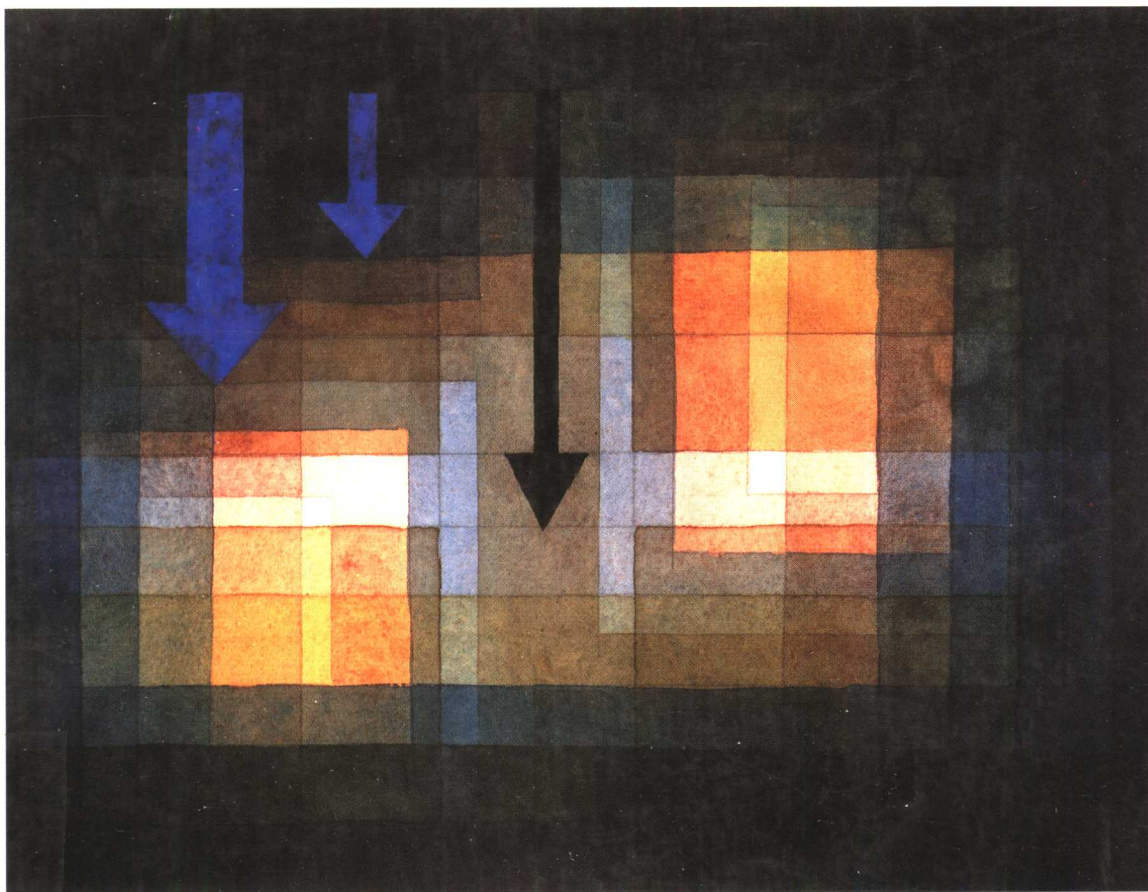


---

# STRUCTURES AND ABSTRACTIONS

An Introduction to  
Computer Science  
with Pascal

---



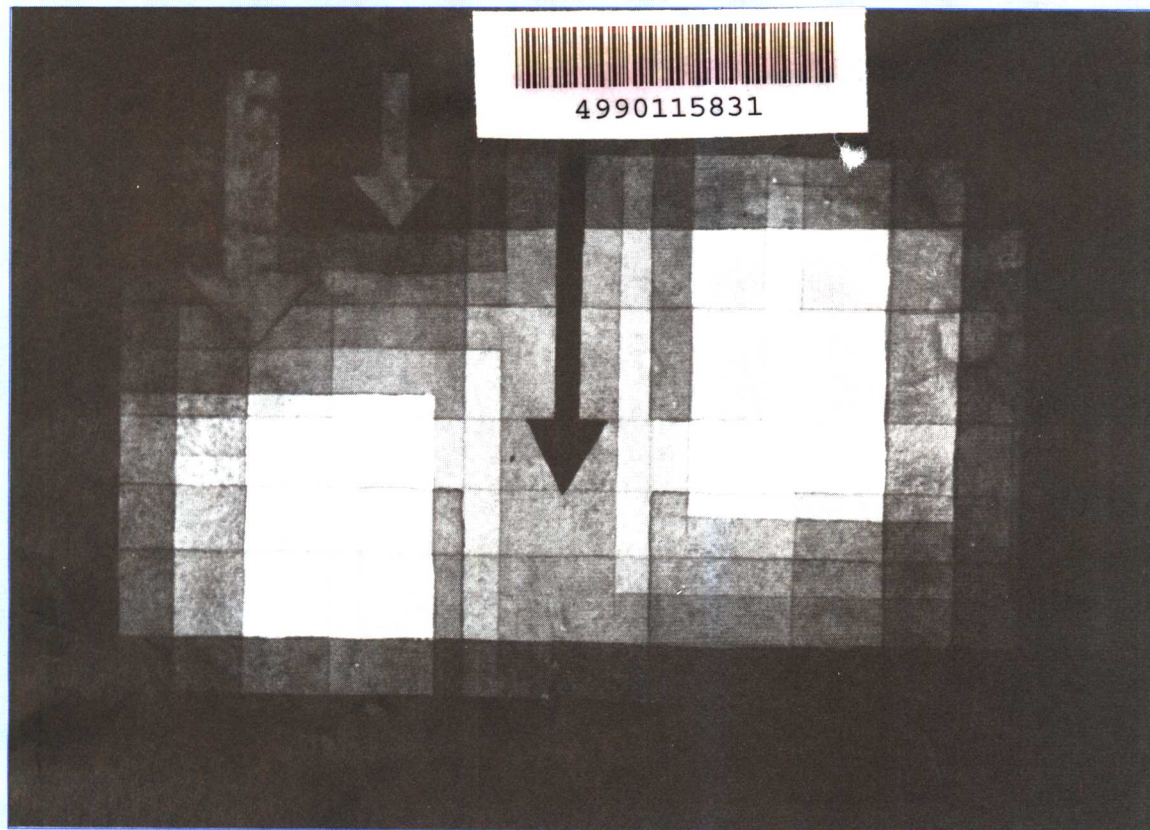
---

WILLIAM I. SALMON

0115831

# STRUCTURES AND ABSTRACTIONS

An Introduction to Computer Science with Pascal



WILLIAM I. SALMON

University of Utah

A GIFT OF  
THE ASIA FOUNDATION  
DISTRIBUTED BY  
SHANGHAI INTERNATIONAL STUDY  
UNIVERSITY LIBRARY

非  
賣  
品

IRWIN

Homewood, IL 60430

Boston, MA 02116

美國亞洲基金會  
上海外國語學院

73.9621  
5172  
872p 24cm  
\*



This symbol indicates that the paper in this book is made from recycled paper. Its fiber content exceeds the recommended minimum of 50% waste paper fibers as specified by the EPA.

## **Dedicated to the Spirit of Liberal Education, wherever it survives.**

© RICHARD D. IRWIN, INC., 1991

*All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the publisher.*

Cover painting: Paul Klee, *Submersion and Separation*, 1923  
8 3/4 x 11 (22.2 x 27.9); watercolor and gouache  
Collection of the Arts Club of Chicago, Arthur Heun Bequest  
Reproduced by permission.

Senior sponsoring editor: Bill Stenquist  
Project editor: Jane Lightell  
Production manager: Irene H. Sotiroff  
Designer and compositor: Bill Salmon  
Cover designer: Image House, Inc.  
Printer: R. R. Donnelly & Sons Company

The programs in this book have been included for instructional use only. They have been carefully tested but are not guaranteed for any particular use. The author and the publisher accept no liabilities for the use of the programs.

