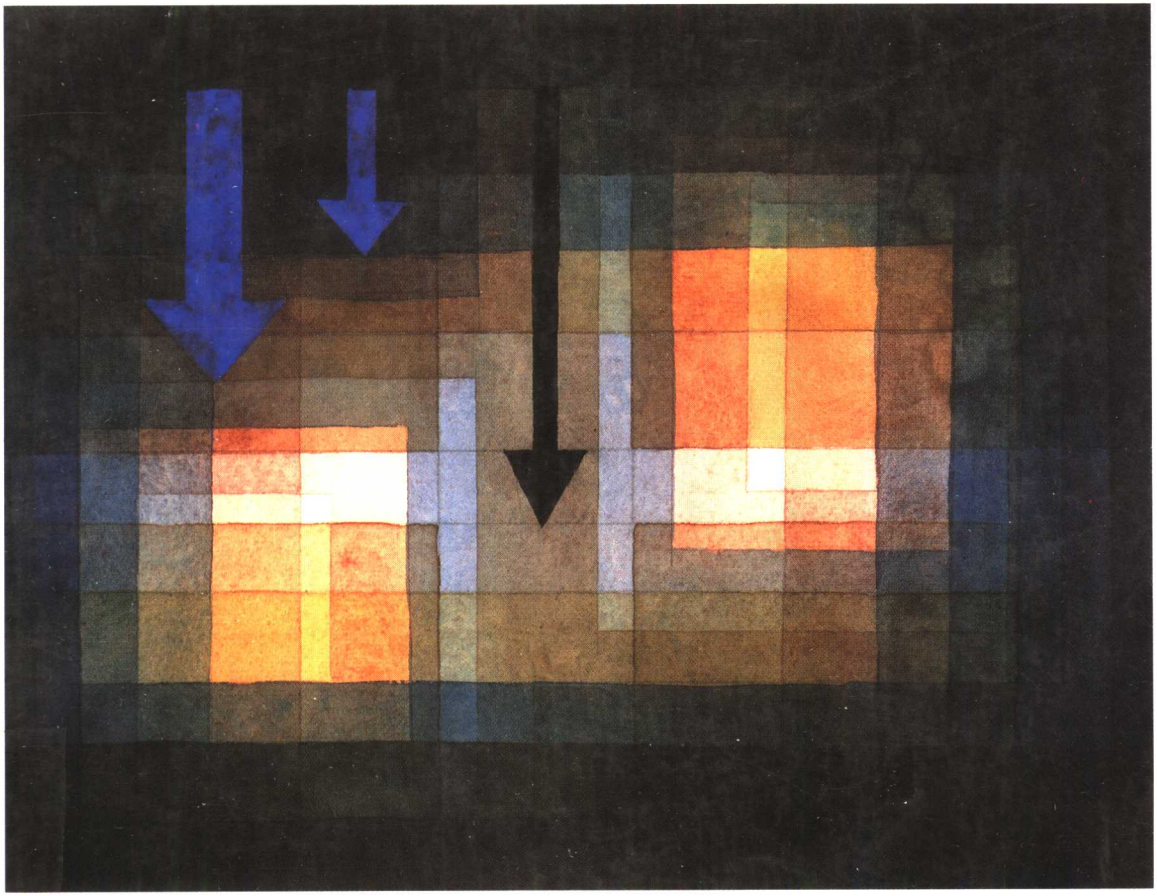
STRUCTURES AND ABSTRACTIONS

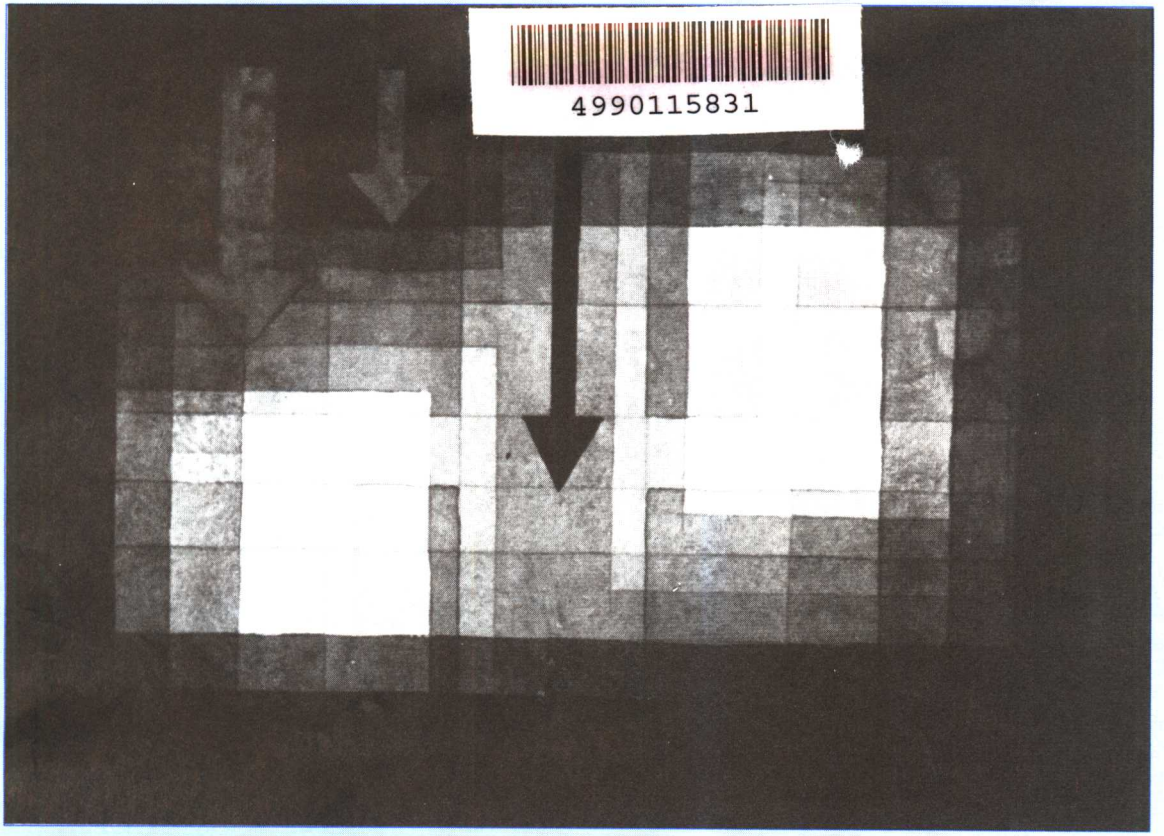
An Introduction to Computer Science with Pascal



