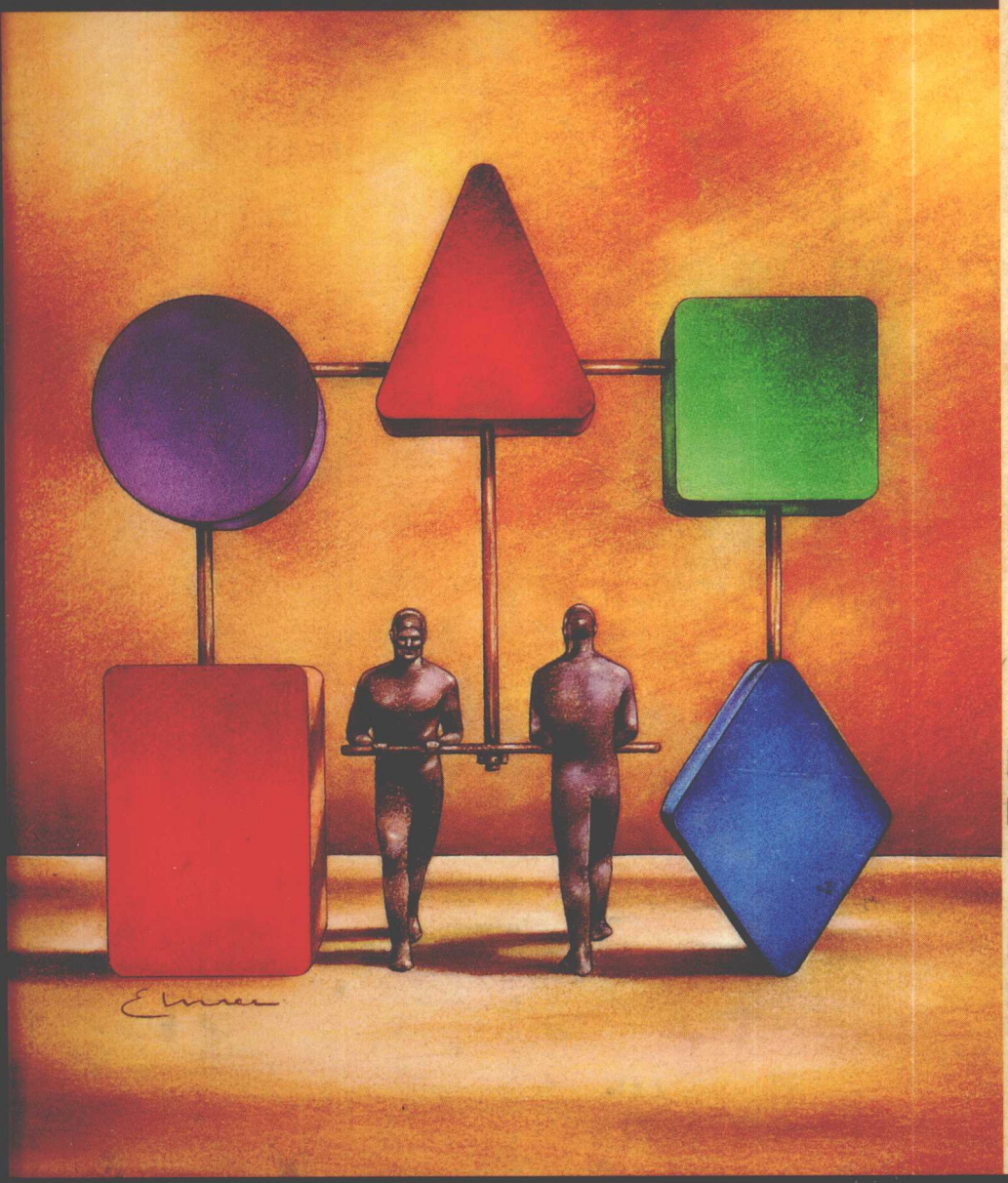
FUNDAMENTALS OF DATA STRUCTURES IN PASCAL

FOURTH EDITION



ELLIS HOROWITZ

SARTAJ SAHNI

Fundamentals of Data Structures in Pascal

FOURTH EDITION

Ellis Horowitz
University of Southern California

Sartaj Sahni
University of Florida

COMPUTER SCIENCE PRESS
An imprint of W. H. Freeman and Company
New York

Cover illustration by Richard Elmer

Fundamentals of Data Structures in Pascal is the result of the combined efforts of the authors. Their names have been listed in alphabetical order with no implication that one is senior and the other junior.