### **Library of Congress Cataloging-in-Publication Data**

Salmon, W. (William I.), 1942-  
Structures and abstractions: an introduction to computer science  
with pascal / William I. Salmon.  
p. cm.

Includes index.  
ISBN 0-256-08273-1

1. Pascal (Computer program language) 2. Data structures  
(Computer science) 3. Abstract data types (Computer science)  
I. Title.

QA76.73.P2S25 1991

530.1'2—dc20

90-26522

*Printed in the United States of America*

3 4 5 6 7 8 9 0 DOC 8 7 6 5 4 3 2

## Preface

A first course in a subject should convey its spirit and ways of thought. In computer science, the first course should convey models of computation, design and engineering methods, and criteria for correctness and usefulness. These should be emphasized from the beginning because they are the scientific and engineering foundations of the subject. It isn't enough to teach a particular programming language; this would leave students without the tools for solving today's complex problems. Considerable classroom experience with a wide range of students has led me to several conclusions:

- Students should write modular programs from the very beginning of the course. When algorithms are presented from a hierarchical viewpoint, modular design can be introduced early and emphasized consistently. There is no need to unlearn bad habits or to apologize for early efforts.
- Modular programming is not difficult for students who concentrate on it without distractions. Therefore, modularity should be covered before control structures and most data types.
- Students need *explicit* techniques for problem solving. When explicit techniques are seen frequently in a variety of programming situations, they become standard “templates” for problem solving, and a foundation for creative solutions to new problems.
- The abstractions of computing don't lend themselves very well to verbal descriptions. Data structures are visual and algorithms have motion, and students must be able to *picture* both the data structures and the actions that occur in a program.
- Good programs don't work correctly by luck; they are engineered so that they *have* to work. Beginners need simple techniques to guarantee the correctness of algorithms before writing programs.



- Recursion is less familiar than iteration, but no more mysterious. These two techniques for repetition are equally important and each illuminates the other. Therefore, recursion and iteration should be taught side-by-side and compared frequently.

## Coverage

This book is intended for introductory computer science courses emphasizing modern software engineering practice. The illustrative programming language is Pascal, but Pascal is not the main subject of the text. The main topics are modularity, hierarchy, abstraction, verification, and analysis, as tools for computational problem solving. These are emphasized constantly, from the very beginning.

*Structures and Abstractions* satisfies curriculum recommendations of the Association for Computing Machinery (ACM) and the Institute of Electrical and Electronic Engineers (IEEE) for a first course in computer science (CS1). It includes all material recommended by the ACM for CS1 courses and also introduces material recently recommended by the ACM/IEEE Joint Curriculum Committee for future introductory courses and by Norman Gibbs and Allen B. Tucker, Jr., for the introductory course in liberal arts curricula. Parts Four and Five of the book overlap the recommended ACM/IEEE curriculum for CS2 (data structures) courses. A sequel to the present book, to be entitled *Abstractions and Analysis*, will be designed specifically for CS2 courses.

The present book provides enough material for a variety of introductory courses, ranging from a single quarter to two semesters in length. There is considerable optional material to allow for various approaches in different teaching situations. A dependency diagram at the end of this preface shows various paths through the book.

The standard edition of *Structures and Abstractions* uses (except in one appendix) only ISO/ANSI Standard Pascal. All examples have been tested in several typical Pascal environments to ensure that they run correctly. A separate edition is available for those preferring a treatment specific to Borland International's Turbo Pascal®.

## Prerequisites

The reader is not assumed to have prior programming experience, but should be computer-literate, with enough experience in mathematics to appreciate the need for rigorous thought and to understand algebraic proofs. The material has been tested with considerable success on several hundred freshmen and sophomores at the University of Utah over a period of several years. I have found that the best predictors of success in the CS1 course are skill in mathematical proofs and word problems and an ability to communicate clearly in writing.

