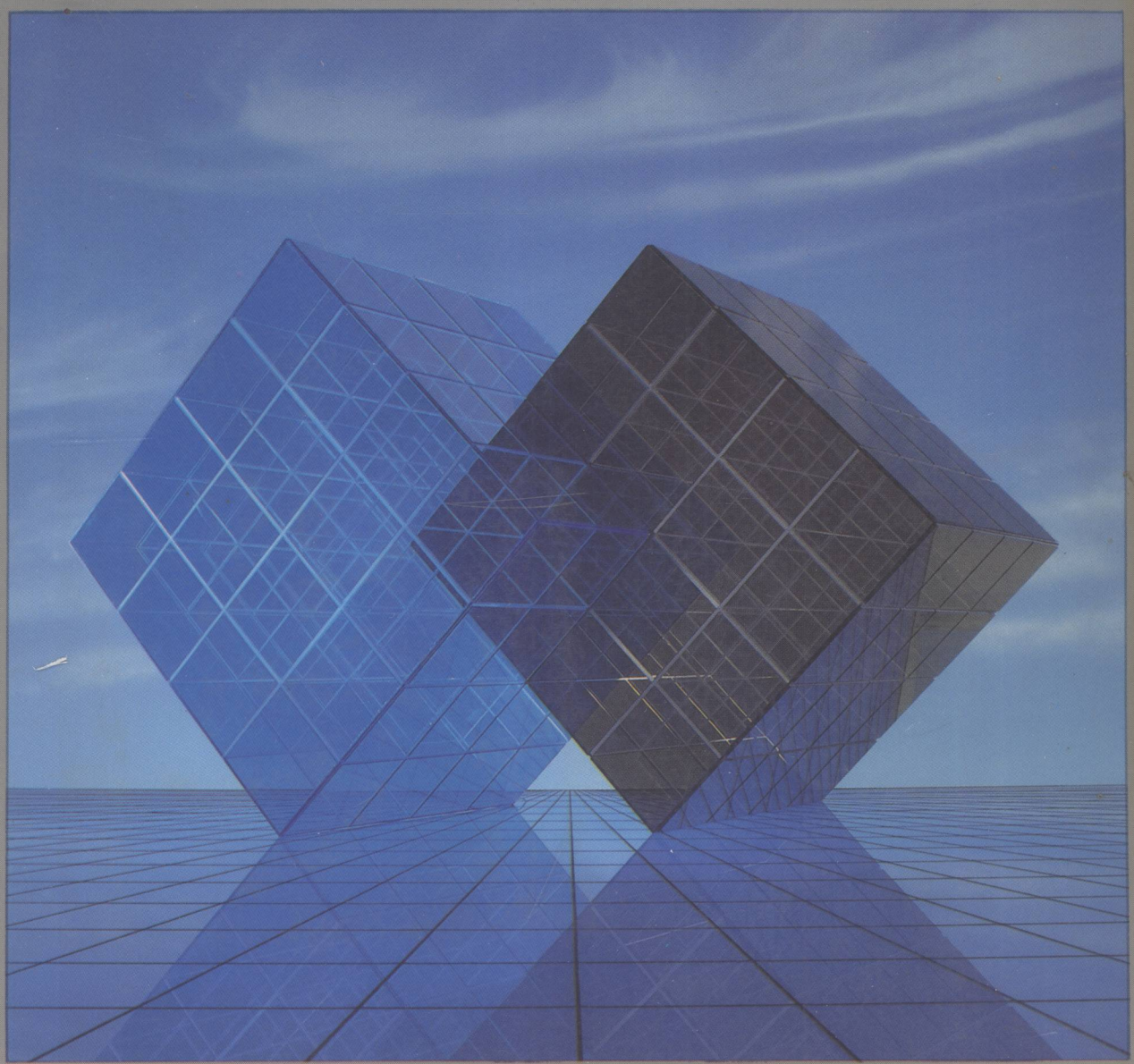
FUNDAMENTAL PROGRAMMING WITH FORTRAN 77

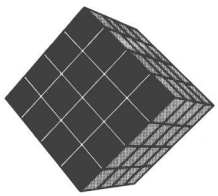
A Science and Engineering Approach



J. DENBIGH STARKEY
ROCKFORD J. ROSS

Fundamental Programming With FORTRAN 77

A Science and Engineering Approach



J. Denbigh Starkey
Rockford J. Ross

Montana State University

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

To our parents
Richard and Mary Starkey
Jim and Dorothy Ross

Production Management: Bookman Productions
Cover Photography: Michel Tcherevkoff for Alleghany
Cover Design: Hal Lockwood, Bookman Productions
Copyeditor: Janet Hunter
Compositor: G&S Typesetters, Inc.

Library of Congress Cataloging in Publication Data

Starkey, J. Denbigh.
Fundamental programming with Fortran 77.

Includes index.
1. FORTRAN (Computer program language) I. Ross, Rockford. II. Title.
QA76.73.F25S77 1987 005.26 86-18899
ISBN 0-314-77805-5

Preface

Fundamental Programming with Fortran 77 is the third book to appear in a series that includes *Fundamental Programming* and *Fundamental Programming with Pascal*. Our purpose for this book is to introduce Fortran 77 within the setting of a true course in programming. Besides giving students rudimentary programming skills and a knowledge of Fortran 77, such a course should leave students with a basic understanding of the science of programming.

The subtitle of our book, *An Engineering and Science Approach,* captures our belief that programming should be taught like other engineering and science courses with a primary emphasis on design and general scientific principles. Just as we would not hire an engineer to build a bridge who knew only the tools and structures used in bridge construction but nothing about bridge design, so we would not trust a programmer who knew only the statements and syntax of Fortran 77 but nothing about program design. Not only is an understanding of design issues crucial to safe and reliable products in both instances, but a knowledge of design methodologies provides a thorough understanding of the discipline in question. Most Fortran books do not provide this experience.