Printed in the United States of America

Computer Science Press

An imprint of W. H. Freeman and Company
The book publishing arm of Scientific American
41 Madison Avenue, New York, NY 10010
20 Beaumont Street, Oxford OX1 2NQ, England

1 2 3 4 5 6 7 8 9 0 R R D 9 9 8 7 6 5 4

PREFACE

This edition retains the in-depth discussion of the algorithms and computing time analyses found in earlier editions of this book. In addition we have attempted to preserve the chapter organization and the presentation style of the earlier editions whenever it was desirable. But this has not kept us from making improvements. For example, the discussion of strings is now found in the chapter on arrays; internal and external sorting methods have now been combined into a single chapter, Chapter 7; the previous chapter on advanced data structures has been split into two chapters, Chapter 9 which deals with heap structures and Chapter 10 which deals exclusively with search structures; exercises are now placed immediately after the relevant section. We have rearranged the sections in each chapter so that the basic material appears early in the chapter and the difficult or optional material appears at the end of the chapter.

One of the major new features in this book, compared to its earlier version, is the inclusion of abstract data types. The major idea is to separate out the issue of data type specification from implementation. Languages such as Ada provide direct support for such a split, but in Pascal there is no equivalent construct. Therefore, we have devised a straightforward notation in which we express an abstract data type. Basically, we provide a description of the objects of the type followed by the names and arguments of the functions of the type. Instructors can discuss with the students the specification of the data type before moving on to implementation issues and concerns for efficiency of the algorithms.

In several sections, we have enhanced the presentation by including additional material. For example, the discussion on the representation of sets by trees has been enhanced by the inclusion of two additional strategies for tree compaction and an addi-

tional strategy for combining two trees during a union operation; the discussion on single-source/all-destinations shortest paths has been extended to include an algorithm for the case when edge weights may be negative; the discussion on binary search trees and red-black trees has been extended to include the operations of split, join, and combine.

USING THIS TEXT FOR A COURSE

For the instructor who intends to use this book and is teaching on a semester basis we present the following two possibilities, a medium pace and a rigorous pace. The medium pace is recommended when the course is for begining computer science majors, possibly their second or third course of the curriculum. Most people, including the authors, have taught according to the medium pace. The outline below corresponds to the curriculum recommended by the ACM, in particular course C2, (Curriculum '78, CACM 3/79, and CACM 8/85).

SEMESTER SCHEDULE - MEDIUM PACE

Week	Subject	Reading Assignment
1	Intro. to Algorithms and Data Organization	Chapter 1
2	Arrays	Chapter 2
3	Arrays (strings)	First program due
4	Stacks and Queues	Chapter 3
5	Linked Lists (singly and doubly linked)	Chapter 4
6	Linked Lists	Second program due
7	Trees (basic facts, binary trees)	Chapter 5
8	Trees (search, heap)	
9	Mid Term	
10	Graphs (basic facts, representations)	Chapter 6
11	Graphs (shortest paths, spanning trees, topological sorting)	Third program due
12	Internal Sorting (insertion, quick, and merge)	Chapter 7
13	Internal Sorting (heap, radix)	Fourth program due
14	Hashing	Chapter 8
15	Heap Structures (Selected Topics)	Chapter 9
16	Search Structures (Selected Topics)	Chapter 10

We recommend that several programming assignments be given, spaced somewhat evenly thoughout the semester. The aim of the first program is primarily to get the students familiar with the computing environment. The second program should emphasize list structures, as discussed in Chapter 4. There are several suggestions for projects at the end of the exercises of Chapter 4. One topic we have chosen to skip is external sorting.

