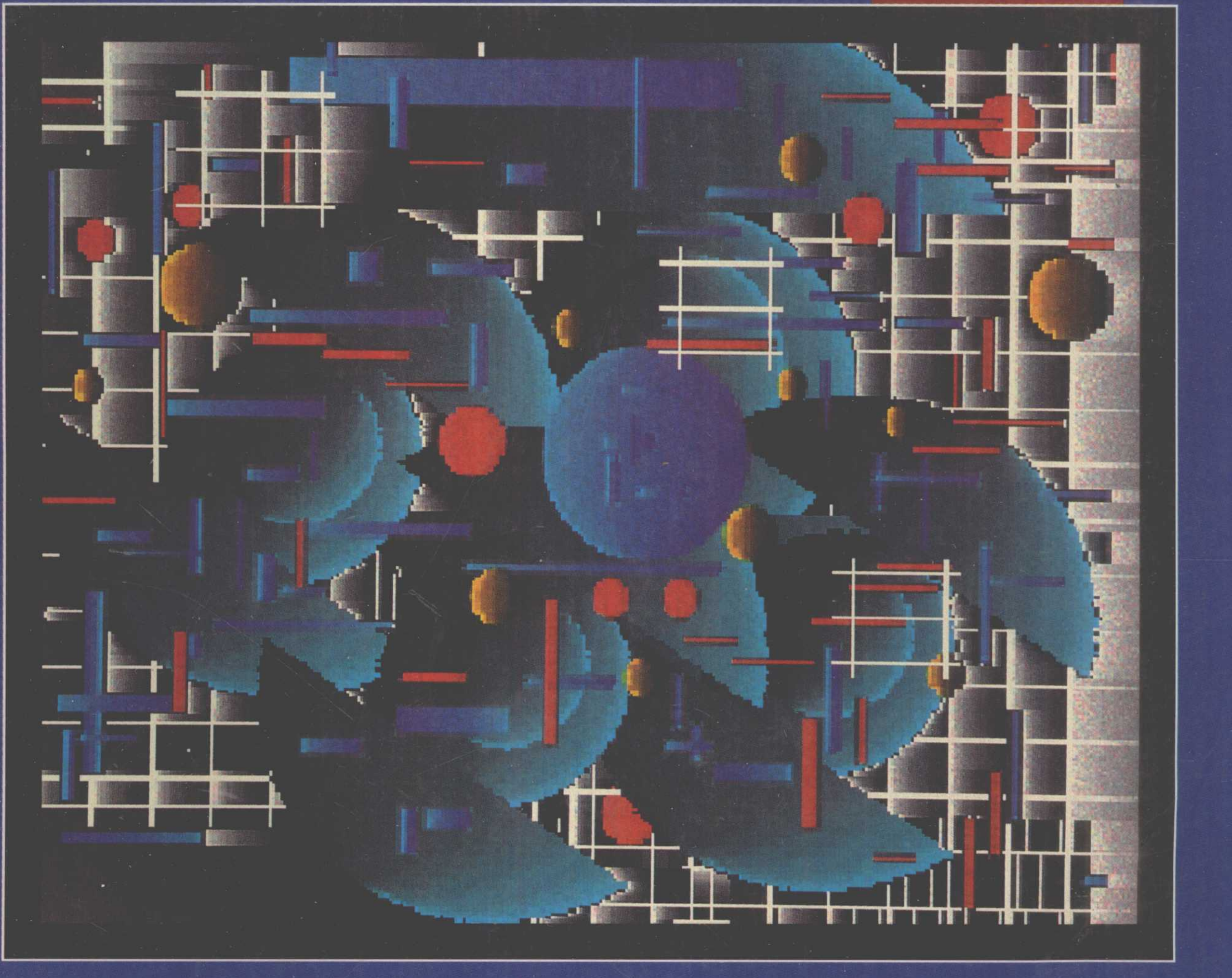
STRUCTURING DATA WITH TURBO PASCAL

A PRACTICAL INTRODUCTION TO ABSTRACT DATA TYPES



William G. McArthur ■■■■■ J. Winston Crawley

TP311.12/M116

STRUCTURING DATA WITH TURBO PASCAL

A PRACTICAL INTRODUCTION TO ABSTRACT DATA TYPES

William G. McArthur
J. Winston Crawley
Shippensburg University

PRENTICE HALL
Englewood Cliffs, New Jersey 07632

Library of Congress Cataloging-in-Publication Data

McArthur, William G.
Structuring data with Turbo Pascal : a practical introduction to abstract data types / William G. McArthur, J. Winston Crawley.
p. cm.