---

## Specific Features

**Design and analysis:** *Structures and Abstractions* places heavy emphasis on the fundamental techniques for proper software engineering, including top-down modular design, bottom-up testing, procedural and data abstraction, the use of assertions and loop invariants, and elementary running time analysis. Modular, hierarchical design is constantly emphasized, beginning in Chapter 3 with the very first program. Chapter 4 introduces I/O procedures for practice in using parameters. Chapters 5 and 6 then provide a complete treatment of procedures and functions with both value and variable parameters. There are hardly any monolithic programs in the book.

**Style:** The book itself reflects the rules of good programming style. For example, subjects like modularity, procedural abstraction, data abstraction, recursion, loop invariants, and dynamic data structures are introduced by themselves in their own modules, so that the reader can focus full attention on them. Then each topic is learned “bottom up,” with a sequence of stepped examples that gradually increase in complexity and abstraction. Like the programs themselves, chapters are short and modular, divided into short sections with clear goals. The modular design of the book makes for more chapters, sections, and subsections, but clearer reading without added length.

**Explicit problem-solving techniques:** Classroom testing has proven that students benefit from the presentation and repeated use of explicit methods for problem solving. Chapter 2 presents eight of the most generally useful. These techniques are illustrated with a nontrivial example in Chapter 2, then used repeatedly throughout the following chapters. By seeing the same techniques applied in different situations, readers gradually learn to use them.

**A spiral, not a “peek-a-boo” approach:** Students need to see why the rules of practice and style are necessary. *Structures and Abstractions* introduces techniques when they are needed, then uses them persistently and consistently. The reader never loses sight of a topic while seeing it unfold, because an important topic is never dropped after being introduced. The gradually deepening and continual exposure to abstraction deepens readers’ understanding and appreciation of each topic.

## Pedagogical Aids

**Chapter outlines and summaries:** Each chapter begins with a few introductory paragraphs connecting the chapter with previous material and outlining the topics to be covered. The chapter ends with a quick summary of the main points that were made.

**Visualization:** Sequences of execution during procedure calls, iteration, recursion, and other complex actions are animated by sequences of diagrams. Altogether, there are more than 35 such animations, containing more than 185 diagrams. In addition, there are hundreds of diagrams of syntax and data structures—over 370 in all.

**Exercises and programming projects:** Questions and exercises are distributed throughout the chapters, after material requiring practice. Some of these exercises are puzzles that simulate typical debugging experiences. Major programming projects are found at the ends of the chapters. Altogether, there are more than 650 problems, occupying 140 pages.

**Applications:** Important programming issues are illustrated by application to typical problems in computer science. These include first glimpses of many of the kinds of problems to be encountered in later courses.

**Debugging aids:** Debugging techniques and examples are discussed frequently. In addition to standard techniques involving modular testing and intermediate output, typical interactive debugging tools are described. Always, the reader is reminded that it is most important to prevent bugs in the first place, by good engineering practices.

---

## Using the Book in Class

Few CS1 courses will be able to cover all the material in this book, so advanced topics are presented in a modular fashion that allows for customization.

**Recursion** takes a long time to sink in, so it is introduced in Chapter 11 and revisited frequently, with heavy use of animation diagrams like those used earlier for nested procedure calls. Iteration and recursion are treated as equally important techniques for repetition, and often both versions of an algorithm are examined. Instructors preferring to deemphasize recursion in their courses can skip some of these sections.

**Abstract data types:** Data abstraction is introduced in Chapter 14, right after arrays. Beginning with Chapter 15 on records, abstract data types are used in more complex applications involving strings, graphics, linked lists, trees, stacks, and queues. For those preferring to implement ADTs by means of units or include files, Appendix H covers these techniques. Instructors preferring to skip ADTs can skip Chapters 14, 19, and 20 and cover only the early sections in Chapters 15–17 plus Chapter 18.