This leaves time to cover one of the most important of techniques, hashing. This topic is used in several courses later on in the curriculum, so it is important to cover it this semester. The instructor will likely not have time to cover the material in the Search Structures chapter. Perhaps one or two topics can be selectively chosen.

The more rigorous pace would be appropriate when the book is used for a first year graduate course, or for an advanced undergraduate course. Our suggested outline follows.

SEMESTER SCHEDULE - RIGOROUS PACE

Week	Subject	Reading Assignment
1	Intro. to Algorithms and Data Organization	Chapter 1
2	Arrays	Chapter 2
3	Stacks and Queues	Chapter 3
		First program due
4	Linked Lists	Chapter 4
5	Trees	Chapter 5
6	Trees continued	Second program due
7	Mid Term	
8	Graphs	Chapter 6
9	Graphs continued	Third program due
10	Internal Sorting	Chapter 7
11	External Sorting	Chapter 7
12	Hashing	Chapter 8
13	Heap Structures	Chapter 9
		Fourth program due
14	Heap Structures	Chapter 9
15	Search Structures	Chapter 10
16	Search Structures	Chapter 10

The programming assignments and midterm exam are paced exactly as in the medium case. However, the lectures proceed at a faster rate. For the rigorous pace, two weeks are allotted for Chapters 9 and 10. This allows the coverage of only a few topics selected from each chapter.

Finally we present a curriculum for an advanced Data Structures course. This presupposes that the student has already encountered the basic material, in particular the material on lists, trees, and graphs. Four weeks on advanced data structures gives the instructor enough time to cover all of the relevant topics in depth.

SEMESTER SCHEDULE - ADVANCED DATA STRUCTURES COURSE

Week	Subject	Reading Assignment
1	Review of Basic Material on Algorithms	Chapters 1-2
2	Review of Basic List structures	Chapters 3-4
3	Review of Trees	Chapter 5
4	Review of Graphs	Chapter 6
5	Review of Internal Sorting	Chapter 7 First program due
6	External Sorting	Chapter 7
7	External Sorting (continued)	
8	Hashing	Chapter 8 Second program due
9	Heap Structures (min-max heaps, deaps, leftist trees)	Chapter 9
10	Mid Term	
11	Heaps Structures (Fibonacci heaps)	Chapter 9
12	Search Structures (Optimal binary search trees)	Chapter 10
13	Search Structures (AVL trees, 2-3 trees, 2-3-4 trees)	Third program due
14	Search Structures (Red-black trees, splay trees, digital trees)	
15	Search Structures (B-trees, tries)	Fourth program due
16	Search Structures	

For schools on the quarter system, the following two quarter sequence is possible. It assumes prior exposure to algorithm analysis and elementary data structures at the level obtained from an advanced programming course.

QUARTER 1

Week	Subject	Reading Assignment
1	Review of algorithms and arrays	Chapters 1-2
2	Stacks and Queues	Chapter 3
3	Linked Lists (stacks, queues, polynomials)	Chapter 4
4	Linked Lists	
5	Trees (traversal, set representation)	Chapter 5 First program due

6	Trees (heaps, search)	
	Mid Term	
7	Graphs (traversal, components)	Chapter 6
8	Graphs (minimum spanning trees)	
9	Graphs (shortest paths)	Second program due
10	Graphs	(activity networks)

QUARTER 2

Week	Subject	Reading Assignment
1	Internal Sorting (insertion, quick, bound, O(1) space merging, merge sort)	Chapter 7
2	Sorting (heap, radix, list, table)	
3	External Sorting	Chapter 7
4	Hashing	Chapter 8
5	Mid Term	First program due
6	Heap Structures (deaps, Min-Max heaps, Leftist trees)	Chapter 9
7	Heap Structures (Fibonacci Heaps)	
8	Search Structures (AVL trees, 2-3 trees, 2-3-4 trees)	Chapter 10
9	Search Structures (Red-black trees, splay trees, digital trees)	Second program due
10	Search Structures (B-Trees, tries)	

Once again we would like to thank the people who have assisted us in debugging this edition. Thanks go to Professor Dinesh Mehta, University of Tennessee Space Institute, and to Mr. Seonghun Cho, Mr. Venkatramanan Narayanan, and Mr. Vishal Walia, University of Florida and to Ms. Penny Hull, Associate Managing Editor, W. H. Freeman.

