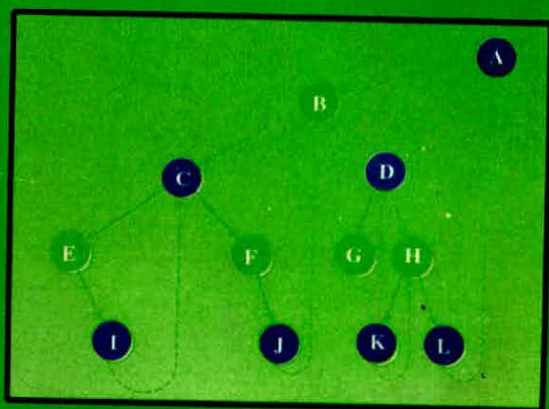


Eastern
Economy
Edition



SECOND EDITION

DATA STRUCTURES USING C AND C++

Yedidyah Langsam • Moshe J. Augenstein
Aaron M. Tenenbaum



Second Edition

DATA STRUCTURES USING C AND C++

**Yedidyah Langsam
Moshe J. Augenstein
Aaron M. Tenenbaum**

Brooklyn College

Prentice-Hall of India Private Limited

New Delhi - 110 001

2000

This Nineteenth Indian Reprint—Rs. 225.00
(Original U.S. Edition—Rs. 2964.00)

DATA STRUCTURES USING C AND C++, 2nd Ed.
by Yedidyah Langsam, Moshe J. Augenstein and Aaron M. Tenenbaum

© 1996 by Prentice-Hall, Inc., Upper Saddle River, New Jersey 07458, U.S.A. All rights reserved. No part of this book may be reproduced in any form, by mimeograph or any other means, without permission in writing from the publisher.

The authors and publisher of this book have used their best efforts in preparing this book. These efforts include the development, research, and testing of the theories and programs to determine their effectiveness. The authors and publisher make no warranty of any kind, expressed or implied, with regard to these programs or the documentation contained in this book. The authors and publisher shall not be liable in any event for incidental or consequential damages in connection with, or arising out of, the furnishing, performance, or use of these programs.

ISBN-81-203-1177-9

The export rights of this book are vested solely with the publisher.

This Eastern Economy Edition is the authorized, complete and unabridged photo-offset reproduction of the latest American edition specially published and priced for sale only in Bangladesh, Burma, Cambodia, China, Fiji, Hong Kong, India, Indonesia, Laos, Malaysia, Nepal, Pakistan, Philippines, Singapore, South Korea, Sri Lanka, Taiwan, Thailand, and Vietnam.

Reprinted in India by special arrangement with Prentice-Hall, Inc., Upper Saddle River, New Jersey 07458, U.S.A.

Nineteenth Printing (Second Edition) December, 2000

Published by Asoke K. Ghosh, Prentice-Hall of India Private Limited, M-97, Connaught Circus, New Delhi-110001 and Printed by Syndicate Binders, B-167, Okhla Industrial Area, Phase I, New Delhi-110020.

Contents

Preface

xiii

1 Introduction to Data Structures

1

1.1 Information and Meaning 1

Binary and Decimal Integers, 2

Real Numbers, 4

Character Strings, 5

Hardware and Software, 6

Concept of Implementation, 8

Example, 8

Abstract Data Types, 13

Sequences as Value Definitions, 17

ADT for Varying-length Character Strings, 18

Data Types in C, 20

Pointers in C, 20

Data Structures and C, 22

Exercises 24

1.2 Arrays in C 24

The Array as an ADT, 26

Using One-Dimensional Arrays, 27

iii

Implementing One-Dimensional Arrays, 28

Arrays as Parameters, 31

Character Strings in C, 32

Character String Operations, 33

Two-Dimensional Arrays, 34

Multidimensional Arrays, 37

Exercises 40

1.3 Structures in C 42

Implementing Structures, 46

Unions, 48

Implementation of Unions, 51

Structure Parameters, 52

Representing Other Data Structures, 54

Rational Numbers, 55

Allocation of Storage and Scope of Variables, 58

Exercises 62

1.4 Classes in C++ 63

The Class Rational, 64

Using the Class Rational, 65

Implementing the Methods, 67

Overloading, 72

Inheritance, 72

Constructors, 74

Exercises 76

2 The Stack

77

2.1 Definition and Examples 77

Primitive Operations, 80

Example, 81

The Stack as an Abstract Data Type, 84

Exercises 85

2.2 Representing Stacks in C 86

Implementing the pop Operation, 90

Testing for Exceptional Conditions, 91

Implementing the Push Operation, 92

Exercises 95

2.3 Example: Infix, Postfix, and Prefix 95

Basic Definitions and Examples, 95

Evaluating a Postfix Expression, 98

Program to Evaluate a Postfix Expression, 99

Limitations of the Program, 102

Contents

Converting an Expression from Infix to Postfix, 102
Program to Convert an Expression from Infix to Postfix, 106
Stacks in C++ Using Templates, 109