**Informal verification:** Pre- and postconditions are used wherever appropriate after Chapter 5, which introduces procedures with parameters. Loop invariants appear in Chapter 10 and are used thereafter. Yet the major discussions of these topics are confined to sections that can be skipped or deemphasized at the instructor's discretion.

**Complexity:** Informal running time analysis is first mentioned in Chapter 10 in connection with nested loops, and thoroughly discussed in Chapter 18 in connection with algorithms for searching and sorting. The requisite mathematics is integrated with the material. Instructors preferring to skip such topics can do so without losing continuity.

**Classic algorithms:** A first course should introduce its students to many of the classical algorithms on which later courses will build. This book includes such algorithms as base conversion, case mapping, counting characters and words in text, exponentiation, greatest common divisor, square roots, Towers of Hanoi, expression parsing, line drawing, string manipulations, binary search, selection sort, quicksort, list processing, evaluation of postfix expressions, pseudorandom number generation, and simulation.

---

## Teaching Aids

An instructor's manual is available, containing

- Tips on teaching the course.
- Sample course outlines.
- Solutions to exercises and projects.
- A bank of exam questions.

**Instructor's Manual**

In computer science, much more is learned in front of the machine than in reading a book. Therefore, a number of major applications illustrating the material in this book appear in a separate Laboratory Manual, for use in a supervised, interactive computer lab or in self-study. The manual guides students through such projects as greatest

**Lab Manual**



common divisor, Turing machines, cellular automata, Eliza and the Turing test, turtle graphics, curve and surface plotting, fractals, comparisons of sorting methods, and the construction of a simple editor.

---

## Acknowledgments

I am deeply indebted to many creative teachers who critiqued several drafts of the book. Their detailed and thoughtful suggestions along the way have had a profound effect on the final version:

Anthony Q. Baxter, University of Kentucky  
David Alan Bozak, SUNY College at Oswego  
Robert A. Christiansen, University of Iowa  
Denis A. Conrady, University of North Texas  
Cecilia Daly, University of Nebraska–Lincoln  
Douglas Dankel II, University of Florida  
Edmund I. Deaton, San Diego State University  
George F. Luger, University of New Mexico  
Michael G. Main, University of Colorado at Boulder  
Andrea Martin, Louisiana State University  
Kenneth L. Modesitt, Western Kentucky University  
George A. Novacky, Jr., University of Pittsburgh  
David L. Parker, Salisbury State University  
Theresa M. Phinney, Texas A&M University  
V. S. Sunderam, Emory University  
Stephen F. Weiss, University of North Carolina–Chapel Hill

In addition, the following reviewers deserve thanks for reviewing either the original proposal or the final manuscript:

Robert B. Anderson, University of Houston  
Brent Auernheimer, California State University, Fresno  
Louise M. Berard, Wilkes College  
Larry C. Christensen, Brigham Young University  
H. E. Dunsmore, Purdue University  
Suzy Gallagher, University of Texas  
David Hanscom, University of Utah  
Robert M. Holloway, University of Wisconsin–Madison  
Ronald P. Johnson, Evangel College  
William E. McBride, Baylor University  
Jane Wallace Mayo, University of Tennessee–Knoxville  
David Phillips, University of Pennsylvania  
Laurie White, Armstrong State College  
Stephen G. Worth III, North Carolina State University–Raleigh  
Marvin Zerkowitz, University of Maryland

Seldom has an author been blessed with such fine and thorough reviewers.

Many teaching assistants and students have helped with the evolution of this book. I would particularly like to thank my teaching assistants, Rich Thomson, Cliff Miller, Elena Driskill, Rory Cejka, Mark Ellens, Mike Stephenson, and Lynn Eggli. I would also like to thank all the students who made suggestions and corrections, especially Alexander Kratsov, Randy Veigel, Mark Nolan, Blair Brandenburg, Ian Adams, and David Swingle. Over the years, I have received many helpful suggestions from John Halleck and LeRoy Eide, of the University of Utah Computer Center and from my wife, Lydia Salmon. They too deserve effuse thanks.