Ellis Horowitz
Sartaj Sahni
September 1993

CONTENTS

CHAPTER 1

BASIC CONCEPTS

1.1 OVERVIEW: SYSTEM LIFE CYCLE

We assume that our readers have a strong background in structured programming, typically attained through the completion of an elementary programming course. Such an initial course usually emphasizes mastering the syntax of a programming language (its grammar rules) and applying this language to the solution of several relatively small problems. These problems are frequently chosen so that they use a particular language construct. For example, the programming problem might require the use of arrays or **while** loops.

In this text we want to move you beyond these rudiments by providing you with the tools and techniques necessary to design and implement large-scale computer systems. We believe that a solid foundation in data abstraction, algorithm specification, and performance analysis and measurement provides the necessary methodology. In this chapter, we will discuss each of these areas in detail. We will also briefly discuss recursive programming because many of you probably have only a fleeting acquaintance with this important technique. However, before we begin, we want to place these tools in a context that views programming as more than writing code. Good programmers regard large-scale computer programs as systems that contain many complex interacting parts. As systems, these programs undergo a development process called the system life cycle. This cycle consists of requirements, analysis, design, coding, and verification phases. Although we will consider them separately, these phases are highly interrelated and fol-

low only a very crude sequential time frame. The References and Selected Readings section lists several sources on the system life cycle and its various phases that will provide you with additional information.

(1) Requirements. All large programming projects begin with a set of specifications that define the purpose of the project. These requirements describe the information that we, the programmers, are given (input) and the results that we must produce (output). Frequently the initial specifications are defined vaguely, and we must develop rigorous input and output descriptions that include all cases.

(2) Analysis. After we have delineated carefully the system's requirements, the analysis phase begins in earnest. In this phase, we begin to break the problem down into manageable pieces. There are two approaches to analysis: bottom-up and top-down. The bottom-up approach is an older, unstructured strategy that places an early emphasis on the coding fine points. Since the programmer does not have a master plan for the project, the resulting program frequently has many loosely connected, error-ridden segments. Bottom-up analysis is akin to constructing a building from a generic blueprint. That is, we view all buildings identically; they must have walls, a roof, plumbing, and heating. The specific purpose to which the building will be put is irrelevant from this perspective. Although few of us would want to live in a home constructed using this technique, many programmers, particularly beginning ones, believe that they can create good, error-free programs without prior planning.

In contrast, the top-down approach begins with the purpose that the program will serve and uses this end product to divide the program into manageable segments. This technique generates diagrams that are used to design the system. Frequently, several alternate solutions to the programming problem are developed and compared during this phase.

(3) Design. This phase continues the work done in the analysis phase. The designer approaches the system from the perspectives of the data objects that the program needs and the operations performed on them. The first perspective leads to the creation of abstract data types, whereas the second requires the specification of algorithms and a consideration of algorithm design strategies. For example, suppose that we are designing a scheduling system for a university. Typical data objects might include students, courses, and professors. Typical operations might include inserting, removing, and searching within each object or between them. That is, we might want to add a course to the list of university courses or search for the courses taught by a specific professor.

Since the abstract data types and the algorithm specifications are language-independent, we postpone implementation decisions. Although we must specify the information required for each data object, we ignore coding details. For example, we might decide that the student data object should include name, social security number, major, and phone number. However, we would not yet pick a specific implementation for the list of students. As we will see in later chapters, there are several possibilities, including arrays, linked lists, or trees. By deferring implementation issues as long as pos-