Includes bibliographical references and index.
ISBN 0-13-853052-1
1. Data structures (Computer scieance) 2. Pascal (Computer program language) 3. Turbo Pascal (Computer Program) I. Crawley, J. Winston. II. Title.
QA76.9.D35M4 1992
005.26'2—dc20 91-3717
CIP

Acquisitions Editor: Marcia Horton
Editor-in-Chief: Marcia Horton
Production Editor: Joe Scordato
Copy Editor: Barbara Zeiders
Design Director: Janet Schmid
Designer: Carla Weise / Levavi & Levavi
Cover Designer: Carla Weise / Levavi & Levavi
Prepress Buyer: Linda Behrens
Manufacturing Buyer: Dave Dickey
Supplements Editor: Alice Dworkin
Editorial Assistant: Diana Penha

Printed in the United States of America
10 9 8 7 6 5 4 3 2 1

ISBN 0-13-853052-1

Prentice-Hall International (UK) Limited, *London*
Prentice-Hall of Australia PTY, Limited, *Sydney*
Prentice-Hall Canada Inc., *Toronto*
Prentice-Hall Hispanoamericana, S.A., *Mexico*
Prentice-Hall of India Private Limited, *New Delhi*
Prentice-Hall of Japan, Inc., *Tokyo*
Simon & Schuster Asia Pte. Ltd., *Singapore*
Editora Prentice-Hall do Brasil, Ltda., *Rio de Janeiro*

PREFACE TO THE INSTRUCTOR

The teaching of data structures in the computer science curriculum is undergoing a transformation. Historically, the study of data structures considered such topics as linked lists, binary trees, stacks, and sorting. Increasingly, this traditional subject matter is being augmented by a study of Abstract Data Types (ADTs). This book is designed to aid in that transition, by combining the study of data structures and ADTs at a level appropriate for a second course in the curriculum.

With the increasing importance of Object Oriented Programming, this early emphasis on ADTs becomes ever more important. A good gounding, early in the curriculum, in the concepts involved in encapsulating ADTs in packages lays a solid foundation from which to move on to the Object Oriented Programming methodology.

With this goal in mind, we present each data type (stack, queue, etc.) at four levels:

1. *Abstract*. At this level we define the ADT. This includes identifying data type names and desired operations. The operations are encapsulated as Pascal subprograms, with the interface completely defined.
2. *Representation*. At least two representations are considered for each data type (one based on arrays, the other on linked structures). Algorithms are developed for each operation.
3. *Implementation*. At this level we develop Pascal code for the data declarations and the operations.
4. *Packaging*. This step involves putting the code into a package. The UCSD Pascal concept of a unit is used as a tool for this step.

To be successful at the level of the second course, some care is required in the presentation of ADTs. It is important to keep the material at an introductory level. Just as important, the presentation must be practical. As we view it, there are two important aspects to this practicality. First, the student must become aware of the practical value of the notion of an Abstract Data Type. We emphasize this by developing a variety of application programs that use the various data types. Some of these are "classical" applications, such as infix-to-postfix conversion. Others are drawn from our experience as consultants in software design and development. The second issue of practicality pertains to packaging. Through our examples we show the student exactly how to build a package for an ADT. This helps move the concept of ADT to a concrete level, and it provides code the students can emulate in developing their own programs.

ON THE PROGRAMMING EXAMPLES

We believe that textbooks should present programs that really work. With this in mind, we have developed and tested all the complete program examples of the book, using Turbo Pascal as the environment. The examples run correctly as presented, on Turbo Pascal 4.0 through 6.0. Unless Borland's compatibility philosophy changes, we expect that they will run correctly on later versions as well.