Exercises 115

Recursion

117

3.1 Recursive Definition and Processes 117

Factorial Function, 117
Multiplication of Natural Numbers, 120
Fibonacci Sequence, 121
Binary Search, 122
Properties of Recursive Definitions or Algorithms, 125

Exercises 126

3.2 Recursion in C 127

Factorial in C, 127
Fibonacci Numbers in C, 131
Binary Search in C, 132
Recursive Chains, 134
Recursive Definition of Algebraic Expressions, 135

Exercises 138

3.3 Writing Recursive Programs 140

The Towers of Hanoi Problem, 142
Translation from Prefix to Postfix Using Recursion, 146

Exercises 150

3.4 Simulating Recursion 153

Return from a Function, 155
Implementing Recursive Functions, 156
Simulation of Factorial, 157
Improving the Simulated Routine, 161
Eliminating gotos, 163
Simulating the Towers of Hanoi, 165

Exercises 170

3.5 Efficiency of Recursion 172

Exercises 173

Queues and Lists

174

4.1 The Queue and Its Sequential Representation 174

The Queue as an Abstract Data Type, 176
C Implementation of Queues, 176
insert Operation, 180

	<i>Priority Queue</i> , 182
	<i>Array Implementation of a Priority Queue</i> , 183
	Exercises 184
4.2	Linked Lists 186
	<i>Inserting and Removing Nodes from a List</i> , 187
	<i>Linked Implementation of Stacks</i> , 191
	<i>getnode and freenode Operations</i> , 193
	<i>Linked Implementation of Queues</i> , 194
	<i>Linked List as a Data Structure</i> , 195
	<i>Examples of List Operations</i> , 198
	<i>List Implementation of Priority Queues</i> , 200
	<i>Header Nodes</i> , 200
	Exercises 202
4.3	Lists in C 203
	<i>Array Implementation of Lists</i> , 203
	<i>Limitations of the Array Implementation</i> , 206
	<i>Allocating and Freeing Dynamic Variables</i> , 207
	<i>Linked Lists Using Dynamic Variables</i> , 211
	<i>Queues as Lists in C</i> , 213
	<i>Examples of List Operations in C</i> , 215
	<i>Noninteger and Nonhomogeneous Lists</i> , 216
	<i>Comparing the Dynamic and Array Implementations of Lists</i> , 217
	<i>Implementing Header Nodes</i> , 218
	Exercises 219
4.4	Example: Simulation Using Linked Lists 220
	<i>Simulation Process</i> , 221
	<i>Data Structures</i> , 222
	<i>Simulation Program</i> , 223
	Exercises 227
4.5	Other List Structures 228
	<i>Circular Lists</i> , 229
	<i>Stack as a Circular List</i> , 229
	<i>Queue as a Circular List</i> , 230
	<i>Primitive Operations on Circular Lists</i> , 231
	<i>The Josephus Problem</i> , 232
	<i>Header Nodes</i> , 234
	<i>Addition of Long Positive Integers Using Circular Lists</i> , 235
	<i>Doubly Linked Lists</i> , 237
	<i>Addition of Long Integers Using Doubly Linked Lists</i> , 239
	Exercises 244
4.6	The Linked List in C++ 245
	Exercises 248

- 5.1 Binary Trees 249
Operations on Binary Trees, 254
Applications of Binary Trees, 255
 Exercises 260
- 5.2 Binary Tree Representations 261
Node Representation of Binary Trees, 261
Internal and External Nodes, 264
Implicit Array Representation of Binary Trees, 265
Choosing a Binary Tree Representation, 269
Binary Tree Traversals in C, 270
Threaded Binary Trees, 273
Traversal Using a father Field, 277
Heterogeneous Binary Trees, 280
 Exercises 281
- 5.3 Example: The Huffman Algorithm 283
The Huffman Algorithm, 287
C Program, 288
 Exercises 291
- 5.4 Representing Lists as Binary Trees 292
Finding the kth Element, 294
Deleting an Element, 296
Implementing Tree-Represented Lists in C, 299
Constructing a Tree-Represented List, 301
The Josephus Problem Revisited, 303
 Exercises 304
- 5.5 Trees and Their Applications 305
C Representations of Trees, 307
Tree Traversals, 309
General Expressions as Trees, 312
Evaluating an Expression Tree, 315
Constructing a Tree, 317
 Exercises 319
- 5.6 Example: Game Trees 321
 Exercises 327

