recursive Solution	273
6-2 How Recursion Works	275
6-3 Designing Recursive Algorithms	277
The Design Methodology	278
Limitations of Recursion	279
Design Implementation—Reverse a Linked List	279
6-4 Another Case Study—Fibonacci Numbers	281

6-5 The Towers of Hanoi	285
Recursive Towers Of Hanoi	286
6-6 C++ Implementations of Recursion	290
Fibonacci Numbers	290
Prefix to Postfix Conversion	291
Towers of Hanoi	297
6-7 Summary	299
6-8 Practice Sets	300
Exercises	300
Problems	302
Projects	303

7 Introduction to Trees 305

7-1 Basic Tree Concepts	306
Terminology	306
Tree Representation	308
7-2 Binary Trees	310
Properties	312
7-3 Binary Tree Traversals	313
Depth-First Traversals	314
Breadth-First Traversals	320
7-4 Expression Trees	322
Infix Traversal	322
Postfix Traversal	324
Prefix Traversal	324
7-5 General Trees	324
Changing General Tree to Binary Tree	325
Insertions into General Trees	326
General Tree Deletions	327
7-6 Huffman Code	327
7-7 Summary	332
7-8 Practice Sets	333
Exercises	333
Problems	337
Projects	337

8 Search Trees 338

8-1 Binary Search Trees	339
Definition	339

Operations on Binary Search

Trees 341

Binary Search Tree Search

Algorithms 342

8-2 AVL Trees 353

AVL Balance Factor 354

Balancing Trees 355

AVL Node Structure 360

AVL Delete Algorithm 367

Adjusting the Balance Factors 372

8-3 AVL Tree Implementation 372

Data Structure 372

Program Design 373

Count Words Summary 377

8-4 AVL Abstract Data Type 378

AVL Tree Data Structures 379

AVL Tree Functions 380

AVL Tree Data Processing 392

AVL Tree Utility Functions 395

8-5 Summary 398

8-6 Practice Sets 399

Exercises 399

Problems 403

Projects 403

9 Heaps 406

9-1 Heap Definition 407

9-2 Heap Structure 407

9-3 Basic Heap Algorithms 408

ReheapUp 409

ReheapDown 411

9-4 Heap Data Structure 412

9-5 Heap Algorithms 414

ReheapUp 414

ReheapDown 415

BuildHeap 416

InsertHeap 418

DeleteHeap 419

9-6 Heap Applications 421

Selection Algorithms 421

Priority Queues 423

9-7 A Heap Program 424

Heap Program Design 425

Heap Functions 430

9-8 Summary 433

9-9 Practice Sets 434

Exercises 434

Problems 436

Projects 436

10 Multiway Trees 439

10-1 m-Way Search Trees 440

10-2 B-Trees 442

B-Tree Insertion 443

B-Tree Insert Design 444

B-Tree Insert Node 446

B-Tree Deletion 454

Traverse B-Tree 467

B-Tree Search 470

10-3 Simplified B-Trees 471

2-3 Tree 471

2-3-4 Tree 472

10-4 B-Tree Variations 472

B*Tree 473

B+Tree 474

10-5 Lexical Search Tree 474

Tries 475

Trie Structure 476

10-6 B-Tree Abstract Data Type 478

Header File 479

Utility Functions 481

Insert Algorithms 485

Delete Algorithms 491

10-7 Summary 499

10-8 Practice Sets 499

Exercises 499

Problems 500

Projects 501

11 Advanced Sorting Concepts 502

11-1 General Sort Concepts 503

Sort Order 503

Sort Stability 504

Sort Efficiency 504

Passes 505

11-2 Insertion Sorts 505

Straight Insertion Sort 505

Shell Sort 508

Insertion Sort Algorithms 513
 Insertion Sort Implementation 515

11-3 Selection Sorts 517

Straight Selection Sort 517
 Selection Sort Algorithms 522
 Selection Sort Implementation 524