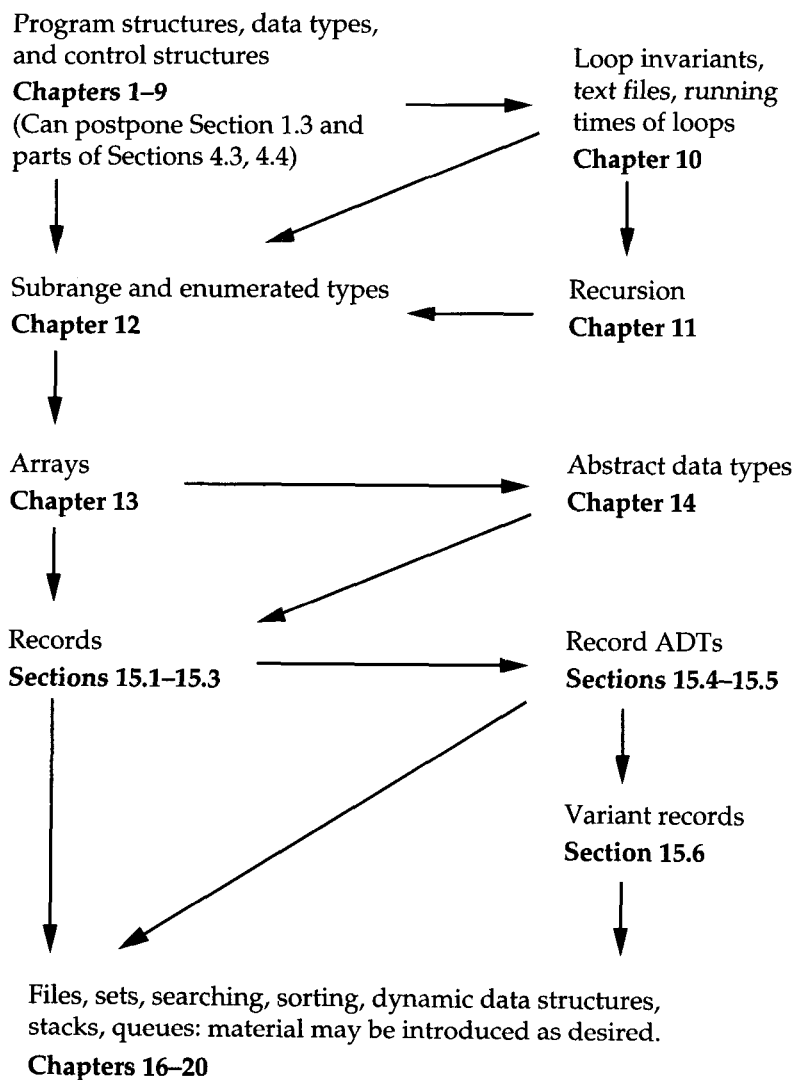
For many years, I have received encouragement and inspiration from Dave Hanscom, the Undergraduate Coordinator in the University of Utah's Computer Science Department. I also owe a special debt to my son, Edward Salmon, whose superb sense of design influenced the page design, the cover, and several of the diagrams and projects in this book.

Many thanks to the talented people at Richard D. Irwin, Inc., who provided more support and help than I thought possible. I particularly thank Bill Stenquist, the sponsoring editor, who said the turtle's head was too small but supported me all the way, and Sheila Glaser and Max Effenson, the developmental editors, who cheered me along while pointing out again and again that "but" is a conjunction.

But I should also mention Jackson P. Slipshod, who makes frequent appearances in the questions and exercises in this book. To the best of my knowledge, he made his first appearance in a chemistry book by Joseph Nordmann, published many years ago by John Wiley & Sons, Inc. Thanks to Dr. Nordmann's fine book, Jackson has been dogging me ever since.