- 6.1 General Background 329
Efficiency Considerations, 331

- O Notation*, 334
- Efficiency of Sorting*, 336
- Exercises 338
- 6.2 Exchange Sorts 339
 - Bubble Sort*, 339
 - Quicksort*, 342
 - Efficiency of Quicksort*, 348
 - Exercises 350
- 6.3 Selection and Tree Sorting 351
 - Straight Selection Sort*, 352
 - Binary Tree Sorts*, 353
 - Heapsort*, 356
 - Heap as a Priority Queue*, 357
 - Sorting Using a Heap*, 359
 - Heapsort Procedure*, 362
 - Exercises 364
- 6.4 Insertion Sorts 365
 - Simple Insertion*, 365
 - Shell Sort*, 366
 - Address Calculation Sort*, 370
 - Exercises 372
- 6.5 Merge and Radix Sorts 373
 - Merge Sorts*, 373
 - The Cook–Kim Algorithm*, 377
 - Radix Sort*, 377
 - Exercises 381

Searching

384

- 7.1 Basic Search Techniques 384
 - Dictionary as an Abstract Data Type*, 385
 - Algorithmic Notation*, 386
 - Sequential Searching*, 387
 - Efficiency of Sequential Searching*, 389
 - Reordering a List for Maximum Search Efficiency*, 390
 - Searching an Ordered Table*, 392
 - Indexed Sequential Search*, 392
 - Binary Search*, 394
 - Interpolation Search*, 397
 - Exercises 398
- 7.2 Tree Searching 401

Inserting into a Binary Search Tree, 404
Deleting from a Binary Search Tree, 404
Efficiency of Binary Search Tree Operations, 407
Efficiency of Nonuniform Binary Search Trees, 409
Optimum Search Trees, 411
Balanced Trees, 413

Exercises 421

7.3 General Search Trees 423

Multiway Search Trees, 423
Searching a Multiway Tree, 426
Implementing a Multiway Tree, 427
Traversing a Multiway Tree, 428
Insertion in a Multiway Search Tree, 430
B-Trees, 435
Algorithms for B-Tree Insertion, 439
Computing father and index, 445
Deletion in Multiway Search Trees, 449
Efficiency of Multiway Search Trees, 453
Improving the B-Tree, 456
B⁺-Trees, 460
Digital Search Trees, 461
Tries, 465

Exercises 467

7.4 Hashing 468

Resolving Hash Clashes by Open Addressing, 470
Deleting Items from a Hash Table, 473
Efficiency of Rehashing Methods, 474
Hash Table Reordering, 476
Brent's Method, 477
Binary Tree Hashing, 480
Improvements with Additional Memory, 482
Coalesced Hashing, 485
Separate Chaining, 488
Hashing in External Storage, 491
Separatior Method, 493
Dynamic Hashing and Extendible Hashing, 494
Linear Hashing, 499
Choosing a Hash Function, 505
Perfect Hash Functions, 508
Universal Classes of Hash Functions, 512

Exercises 513

Graphs and Their Applications 515

8.1 Graphs 515

Application of Graphs, 517

C Representation of Graphs, 520
Transitive Closure, 521
Warshall's Algorithm, 525
Shortest-Path Algorithm, 526

Exercises 528

8.2 A Flow Problem 529

Improving a Flow Function, 531
Example, 535
Algorithm and Program, 537

Exercises 541

8.3 Linked Representation of Graphs 541

Dijkstra's Algorithm Revisited, 547
Organizing the Set of Graph Nodes, 549
Application to Scheduling, 550
C Program, 554

Exercises 557

8.4 Graph Traversal and Spanning Forests 560

Traversal Methods for Graphs, 560
Spanning Forests, 563
Undirected Graphs and Their Traversals, 566
Depth-First Traversal, 568
Applications of Depth-First Traversal, 571
Efficiency of Depth-First Traversal, 572
Breadth-First Traversal, 573
Minimum Spanning Trees, 574
Kruskal's Algorithm, 576
Round-Robin Algorithm, 577

Exercises 577

9 Storage Management

579

9.1 General Lists 579

Operations That Modify a List, 582
Examples, 583
Linked List Representation of a List, 584
Representation of Lists, 587
clist Operation, 588
Use of List Headers, 591
Freeing List Nodes, 593
General Lists in C, 594
Programming Languages and Lists, 597