This book embodies an approach that is consistent with our responsibilities and objectives as computer scientists and computer engineers. Here, we first present the science of computing and the design issues fundamental to programming in sufficient depth. Then we describe how to implement well-designed and analyzed programs in Fortran 77. Students may not learn as many of the individual details of Fortran 77 in one term this way, but they will be markedly better programmers, and they will have the proper foundation for further individual study of programming.

We believe strongly in this method of instruction. Consider giving auto-mechanics' students a short course in the use of mechanics' tools, showing them how the tools work and having them practice tightening and loosening a few bolts. Then, a year later, give these same students a defective engine and say, "Fix it." Impossible? Of course. However, we would be guilty of a similar fault if we only presented the tools (Fortran statements) in this book without at the same time demonstrating the logical processes necessary for turning problems into well-designed programs. Thus, we emphasize program design, analysis, and verification as the most important aspects of programming. There are two primary benefits to this approach for the engineering, math, science, and business students who need a course in Fortran.

1. Engineering Design. A key issue in all engineering curricula is design. ABET, the Accreditation Board for Engineering and Technology, requires a substantial design component in all engineering disciplines. The emphasis of this book is approximately three-quarters program design (software engineering) and one-quarter Fortran implementation.

2. Advanced Study. A course taught with this book as its basis gives students a background that makes advanced study easier. While students may need to learn a different programming language (e.g., Pascal) as a prerequisite for advanced computer science courses, they will *not* need to relearn the issues fundamental to programming, making this transition to upper-level courses much easier. They can, in fact, usually learn the new language with little difficulty.