In an environment as powerful as Turbo Pascal (especially beginning with version 6.0), there are many non-standard features we could have incorporated in the examples. We tried, however, to concentrate always on the primary goal of the book, the study of Abstract Data Types and their representations, implementations, and packaging. In accordance with this goal, we make extensive use of Turbo Pasal's *unit* as a packaging tool. We also make free use of its built-in string data type. For most of the other non-standard features, we are of the opinion that emphasizing the features would detract from our ultimate goal of teaching how to use and develop Abstract Data Types.

HOW TO USE THE BOOK

The book is organized into three parts. Each part consists of two or three chapters, and each chapter contains three or four sections. A high-level outline of the contents follows.

PART I. PRELIMINARIES

Review material (Chapter 0)
Present some tools (Chapter 1):
- Arrays
- Records
- Simple linked lists
- Packaging ADTs

PART II. FOUR FUNDAMENTAL ABSTRACT DATA TYPES

Use the tools for four fundamental ADTs (Chapters 2 through 4)
- Stacks
- Queues
- Ordered lists
- Tables

PART III. MORE DATA STRUCTURES AND DATA TYPES

Expand the notion of linked lists (Chapter 5)
- Recursion with linked lists
- More complex linked lists (doubly linked, etc.)
- Strings as a case study

Develop the concept of a binary tree (Chapter 6)
- The tree as ADT
- Linked trees
- Recursion with trees
- Using trees to implement the table ADT

Within each chapter that develops an ADT, we use a uniform presentation. In the first section (two sections in Chapter 4) we present the abstract level, providing three simple applications using the ADT. In the next section we discuss representation and implementation of the ADT, and present a package for the ADT. We also describe some possible enhancements to the basic ADT. In the final section of each chapter one or two case studies are presented in the form of larger application programs that use the ADT.

Most courses using this book would probably go straight through the book from Chapter 1 to Chapter 6, covering some but perhaps not all of the case studies. This might be preceded by some material from Chapter 0, depending on the background of the students. If time permits, more case studies could be examined, or parts of Appendix A (Additional Topics) could be discussed in class.

Some instructors prefer to begin by looking at all the ADTs at the abstract level. Only after this has been done would representation and implementation be considered. The book would support this approach, simply by skipping over the middle section(s) of each ADT chapter. Because enhancements to the ADT are given in these middle sections, the instructor would have to describe some of these enhancements that are used in the case studies. (The supplementary material supplied to the instructor includes disks with complete code for all the packages discussed in the text. The students could write applications that use these packages, prior to studying the code that makes up the packages.)

FEATURES

Practical Introduction to Abstract Data Types (ADTs). In addition to theory, includes extensive practical guidance on how to develop and package ADTs, and *how to use them in application programs*. The text is rich in code; the reader will have ample examples of how to develop and use ADTs, including implementation in Pascal.

Four Levels. All ADTs are presented at four levels: abstract, representation, implementation, and packaging. The discussion of the various levels, specifically the guidance on packaging, reinforces the important concept of *information hiding*.

Abstraction. To aid the student in separating the abstract type from its implementation, separate sections are devoted to the abstract level. These sections include numerous exercises in *using* the ADT, prior to considering representation and implementation.

Software Engineering Model. Case studies and other examples are developed completely, including top-down design of algorithm, analysis of algorithm where appropriate, writing code, and testing and maintenance issues. This aids the students in understanding the examples and presents a model for their own program development.

Testing. Students are given extensive practical guidance in choosing appropriate test data for their programs. Almost every section of the text includes guidance specifically tailored to the current topic.

Case Studies. There are 7 case studies, consisting of carefully worked out applications using ADTs. The case studies are nontrivial (but not overly complex). The case studies include traditional topics and other topics drawn from the authors' experience.

Treatment of Sorting and Searching. These topics are integrated into the applications where appropriate. For example, bucket sort is discussed in the chapter on queues, and heap sort is described in the chapter on trees. (Appendix C contains a summary of the methods covered, with references to the pertinent sections.)