WILLIAM I. SALMON

STRUCTURES AND ABSTRACTIONS

An Introduction to Computer Science with Pascal



WILLIAM I. SALMON
University of Utah

IRWIN
Homewood, IL 60430
Boston, MA 02116

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

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

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

Instructor's Manual

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.

Lab 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

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 Zelkowitz, 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 Brandenberg, 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.

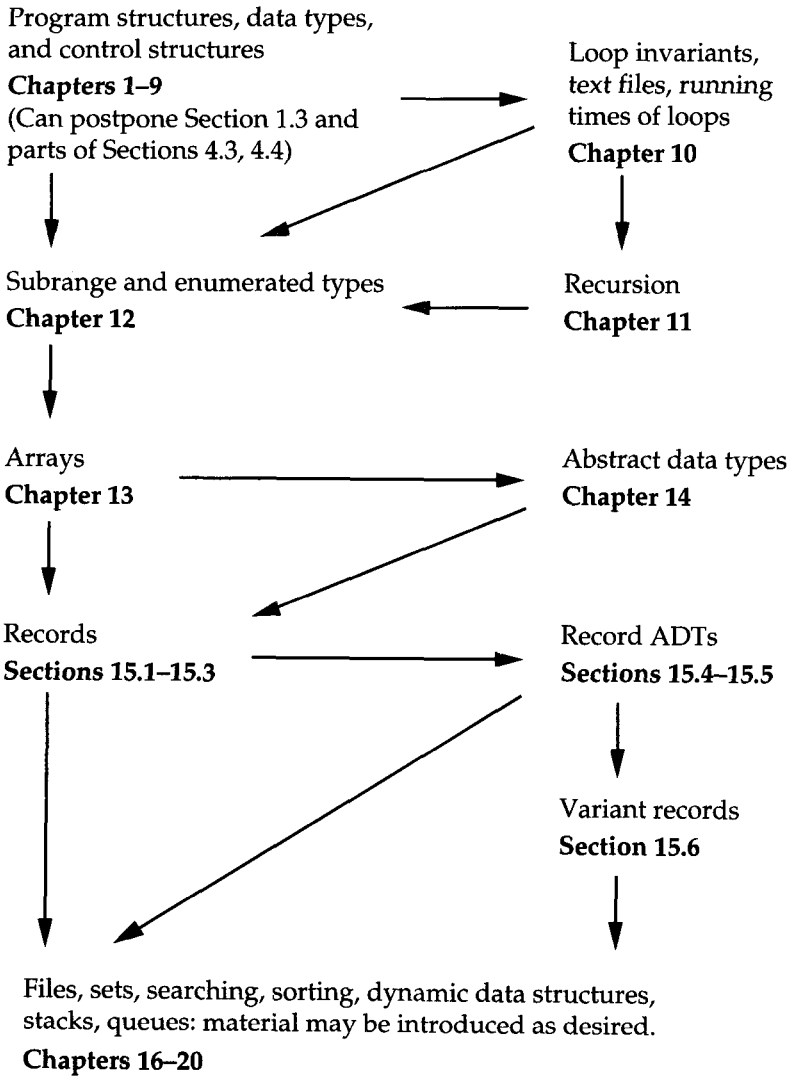
For many years, I have received encouragement and inspiration from Dave Hanscom, the Undergraduate Coordinator in the University of Utah's Computer Science Department. I also owe a special debt to my son, Edward Salmon, whose superb sense of design influenced the page design, the cover, and several of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