Exercises 599

9.2 Automatic List Management 599

Reference Count Method, 599
Garbage Collection, 605
Algorithms for Garbage Collection, 606
Collection and Compaction, 613
Variations of Garbage Collection, 619

Exercises 620

9.3 **Dynamic Memory Management** 621

Compaction of Blocks of Storage, 622
First Fit, Best Fit, and Worst Fit, 625
Improvements in the First-Fit Method, 631
Freeing Storage Blocks, 632
Boundary Tag Method, 633
Buddy System, 636
Other Buddy Systems, 643

Exercises 645

Bibliography and References

647

Index

663

Preface

This text is designed for a two-semester course in data structures and programming. For several years, we have taught a course in data structures to students who have had a semester course in high-level language programming and a semester course in assembly language programming. We found that a considerable amount of time was spent in teaching programming techniques because the students had not had sufficient exposure to programming and were unable to implement abstract structures on their own. The brighter students eventually caught on to what was being done. The weaker students never did. Based on this experience, we have reached the firm conviction that a first course in data structures must go hand in hand with a second course in programming. This text is a product of that conviction.

The text introduces abstract concepts, shows how those concepts are useful in problem solving, and then shows how the abstractions can be made concrete by using a programming language. Equal emphasis is placed on both the abstract and the concrete versions of a concept, so that the student learns about the concept itself, its implementation, and its application.

The languages used in this text are C and C++. C is well suited to such a course since it contains the control structures necessary to make programs readable and allows basic data structures such as stacks, linked lists, and trees to be implemented in a variety of ways. This allows the student to appreciate the choices and tradeoffs which face a programmer in a real situation. C is also widespread on many different computers and it continues to grow in popularity. As Kernighan and Ritchie indicate, C is "a pleasant, expressive, and versatile language."

We have included information on C++ in the early chapters, introducing the features of C++ and showing how they can be used in implementing data structures. No specific background in C++ is needed. Classes in C++ are introduced in a new Section 1.4. This section discusses classes, including function members. It also introduces inheritance and object orientation. The section includes an example of implementing abstract data types in C++, as well as polymorphism. To Section 2.3 we have added an implementation of stacks in C++ using templates. This shows how complex data structures can be parameterized for different base types. A new Section 4.6 has been added, showing how linked lists can be implemented in C++. Such an implementation shows the limitations, as well as the power, of encapsulation in implementing data structures. The point should be made that encapsulated data structures must be designed carefully to allow users to do what they need in a data structure. Also discussed in this context are C++ dynamic allocation and freeing of storage.

The only prerequisite for students using this text is a one-semester course in programming. Students who have had a course in programming using such languages as FORTRAN, Pascal, or PL/I can use this text together with one of the elementary C or C++ texts listed in the Bibliography. Chapter 1 also provides information necessary for such students to acquaint themselves with C.

Chapter 1 is an introduction to data structures. Section 1.1 introduces the concept of an abstract data structure and the concept of an implementation. Sections 1.2 and 1.3 introduce arrays and structures in C. The implementations of these two data structures as well as their applications are covered. Chapter 2 discusses stacks and their C implementation. Since this is the first new data structure introduced, considerable discussion of the pitfalls of implementing such a structure is included. Section 2.3 introduces postfix, prefix, and infix notations. Chapter 3 covers recursion, its application, and its implementation. Chapter 4 introduces queues, priority queues, and linked lists and their implementations both using an array of available nodes as well as using dynamic storage. Chapter 5 discusses trees, Chapter 6 introduces O notation and covers sorting, while Chapter 7 covers both internal and external searching. Chapter 8 introduces graphs, and Chapter 9 discusses storage management. At the end of the text, we have included a large Bibliography with each entry classified by the appropriate chapter or section of the text.

A one-semester course in data structures consists of Section 1.1, Chapters 2 through 7, and Sections 8.1, 8.2, and part of Section 8.4. Parts of Chapters 3, 6, 7, and 8 can be omitted if time is pressing.

This text is suitable for courses based upon the Algorithms and Data Structures knowledge unit (AL 1-6, 8) as well as sections of the Programming Languages knowledge unit (PL 3-6, 10, 11) as described in the report *Computing Curricula 1991* of the ACM/IEEE-CS Joint Curriculum Task Force. It follows closely the sample Data Structures and Analysis of Algorithms course presented in the report and may be used in second- and third-tier classes of a typical computer science curriculum for both majors and nonmajors.