Programming Projects. Each of the 26 sections of the text contains approximately 4 to 8 programming projects. Some programming projects build on the case studies or other examples; others indicate new ideas for exploration by the student. Some are suitable for individual work; others are ideal as group assignments. Section 4-4 is an especially fruitful source of team programming assignments.

OTHER AIDS TO TEACHING AND LEARNING

Review Material. Chapter 0 contains coverage of material that may have been covered only marginally (or not at all) in the first course. This includes a thorough (two-section) discussion of recursion, an overview of program testing methods, and a review of scope. In addition, Appendix D contains a complete review of the Pascal language in the form of syntax diagrams.

Examples. There are many examples of various types. For each ADT there are three simple applications that use the ADT, plus one or two more extensive case studies that use the ADT. In addition, implementation code and packaging methods are shown for each ADT.

Exercises. In addition to the programming projects already descibed, each section of the text contains approximately 20 to 30 routine exercises. The exercises range in difficulty from trivial to moderate.

Answer Key. Answers are given for all odd-numbered exercises. (The instructor's supplementary material includes answers for the even-numbered exercises.)

Defensive Programming Tips. In almost every section of the text there are discussions of common misconceptions and errors, and how to avoid them (defensive programming).

Review Subsections. Every section closes with a review of the material discussed in that section.

Teaching Orientation. The text teaches the student, rather than just presenting material. A *spiral approach* is used where appropriate. For example, linked lists are introduced in Chapter 1 and used in several simple settings before continuing to a general discussion in Chapter 5.

Graphical Aids. Coloring and shading are used to highlight specific portions of examples and diagrams.

Other Appendices. These include a summary of the interface descriptions of the various ADTs discussed, a summary of the ADT implementations covered, using arrays for linked structures, sets, priority queues, hashing, and other topics.

MATERIALS FOR THE INSTRUCTOR

To aid the instructor, an extensive package of reference materials has been created. Portions of this package are in printed form as an instructor's guide, and portions are available on diskette. The printed material includes:

1. A summary of the material discussed for each section, with teaching suggestions.
2. Overhead transparency masters.
3. Solutions to even-numbered exercises.
4. Sample exams.

The electronic material includes:

1. All program examples from the text. This includes code for all case studies, and code for all ADTs developed in the text. This allows the instructor to assign meaningful projects of the form, "Modify the example in the text." It also allows the instructor to assign as student projects applications that use the ADTs at the abstract level.
2. The sample exams.
3. All the exercises from the text. This will provide an easy way for the instructor to create her own exams to supplement the sample exams provided.
4. The utilities from Appendix E of the text.
5. All the solutions from the textbook and the instructor's guide.

ACKNOWLEDGMENTS

We would like to acknowledge those who reviewed early versions of this text: Leon Levine of UCLA, Andrew Bernat of the University of Texas at El Paso, Emil C. Neu of Stevens Institute of Technology, Frank Gergelyi of Metropolitan Technical Institute, David J. Kroger of Miami University, Eileen Entin, Ivan B. Liss of Radford University, Lawrence H. Levine of Baruch College, CUNY, Ronald E. Bass of Austin Community College, Vernor Vinge of San Diego State University, John J. Cupak, Jr. of The Pennsylvania State University, Charles M. Williams of Georgia State University, and George Novacky of the University of Pittsburgh.

William G. McArthur
J. Winston Crawley

PREFACE TO THE STUDENT

The purpose of this book is to help you learn about data structures and Abstract Data Types. These are important tools that have applications in a broad range of areas. You will draw upon your experience in programming in Pascal to develop and use these tools. In this preface we want to point out some features of the book that may aid you in your study.

Syntax Diagrams. Appendix D contains a review of the Pascal language in the form of syntax diagrams. You can use that appendix now to give yourself a quick review of the language. Later, you can use it as reference if you cannot remember quite how to write the code you need.

Chapter 0. Chapter 0 reviews topics that are sometimes (but frequently not) covered in the first course. These include recursion, program testing, and scope. Your instructor may include parts of this chapter in the course material.