PHILOSOPHY

In determining the content of this book we had a definite pedagogical model in mind. This model stemmed from our realization a number of years ago that when we asked students to design a substantial program independently, many would return with a program consisting primarily of one or two large routines with little modularity; their programs were difficult to read and difficult to modify. Furthermore, when we asked the students simple questions about the efficiency or correctness of their programs, they 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, subroutines and functions were normally introduced late in these textbooks, almost as afterthoughts, as the "right way" to program. Students were mistakenly led to believe that subroutines 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 tools we use are case studies. 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. The case studies do not require mathematics beyond precalculus. Also, the case studies have been chosen to illustrate general program-design techniques.

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 details of Fortran 77, 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 Fortran 77). Each of the programs we design in the pseudolanguage is translated into a complete, working Fortran 77 program in a later section, where the new details of the Fortran 77 language are discussed separately from the problems involved in the program design.

3. Immediate Introduction of Subroutines and Functions. From the first case study on we teach that programs are collections of short, well-defined subroutines and functions, which are organized and called from an initial procedure (main program). The crucial concepts of subroutines, functions, parameters, and modular program design are thus ingrained into the habits and practice of students from the beginning. Students learn these topics without any 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, semiformal verification steps (particularly for loops), program testing, selection of proper test data, robustness, and debugging techniques. Students receive a practical knowledge of the concepts of program verification.

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 case study as appropriate. 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; those not pursuing the subject after this course will understand practical methods for determining program efficiency.

ORGANIZATION

All chapters except the first follow a specific format designed to implement our philosophy. Each begins with three major sections: Getting Acquainted, In Retrospect, and The Challenge. 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 subroutines and functions developed there are often used in later case studies. In Retrospect summarizes these new concepts and provides a place to which to turn for review. The Challenge presents more challenging case studies involving the new concepts of the chapter. The Fortran implementation portions of each chapter mirror the previous sections exactly. In the Getting Acquainted with Fortran, Fortran in Retrospect, and The Challenge in Fortran sections, we translate into Fortran 77 and review the pseudolanguage programs of the case studies. All Fortran programs have been written to conform to the Fortran 77 standard.

Appendix A, Other Fortran Features, provides a concise reference manual for advanced Fortran 77 topics not covered in the text. 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 Fortran programs were tested on a VAX 11/750 computer using the Fortran 77 compiler developed by S. I. Feldman and P. J. Weinberger.

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 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 Fortran program is given in the exercises at the end of the chapter. The students can type this program in and run it 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 found its working version. We would be delighted to receive any comments you have, and corrections will be gratefully accepted and included in future printings or editions.

IN GRATITUDE

Special thanks go to Ruth Barton, Michigan State University, and Donna McClelland, Montana State University, who contributed their expertise to this book's development. Thanks are also due to the teaching assistants who used various forms of the Fortran portion of this book in teaching introductory Fortran courses at Montana State University: Mike Turner, Brian Thome, Bob Wall, and Jim Hill.

We would also like to thank those people who earlier reviewed the Fundamental Programming material that appears in this 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); and Stephen F. Weiss (University of North Carolina, Chapel Hill).

Finally, Cheryl Ross, in addition to her responsibilities as wife and mother, cheerfully carried out the job of typing the manuscript.

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 have a sound understanding of the programming process. For example, it may well be that the most useful thing that a future engineer or scientist gains from this book is not a knowledge of Fortran 77 but an understanding of program correctness and efficiency, since the successful design of a program by members of a team or the speed of a particular software component of a system may be crucial in later projects. 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.

For highlighting concepts in the book we sometimes print words in *italics* or **boldface** type. *Italics* are reserved for phrases we want to emphasize and for terms that are being defined. **Bold** print is used in our program designs to mark keywords that are important. These uses of italic and boldface type will become clear as you read the book.

Whether you are a computer science student or a student from another discipline, this book will be useful to you now and later as a reference. One warning: if you have learned to program on your own, be careful! We have seen many sad cases of students with previous programming experience who started well but ended up doing poorly because they never shook off their earlier bad habits. If you use this book you will learn to be a competent programmer. We hope you enjoy learning to program.

J. Denbigh Starkey
Rockford J. Ross

Contents