William I. Salmon

## Sequencing of Material



# Contents

---

## PART ONE: COMPUTING

---

### 1. Computers and Computation

Introduction	2
1.1 Algorithms, Machines, and Programs	3
Questions	7
1.2 Computer Systems	8
Questions	15
1.3 Data Storage	16
Exercises	23
1.4 Programs	24
1.5 Editing, Compiling, and Executing a Pascal Program	28
1.6 The Need for Software Engineering	31
Looking Back	34
Questions and Exercises	35
References for Further Study	36
Terms Emphasized in Chapter 1	37

### 2. Explicit Techniques for Problem-Solving and Algorithm Design

Introduction	38
2.1 Clarify the Problem	39
2.2 Plan the User Interface First	39
2.3 Divide until Trivial; Then Conquer	41
2.4 Top-Down Design	41
2.5 Use Diagrams	45
2.6 Build on Previous Work	46
2.7 Structure the Data	47
2.8 Don't be Satisfied with Your First Idea	47
Looking Back	48
Exercises	49
References for Further Study	51
Terms Emphasized in Chapter 2	52

---

## PART TWO: PROGRAM STRUCTURES AND DATA TYPES

---

### 3. Program Structures

Introduction	54
3.1 A Pascal Program	55
Questions and Exercise	61

3.2	Syntax Diagrams	62
	Questions	64
3.3	Identifiers	65
	Questions and Exercises	66
3.4	Modularity and Hierarchy	67
	Exercises	73
3.5	Hierarchical Design	74
	Question and Exercises	77
3.6	Testing and Debugging	78
	Exercises	81
	Looking Back	82
	References for Further Study	83
	Terms Emphasized in Chapter 3	84
 <b>4. Real Data and I/O</b>		
	Introduction	85
4.1	Real-Number Data	86
	Questions and Exercises	93
4.2	Roundoff Error	95
	Questions and Exercises	97
4.3	Using Parameters with Write and WriteLn	98
	Questions and Exercises	105
4.4	Using Parameters with Read and ReadLn	108
	Questions and Exercises	115
4.5	Designing an Interactive Program	117
	Exercise	122
4.6	Testing and Debugging	123
	Looking Back	125
	Programming Projects	127
	References for Further Study	128
	Terms Emphasized in Chapter 4	129
 <b>5. Procedures</b>		
	Introduction	130
5.1	Sending Parameters to a Procedure	131
5.2	Procedures with Value Parameters	133
	Questions and Exercises	146
5.3	Procedures with Variable Parameters	148
5.4	A Procedure with Both Value and VAR Parameters	151
	Questions and Exercises	154
5.5	Program States; Pre- and Postconditions	156
5.6	Application: A Procedure to Swap Two Values	161
	Question	164

5.7	Global and Local Scope	164
5.8	Nested Procedure Calls; Local Procedures	171
	Questions and Exercises	178
5.9	Testing and Debugging	181
	Looking Back	184
	Programming Projects	185
	References for Further Study	186
	Terms Emphasized in Chapter 5	187
 <b>6. Functions</b>		
	Introduction	188
6.1	Calling a Function	190
6.2	Declaring a Function	192
6.3	When Do We use a Function?	200
	Questions and Exercises	201
6.4	Standard Functions in Pascal	203
	Exercise	204
6.5	Testing and Debugging	205
	Looking Back	206
	Programming Projects	206
	References for Further Study	208
	Terms Emphasized in Chapter 6	208
 <b>7. Ordinal Data Types</b>		
	Introduction	209
7.1	Ordinal Data	211
7.2	Integer Data	211
	Questions and Exercises	218
7.3	Application: Displaying Integers in Binary	219
	Exercise	226
7.4	Character Data	222
7.5	Application: Mapping Lowercase letters to Uppercase	230
	Question and Exercises	232
7.6	Boolean Data	233
	Exercises	241
7.7	Application: Character-Property Functions	243
	Exercises	246
7.8	Operator Precedence	247
	Exercises	249
	Looking Back	250
	Programming Projects	251
	References for Further Study	252
	Terms Emphasized in Chapter 7	253



