DATA STRUCTURES

A Pseudocode Approach with C++

Richard F. Gilberg • Behrouz A. Forouzan

Data Structures

A Pseudocode Approach with C++

Richard F. Gilberg
De Anza College

Behrouz A. Forouzan
De Anza College

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*

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)

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 Kioumehr
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.

Features of This Book

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

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 predefined 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 reenforce 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.

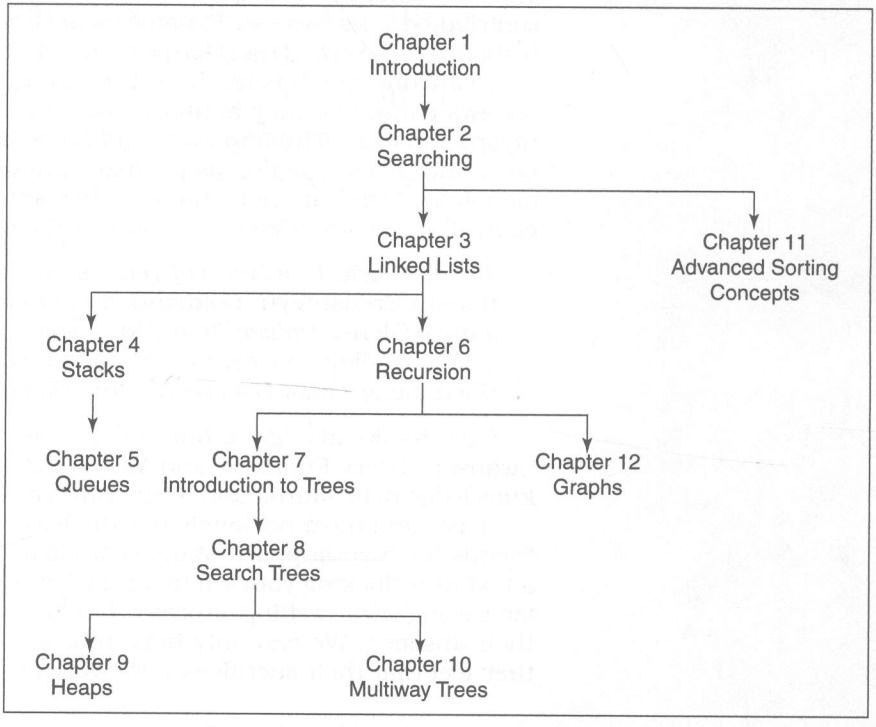
Projects are longer, major assignments that may take an average student 6 to 9 hours or more to develop.

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 cliche, "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 Erenshteyn, *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

Stack ADT–Linked List Implementation

Stack ADT–Array Implementation

Queue ADT–Linked List Implementation

Queue ADT–Array Implementation

AVL Tree ADT

(continued in the back of the book)

Contents

1 Introduction

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.