11-4 Exchange Sorts 526

Bubble Sort 526
 Bubble Sort Algorithm 527
 Quick Sort 529
 Exchange Sort Algorithms 536

11-5 Summary 538

Exchange Sort Implementation 538

11-6 External Sorts 542

Merging Ordered Files 543
 Merging Unordered Files 544
 The Sorting Process 546
 Sort Phase Revisited 551

11-7 Summary 553

11-8 Practice Sets 554

Exercises 554
 Problems 556
 Projects 556

12 Graphs 560

12-1 Terminology 561

12-2 Operations 563

Add Vertex 563
 Delete Vertex 563
 Add Edge 563
 Delete Edge 564
 Find Vertex 564
 Traverse Graph 564

12-3 Graph Storage Structures 568

Adjacency Matrix 568
 Adjacency List 569

12-4 Graph Algorithms 570

Create Graph 571
 Insert Vertex 572
 Delete Vertex 573
 Insert Arc 575

Delete Arc 577

Retrieve Vertex 578

First Arc 579

Depth-First Traversal 581

Breadth-First Traversal 583

12-5 Networks 585

Minimum Spanning Tree 585

Shortest Path Algorithm 591

12-6 Abstract Data Type 596

Insert Vertex 598

Delete Vertex 599

Insert Arc 600

Delete Arc 602

Depth-First Traversal 604

Breadth-First Traversal 605

12-7 Summary 607

12-8 Practice Sets 608

Exercises 608

Problems 610

Projects 611

Appendixes

A ASCII Tables 615

B Structure Charts 620

**C Program Standards and
 Styles 627**

D Random Numbers 633

E Standard C++ Libraries 639

F C++ Function Prototypes 641

**G Classes Related to Input and
 Output 650**

H The String Class 655

I Pointers to Functions 666

J Inheritance 670

K C++ Templates 685

L Standard Template Library 692

Solutions to Selected Exercises 712

Glossary 735

Index 747

Introduction

1

This text assumes that the student has a solid foundation in structured programming principles and has written programs of moderate complexity. Although the text uses C++ for all of its implementation examples, the design and logic of the data structure algorithms are based on pseudocode. This approach creates a language-independent environment for the algorithms.

In this chapter we establish a background for the tools used in the rest of the text, most specifically pseudocode, the abstract data type, and algorithm efficiency analysis. We also introduce the measures we use throughout the text to discuss algorithm efficiency.

1-1 PSEUDOCODE

Although several tools are used to define algorithms, one of the most common is **pseudocode**. Pseudocode is an English-like representation of the code required for an algorithm. It is part English, part structured code. The English part provides a relaxed syntax that is easy to read. The code part consists of an extended version of the basic algorithmic constructs—sequence, selection, and iteration.

Note

One of the most common tools for defining algorithms is pseudocode, which is part English, part structured code.

In this text we use pseudocode for both data structures and code. The basic format for data types consists of the name of the data and its type enclosed in pointed brackets as shown below

```
count <integer>
```

The structure of the data is indicated by indenting the data items as shown below.

```
node
  data    <dataType>
  link    <pointer to node>
end node
```

This data definition describes a node in a self-referential linked list that consists of a nested structure (data) and a pointer to the next node (link). It assumes that the data description for `dataType` has been previously defined.

As mentioned, the pseudocode is used to describe an algorithm. To facilitate a discussion of the algorithm statements, we number them using the hierarchical system shown in Algorithm 1-1 and fully described in the following sections.

Algorithm Header

Each algorithm begins with a header that names it, describes its parameters, and lists any pre- and postconditions. This information is important because the programmer using the algorithm often sees only the header information, not the complete algorithm. Therefore, the header information must be complete enough to communicate to the programmer everything he or she must know to use the algorithm.

In Algorithm 1-1 there is only one parameter, page number. Parameters are identified as pass by reference (ref) or pass by value (val). The type is included in pointed brackets after the identifier.