Examples. There are many examples of various types. Most of the examples are either individual procedures or functions, or fairly short complete programs. You should study these examples thoroughly, relate them to the material being discussed, and adapt their ideas to your own programs.

There are two other types of program examples which require a slightly different approach. First, there are some **reference** sections that follow the programming projects (see the end of Section 2-2, for example). These sections show how to "package" the code that has already been developed in the section. There is no need to restudy all the code. Instead, you should look at how the code is put together to form a "package." (We will discuss what a package is in Chapter 1.)

Second, there are **case studies** in the final section of Chapters 2 through 6. These are larger application programs that use the tools developed in the chapter. In studying them, you need to remember that you are trying to understand a complete large program that someone else has written. This always takes some effort, even for experienced programmers. Do not get discouraged if you have to go over parts of the development several times before you fully grasp the example.

Exercises. Each section of the text contains a number of exercises. Most of these are to be solved using pencil and paper rather than by writing and running a computer program. They range in difficulty from trivial to moderate. In the answer key we give solutions for all odd-numbered problems. Doing routine exercises is like practicing a sport—the more you do, the better you will be. In addition, by checking your solutions to the odd-numbered exercises, you can measure your mastery of the material in the section. (There are also Programming Projects designed to form the basis for programming assignments by the instructor.)

Defensive Programming Tips. In almost every section of the book we describe common misconceptions and errors and tell how to avoid them (defensive programming). Use these when you write your programs to cut down on both time and frustration.

Review Subsections. Every section closes with a review of the material discussed in that section. These can be used to identify areas where you are not sure of your mastery, and as study guides for tests.

Testing. Each section contains a discussion of program testing. These discussions give pointers on how to uncover bugs in your programs. The general ideas described in Chapter 0 are applied to each specific topic covered in the text. Following these guidelines will help ensure that the instructor will not find errors in the programs you turn in.

William G. McArthur
J. Winston Crawley

CONTENTS

0 REVIEW

In this chapter you will learn:

- How to use contour diagrams to determine the scope of variables.
- The significance of global and local variables.
- How to nest subprograms in Pascal.
- How to use recursion in numerical situations.
- How to write recursive solutions to nonnumerical problems involving arrays.
- How to trace recursive subprograms.
- The purpose of program testing.
- How to test a program incrementally (top down, bottom up, and combinations).
- How to choose test cases to maximize the effectiveness of your test plan.
- Specific testing guidelines for commonly used algorithms.

INTRODUCTION

IN THIS CHAPTER WE DISCUSS A NUMBER OF TOPICS THAT ARE SOMETIMES (BUT frequently not) included in a first programming course. Some or all of the material may be familiar to you. If so, you should profit from reviewing it. If the material is new to you, you should treat this as the starting point for the second course. All the material presented in this chapter is used and referred to at various places throughout the remainder of the book.

As you study the chapter, you may wish to pay attention to its organization. This chapter is typical of the rest of the book. Each chapter is divided into sections, and most sections contain certain features in common. Among these are:

a. *Defensive Programming Tips*. We point out common errors or misconceptions, and give tips on how to avoid pitfalls.
b. *Testing*. After we discuss general testing principles in Section 0-4, we apply these principles to the subject matter of nearly every section of the book.
c. *Review*. Nearly every section ends with a review subsection. Use these to help you see if you have understood the material in the section.

d. *Exercises*. These are problems requiring a short to medium-length solution. The solution may involve writing a piece of a program—perhaps a procedure or function. These exercises are intended to be done by hand, but you can certainly try out your solutions on the computer. In the answer key at the end of the book we provide solutions for all the odd-numbered exercises.
e. *Programming Projects*. These are exercises that involve writing complete computer programs. They are intended to be done on the computer. Your instructor may assign some of these, perhaps with additional or modified requirements.

This review chapter has three major topics. In Section 0-1 we consider the scope of variables and nested declarations of subprograms. Two sections are devoted to recursion. In Section 0-2 we describe some numerical applications that use recursion. This topic may be a review for many people. The follow-up, in Section 0-3, is more likely to be new material for most readers. In that section we illustrate how recursion can be applied to problems involving arrays. Finally, in Section 0-4 we describe how to test your programs to find and remove bugs.

