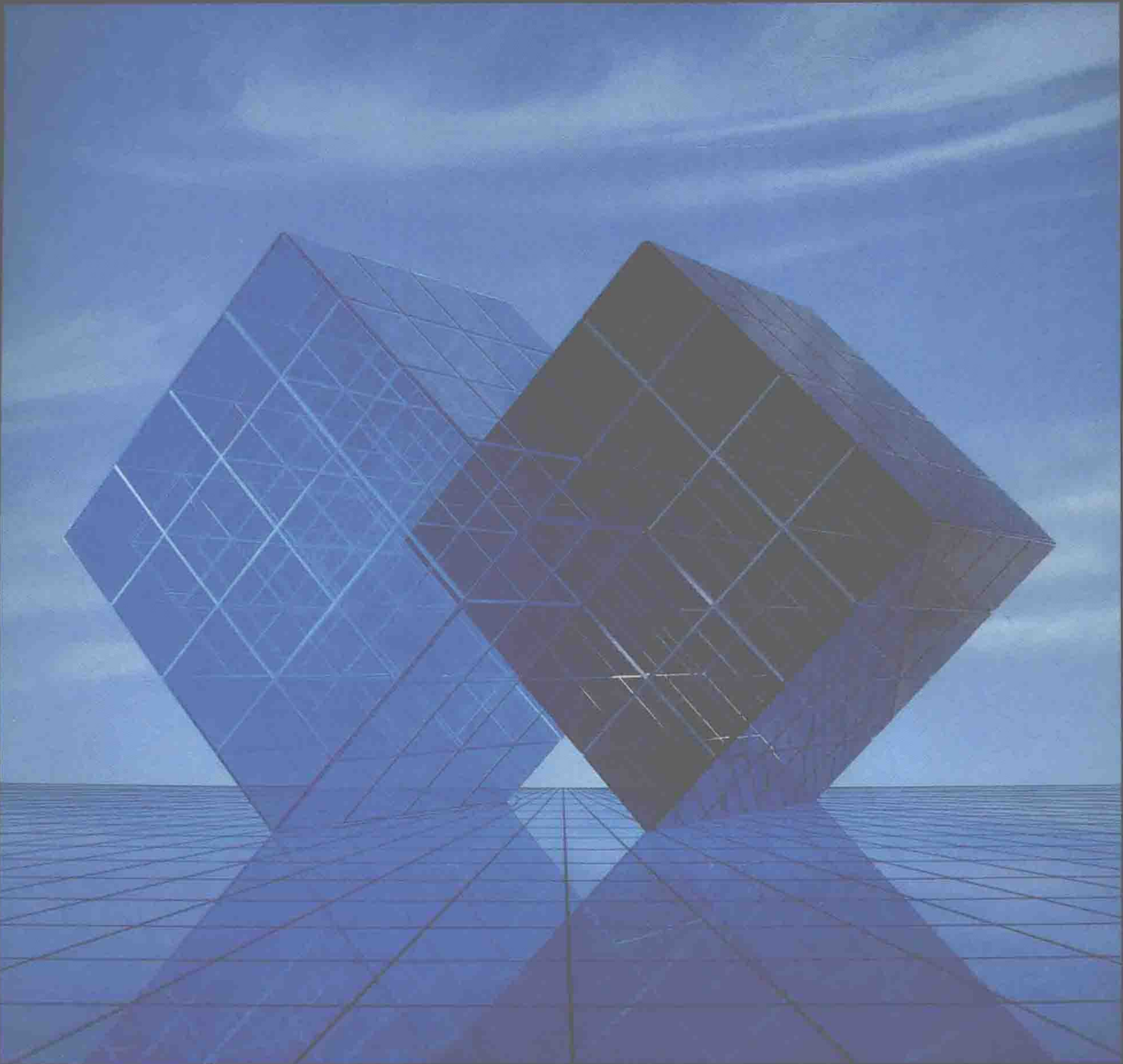
FUNDAMENTAL PROGRAMMING WITH PASCAL



J. DENBIGH STARKEY
ROCKFORD J. ROSS

Fundamental Programming With Pascal

J. Denbigh Starkey
Washington State University

Rockford J. Ross
Montana State University

West Publishing Company
St. Paul • New York • Los Angeles • San Francisco

Cover photography by Michel Tcherevkoff for Alleghany.

Printed in the United States of America

Library of Congress Cataloging in Publication Data

Starkey, J. Denbigh.
Fundamental programming with Pascal.

Bibliography: p.
Includes index.
1. PASCAL (Computer program language) I. Ross, Rockford. II. Title.
QA76.73.P2S73 1984 001.64'24 84-2244
ISBN 0-314-77806-3

Fundamental Programming With Pascal

To Susan
and
To Heidi, Jason, and Jennifer

Preface

Our purpose in writing this book was to present an introduction to programming in true textbook form. What we wanted was a textbook that could be used by students and instructors alike with the confidence that the issues fundamental to programming were covered in a thorough and integrated fashion. We wanted to ensure that students were not left dangling by incomplete discussions or topics devoid of application, and that instructors would not be forced to refer to other sources to fill in gaps left in the book. What we did *not* want was yet another Pascal programming language manual (many fine Pascal manuals already exist) or a book that only discussed the fundamental issues of programming in isolated sections without integrating these concepts into the everyday development of programs. Learning the details of a programming language like Pascal is relatively easy; the challenging part is learning how to program.