The text is suitable for course C82 and parts of courses C87 and C813 of Curriculum 78 (*Communications of the ACM*, March 1979), courses UC1 and UC8 of the Undergraduate Programs in Information Systems (*Communications of the ACM*, December 1973) and course I1 of Curriculum 68 (*Communications of the ACM*, March

1968). In particular, the text covers parts or all of topics P1, P2, P3, P4, P5, S2, D1, D2, D3, and D6 of Curriculum 78.

Algorithms are presented as intermediaries between English language descriptions and C programs. They are written in C style interspersed with English. These algorithms allow the reader to focus on the method used to solve a problem without concern about declaration of variables and the peculiarities of real language. In transforming an algorithm into a program, we introduce these issues and point out the pitfalls that accompany them.

The indentation pattern used for programs and algorithms is based loosely on a format suggested by Kernighan and Ritchie (*The C Programming Language*, Prentice Hall, 1978) which we have found to be quite useful. We have also adopted the convention of indicating in comments the construct being terminated by each instance of a closing brace (}). Together with the indentation pattern, this is a valuable tool in improving program comprehensibility. We distinguish between algorithms and programs by presenting the former in italics and the latter in roman.

Most of the concepts in the text are illustrated by several examples. Some of these examples are important topics in their own right (e.g., postfix notation, multiword arithmetic, etc.) and may be treated as such. Other examples illustrate different implementation techniques (such as sequential storage of trees). The instructor is free to cover as many or as few of these examples as he or she wishes. Examples may also be assigned to students as independent reading. It is anticipated that an instructor will be unable to cover all the examples in sufficient detail within the confines of a one- or two-semester course. We feel that at the stage of a student's development for which the text is designed, it is more important to cover several examples in great detail than to cover a broad range of topics cursorily.

All the programs and algorithms in this text have been tested and debugged. We wish to thank Miriam Binder and Irene LaClaustra for their invaluable assistance in this task. Their zeal for the task was above and beyond the call of duty and their suggestions were always valuable. Of course, any errors that remain are the sole responsibility of the authors.

The exercises vary widely in type and difficulty. Some are drill exercises to ensure comprehension of topics in the text. Others involve modifications of programs or algorithms presented in the text. Still others introduce new concepts and are quite challenging. Often, a group of successive exercises includes the complete development of a new topic which can be used as the basis for a term project or an additional lecture. The instructor should use caution in assigning exercises so that an assignment is suitable to the student's level. We consider it imperative for students to be assigned several (from five to twelve, depending on difficulty) programming projects per semester. The exercises contain several projects of this type.

We have attempted to use the C language, as specified in the second edition of the Kernighan and Ritchie text. This corresponds to the C ANSI Standard. Programs given in this book have all been developed using Borland C++ but have only made use of features as described in the evolving ANSI C++ draft standard. They should run without change on a wide variety of C++ compilers. See the reference manual for your particular system or consult the "Working Paper for Draft Proposed International Standard for Information System—Programming Language C++," available from the

American National Standards Institute (ANSI) Standards Secretariat: CBEMA, 1250 Eye Street NW, Suite 200, Washington, DC 20005. You should, of course, warn your students about any idiosyncrasies of the particular compiler they are using. We have also added some references to several personal computer C and C++ compilers.

Miriam Binder and Irene LaClaustra spent many hours typing and correcting the original manuscript as well as managing a large team of students whom we mention below. Their cooperation and patience as we continually made up and changed our minds about additions and deletions are most sincerely appreciated.

We would like to thank Shaindel Zundel-Margulis, Cynthia Richman, Gittie Rosenfeld-Wertenteil, Mindy Rosman-Schreiber, Nina Silverman, Helene Turry, and Devorah Sadowsky-Weinschneider for their invaluable assistance.

Vivienne Esther Langsam and Tziyonah Miriam Langsam spent many hours revising the index for the second edition of this book. We would like to thank them for helping us complete the book in the face of a fast approaching deadline.

The staff of the City University Computer Center deserves special mention. They were extremely helpful in assisting us in using the excellent facilities of the center. The same can be said of the staff of the Brooklyn College Computer Center.

We would like to thank the editors and staff at Prentice Hall and especially the reviewers for their helpful comments and suggestions.

Finally, we thank our wives, Vivienne Esther Langsam, Gail Augenstein, and Miriam Tenenbaum, for their advice and encouragement during the long and arduous task of producing such a book.

YEDIDYAH LANGSAM
MOSHE J. AUGENSTEIN
AARON M. TENENBAUM

To my wife, Vivienne Esther
YL

To my wife, Gail
MA

To my wife, Miriam
AT