0-1 SCOPE OF VARIABLES

In this section we discuss the concept of **scope** briefly as it relates to the Pascal programming language. This term, *scope*, deals with where in a program a particular variable or other identifier may be referenced. We begin with the simplest situations and conclude with a discussion of nested procedures.

Comment Throughout the section we refer to the term *procedure*. However, our discussion applies equally well to *functions*. ■

SIMPLE SCOPE: NO NESTED PROCEDURES

Consider the following sample program:

```
program NonNested1 (Input, Output);
var
   I, J           : integer;

procedure P1(X : integer; var Y : integer);
var
   Z              : integer;
begin  {P1}
   .   .   .
end; {P1}

procedure P2(M : integer);
var
   N, P          : integer;
begin   {P2}
   .   .   .
end;   {P2}

begin   {NonNested1}
   .   .   .
end.
```

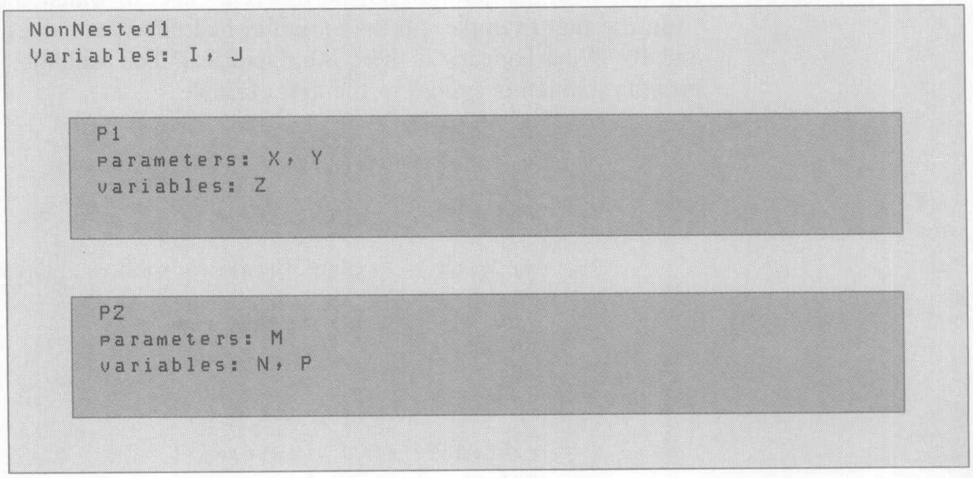
The program has several variables: I, J, M, N, P, X, Y, and Z. We can identify the scope for each of these; that is, we can identify the areas of the program in which we could legally refer to the variable.

I, J. These variables were declared in the main program. They may be legally referenced anywhere within the body of the main program. They may also be legally referenced anywhere within the body of the P1 or P2 procedure. Their *scope* is the entire program.

X, Y, Z. These were declared in the P1 procedure, either as parameters of the procedure or as local variables. They may be referenced only within the body of P1. Referring to X, Y, or Z within the body of P2 or within the main program is illegal. Their *scope* is the P1 procedure.

M, N, P. These are similar to X, Y, and Z. Their *scope* is the P2 procedure. Any reference to M, N, or P within P1 or within the main program is illegal.

A technique sometimes used to aid in understanding the scope of variables is a **contour diagram**. A contour diagram is a set of nested boxes symbolizing the various subprograms (and main program). For example, following is a contour diagram for the sample program.



To use a contour diagram to identify scope, we visualize the boxes as being made of one-way mirrors. The inside of the box is glass, so we can see out. The outside is a mirror, so we cannot see in.

For example, while the computer is executing the P1 procedure it can "see" the variables (X, Y, Z) within the P1 procedure. It can also look out through the glass and see the variables I and J defined in the main program. However, it cannot see into the P2 box, so the variables M, N, and P are not visible. Similarly, while the body of the main program is executing, only the variables I and J are visible. Any variables defined within the P1 or P2 boxes cannot be referenced.