In striving towards our goals we have held to one basic tenet: students learn best by example. Each new programming concept introduced is discussed as it is needed within the framework of a complete top-down development of a program. Related issues, such as program efficiency and correctness, are integrated within this discussion rather than being relegated to isolated sections. Thus, this is a large book, not because it encompasses so much new material, but because of the novel and thorough presentation of the fundamental topics.

PHILOSOPHY

In determining the content of the book we had a definite pedagogical model in mind. This model stemmed from the startling observation that when students were asked to design a substantial program independently, many would return with a program consisting primarily of one or two large routines with little modularity — programs that were difficult to read and difficult to modify. Furthermore, when asked simple questions about the efficiency or correctness of their programs, the students were often at a loss for answers. This was true in spite of our efforts at teaching structured design, correctness, and efficiency of programs. What had gone wrong? The answer was surprisingly simple: we weren't practicing what we preached. Traditional textbooks used in the introductory courses either did not cover these topics well or they covered them in isolated sections of the text. Furthermore, procedures and functions were normally introduced late in these textbooks, almost as an afterthought, as the "right way" to program. Students were mistakenly led to believe that procedures and functions were difficult topics of more bother than they were worth; it's no wonder that they were avoiding their use later. In designing our book, then, our philosophy was to introduce programming in a way that would reinforce proper programming style and habits from the start. We do this as follows:

(1) Case Studies. The central pedagogical tool we use is the case study. These are programming problems for which complete, working programs are designed in a top-down, structured fashion as new programming concepts are introduced. The problem of exploring new concepts in isolation from practical experience is thus avoided. In all there are 51 complete case studies in the book.

(2) Use of a Pseudolanguage. The solutions to the case studies are developed in a structured, top-down fashion in a simple pseudolanguage. This allows us to concentrate on programming rather than the distracting details of Pascal as the programs are developed. Students should learn that program development in a pseudolanguage is a completely separate process (now widely practiced in industry) from the implementation of the resulting program in some particular programming language (in this case Pascal). Each of the programs we design in the pseudolanguage is translated into a complete, working Pascal program in a later section where the new details of the Pascal language can be discussed separately from the problems involved in the program design.

(3) Immediate Introduction of Procedures and Functions. From the first case study on we teach that programs are collections of short, well-defined procedures and functions, which are organized and called from an initial procedure (main program). The crucial concepts of procedures, functions, parameters, and modular program design are thus ingrained into the habits and practice of students from the beginning. Students learn these topics without problem, and their later programming practices are greatly enhanced as a result.

(4) Inclusion of Program Correctness. As part of each case study we include an integrated discussion of program correctness. This starts out quite simply with the early case studies but eventually includes the notions of a program walkthrough, semi-formal verification steps (particularly for loops), program testing, choosing proper test data, robustness, and debugging techniques. Students receive a practical knowledge of the concepts of program verification and are provided a sound basis for advanced courses on the topic.

(5) Integrated Discussion of Program Efficiency. The execution time efficiency (time complexity) and storage space requirements (space complexity) of the programs are discussed for each relevant case study. Time complexity is determined by doing a count of the number of statements executed, and space complexity is determined by counting the number of storage cells used. These simple, intuitive approaches are accurate and practical. Students continuing on in computer science will have a basis for advanced study of these topics, while those terminating after this class will understand practical methods for determining program efficiency.

BOOK ORGANIZATION

All chapters except the first follow a specific format designed to implement our philosophy. Each has four major sections, **Getting Acquainted, In Retrospect, The Challenge,** and **Pascal Implementation.** In the Getting Acquainted section, simple case studies introducing the new programming concepts of the chapter are studied. All of these case studies should be covered because the procedures and functions developed there are often used in later case studies. In Retrospect summarizes these new concepts and provides a place to turn to for review. The Challenge presents more challenging case studies involving the new concepts of the chapter. The Pascal Implementation section mirrors the previous sections exactly in translating the pseudolanguage programs of the case studies into Pascal. All Pascal programs have been written to conform to the ISO standard. Appendix A, **Pascal Reference,** provides a concise reference manual for ISO Pascal. To help acquaint students with this language reference appendix, syntax diagrams from the appendix should be presented in class as new Pascal statements are introduced. Eight groups of exercises are integrated into each chapter, and answers to some of these are found in appendix B, **Answers to Selected Exercises.**

The first chapter of the text is different from the other chapters; it describes a model computer and the simple operations that a computer can perform, providing the motivation for the rest of the book by answering the question, "Why must we write programs?". It was carefully written so that students could read it on their own during the first week of class as the instructor tended to other matters (such as describing how to use the computer). A complete, simple Pascal program is given at the end of the exercises, which the students can type in and run as their first assignment to help acquaint them with their computer terminal and text editor.