---

**PART THREE: CONTROL STRUCTURES**

---

**8. Decision Structures**

Introduction	256
8.1 Sorting Three Numbers	257
8.2 Conditional Execution	259
Exercises	267
8.3 Two-Way Decisions	269
Exercises	272
8.4 Nested Decisions	273
Exercises	278
8.5 Application: Menus	280
8.6 CASE Structures	281
Question and Exercises	284
8.7 Testing and Debugging	285
Looking Back	286
Programming Projects	287
References for Further Study	289
Terms Emphasized in Chapter 8	290

**9. Repetition by Iteration**

Introduction	291
9.1 The WHILE Structure	292
9.2 Application: Counting Characters in Input	293
Questions and Exercises	300
9.3 Fenceposts, Semicolons, and Infinite Loops	301
9.4 Counter-Driven WHILE Structures	304
Exercises	306
9.5 FOR Structures	309
9.6 Using FOR Structures	317
Questions and Exercises	319
9.7 Application: Monte Carlo Calculation of $\pi$	321
9.8 REPEAT..UNTIL Structures	327
Questions and Exercises	329
9.9 Testing and Debugging	332
Looking Back	334
Programming Projects	334
References for Further Study	336
Terms Emphasized in Chapter 9	337

**10. Iteration by Design**

Introduction	338
10.1 Designing for Correctness: Loop Invariants	339
Questions and Exercises	347

10.2 Application: Raising a Real Number to an Integer Power	349
Questions and Exercises	355
10.3 Application: The Greatest Common Divisor	356
Exercises	362
10.4 Application: Square Roots by an Iterative Method	363
Questions and Exercises	369
10.5 Nested Loops and Running Times	371
Questions and Exercises	376
10.6 Text Files Revisited	379
Questions	383
10.7 Reading and Writing Text Files	383
Questions and Exercises	392
Looking Back	394
Programming Projects	394
References for Further Study	397
Terms Emphasized in Chapter 10	398
<b>11. Repetition by Recursion</b>	
Introduction	399
11.1 The Power of Recursion	400
11.2 The Structure of a Recursive Subprogram	405
Questions	406
11.3 Iteration versus Recursion	406
Exercises	412
11.4 Recursion for Square Roots	412
Question and Exercises	417
11.5 The Towers of Hanoi	418
Question	423
11.6 Tail Recursion	423
11.7 When <i>Not</i> to Use Recursion	426
Looking Back	428
Programming Projects	428
References for Further Study	430
Terms Emphasized in Chapter 11	430

---

## PART FOUR: DATA STRUCTURES AND DATA ABSTRACTIONS

---

<b>12. Programmer-defined Types</b>	
Introduction	432
12.1 Subrange Types	433
12.2 Advantages of Subrange Types	437
Questions and Exercises	441

12.3 Enumerated Types	443
Questions	446
12.4 Application: A State Variable in Parsing	446
Questions	451
Looking Back	452
Programming Projects	453
References for Further Study	453
Terms Emphasized in Chapter 12	454
<b>13. Arrays</b>	
Introduction	455
13.1 One-Dimensional Arrays	457
13.2 Storing Values in an Array	462
Exercises	464
13.3 Pitfalls with Arrays	468
Exercises	476
13.4 Parallel Arrays	479
Exercises	487
13.5 Multidimensional Arrays	489
Questions and Exercises	494
13.6 Application: A Payroll Program	496
Looking Back	501
Programming Projects	502
References for Further Study	509
Terms Emphasized in Chapter 13	510
<b>14. Arrays as Abstract Data Types</b>	
Introduction	511
14.1 String Arrays	512
Exercises	516
14.2 Abstract Data Types	517
Exercises	520
14.3 A String Abstract Data Type	520
Exercises	538
14.4 Turtle Graphics	541
Question and Exercises	564
Looking Back	566
Programming Projects	567
References for Further Study	572
Terms Emphasized in Chapter 14	573
<b>15. Records</b>	
Introduction	574
15.1 Records	575
Questions and Exercises	581