We hope that you will find this book as easy and pleasant to use as we have. However, you will probably find parts that you dislike or disagree with. We would be delighted to receive any comments you may have, and corrections will be gratefully accepted and included in future printings or editions.

IN GRATITUDE

Those to whom we are most indebted for the form and content of this book are the thousands of students in the introductory programming course at

Washington State University over the past two years. Their questions, comments, sharp-eyed ability to catch errors, and enthusiasm for the material kept us going. Also to be thanked are the many reviewers who helped us through various stages of the book:

Gabriel Barta
(University of New Hampshire)
Rodney M. Bates
(Kansas State University)
Leland L. Beck
(San Diego State University)
Don Cartlidge
(New Mexico State University)
Cecelia R. Daly
(University of Nebraska)
Nancy Duffrin
(SUNY at Stony Brook)
Arthur C. Fleck
(University of Iowa)
Tamar E. Granor
(University of Pennsylvania)
James L. Hein
(Portland State University)
Rachelle Heller
(University of Maryland)
Leon Levine
(University of California, Los Angeles)
Gene Mahalko
(University of North Dakota)
Lawrence H. Miller
(University of California, Los Angeles)
Ralph Moore
(Modesto Junior College, California)
Keith R. Pierce
(University of Minnesota)
Alan L. Schwartz
(University of Missouri, St. Louis)
Robert F. Simmons
(University of Texas, Austin)
Stephen F. Weiss
(University of North Carolina, Chapel Hill)

Their careful evaluations and widespread support for the book were indeed helpful. The many instructors who labored in front of the classes with rough drafts of the book receive our appreciation, too.

Although many helped us along the way, there are others whose help was particularly important. Roger Hirsch first interested us in writing the book, without his encouragement we might never have started. Shirley Farmer was unflagging in her typing of the manuscript (even when her eyes said "Oh no, not chapter 4 again!") We also thank Mike Langston, whose enthusiasm for the book and personal friendship as a colleague did not temper his constructive criticism of our efforts. Then there are all those at West Publishing Company, who probably never saw so many deadlines come and go before and yet continued to provide the necessary support to keep the project moving. Editor Pete Marshall, production editor Deanna Quinn, and marketing coordinator Reneé Grevious worked so intensely with us that they now seem more like friends than business associates. We also thank Pamela McMurry for carefully copyediting our manuscript; she not only read for syntactic errors but for a true understanding of the material.

Then there are the few without whose support the project definitely would have been doomed: our families. To those who missed us on family occasions, holidays, and weekends, yet continued to stand by as the project seemed to extend indefinitely — well, what can we say? Without you there would be no book.

J. Denbigh Starkey
Rockford J. Ross

To the Student

This book has been designed with you in mind. We have given numerous examples of all important programming concepts and provided exercises to reinforce your learning. If you study this material carefully you will be well prepared for advanced courses in computer science; if this is the only course that you plan to take, you will have a sound understanding of the programming process to apply in later life. For example, it may well be that the most useful things that a future engineer learns from this book are program correctness and efficiency if he or she is later involved in projects where the successful design of a program by members of a team is crucial, or where the speed of a particular software component of a system is important. Similarly, business students may later find that they are responsible for decisions about the purchase or use of programs, and a practical, working knowledge of the concepts of program design, efficiency, and correctness may be far more important than actual programming skills. In short, these topics are of concern not only to computer professionals but to all who will be involved with computers in the future.

To use this book most effectively, you should read the chapters in the order given. The first chapter answers the question, "Why must computers be programmed?" You can read it as you are becoming used to the actual computer you are using; this chapter contains no programming assignments. In subsequent chapters you should study each of the case studies in the Getting Acquainted sections in succession. The order is important because later case studies build on earlier ones. After you understand a case study you should turn to the corresponding Pascal section to see how the program developed in the case study is translated into Pascal. This process of learning the design of a program and then turning to the Pascal section to see how the program is actually written in Pascal is one you will repeat often. The In Retrospect section of each chapter is especially for you. You should read it carefully and then use it as a reference whenever you need to review a particular topic. The Challenge section contains advanced case studies exploring the new topics of the chapter. Later introductory case studies do not depend on previous Challenge case studies, but the Challenge case studies give you a more intimate look at the programming process. You should also learn to use appendix A, Pascal Reference, as quickly as possible. It gives a detailed but concise description of Pascal, and it should be used whenever you want to review features of the Pascal programming language.

Whether you are a computer science student or a student from another discipline we have designed this book to be useful to you now and later as a reference. And one warning: if you have learned a programming language previously on your own, try to forget what you learned. We have seen many sad cases of students coming in with previous programming experience who start well but end up doing poorly because they never shook their previously learned bad habits! Using this book you will not only learn Pascal, you will also learn to be a good programmer.

Contents