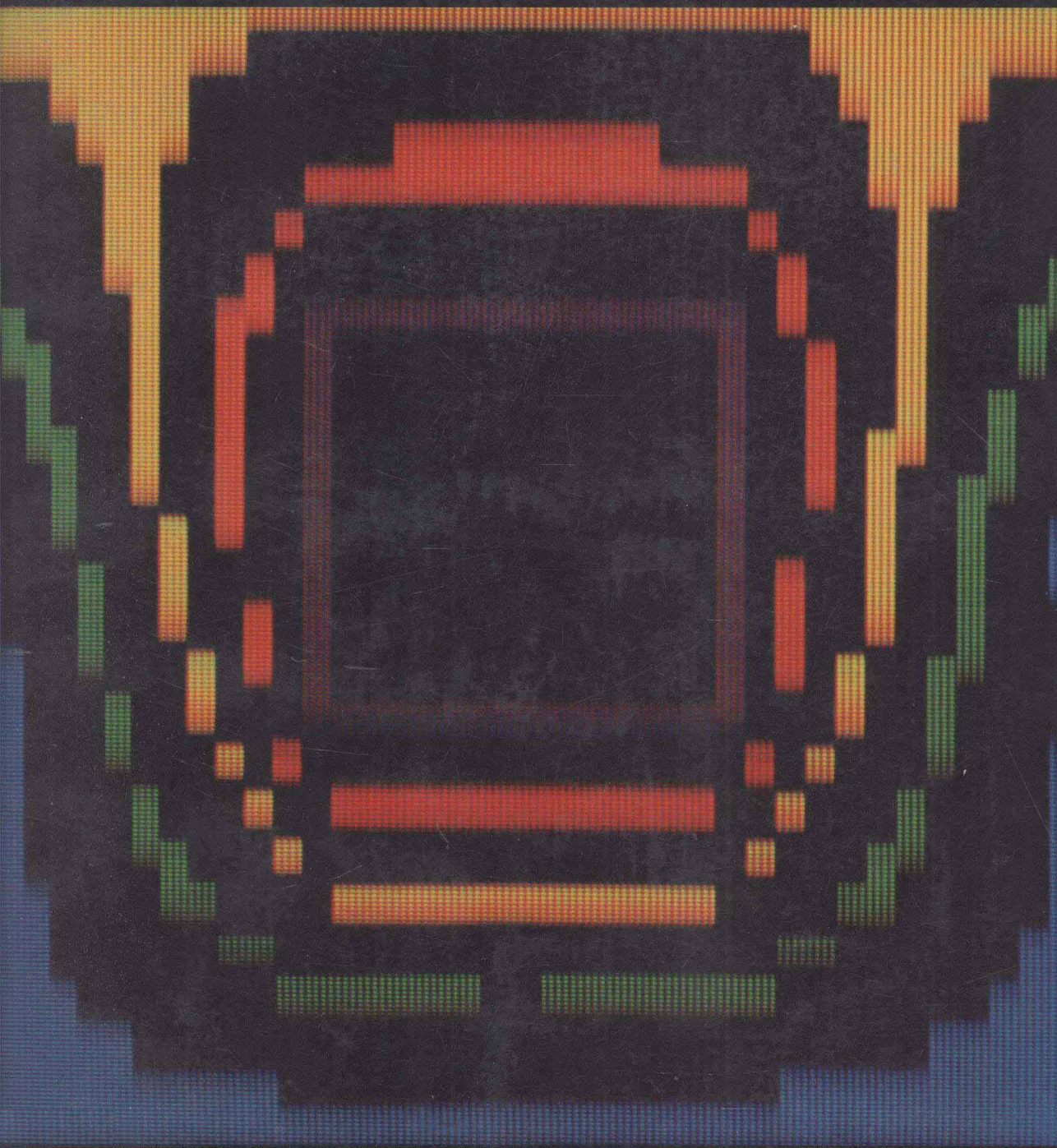
COMPUTING FOR ENGINEERS AND SCIENTISTS WITH FORTRAN 77



Daniel D. McCracken

COMPUTING FOR ENGINEERS AND SCIENTISTS WITH FORTRAN 77

Daniel D. McCracken

City College of New York

John Wiley & Sons

New York Chichester Brisbane Toronto Singapore

Library of Congress Cataloging in Publication Data:

McCracken, Daniel D.
Computing for engineers and scientists with Fortran 77.

Includes index.
1. Engineering—Data processing. 2. Science—Data processing. 3. FORTRAN (Computer program language) I. Title.
TA345.M395 1984 620'.0028'4 83-23473
ISBN 0-471-09701-2

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

COMPUTING FOR ENGINEERS AND SCIENTISTS WITH FORTRAN 77

To Charles B. Stoll and Walker G. Stone
for their support and encouragement as my first editors

PREFACE

This book uses the study of programming in Fortran 77 to provide:

- A programming competence that an engineer or scientist can use in work or study.
- An understanding of the powers and limitations of computers.
- A basis for effective communication with programming experts when more difficult tasks are encountered.
- An appreciation for the increasingly common ways to use a computer as a tool *without* any conventional programming.

Since Fortran is the *lingua franca* of applications in engineering and science, it is the choice for this book. Fortran 77 is used, specifically, because it is the latest standard and because its newer features permit the writing of structured programs that are much easier to understand and maintain than the programs written in earlier versions of Fortran.

As with my previous books, there are many illustrative programs. The examples are drawn from engineering, science, and college mathematics, at a level consistent with the fact that a course of this type is often taught in the freshman year. A number of standard techniques in elementary numerical methods are introduced through programming examples, recognizing that many students will never take a formal course in numerical methods. Throughout the text there are warnings about traps for the unwary, such as the accumulation of roundoff errors, the effects of finite precision, and the foolishness of blind reliance on double precision.

Readers with the goals assumed here have no need to know all of the Fortran 77 language. Some of the older and less desirable features of the language (the arithmetic IF, the assigned GO TO, etc.) are not covered at all. Others are covered in part. List-directed input and output serve nicely while the reader is learning more fundamental matters, for example, and then formatted input and output are considered at an appropriate depth in Chapter 6.

Heavy emphasis is placed on good programming style. Logic structures are held to a bare minimum and are used in a consistent way. Statement numbers are never attached to statements other than CONTINUE and FORMAT. The GO TO statement is used only in simple Fortran 77 implementations of the WHILE and REPEAT structures. Pseudocode is introduced in the third chapter, and subprograms are introduced in the fourth—the latter much earlier than in most other Fortran texts. Subprograms are used consistently thereafter as a program organizing tool, and effective use of subroutine libraries is emphasized with several examples from the IMSL library. In Chapter 5, after the reader has enough background to understand the issues, there is a full treatment of program development, including modularization, top-down and stepwise development, and program testing. These issues are reinforced in later chapters with larger example programs.

There are many exercises, ranging from "fingerwork" to reinforce the syntax, to challenges that might serve for term problems. Answers to nearly half of the exercises are given at the end of the book.

The final chapter is a "horizon-broadener," with a wide-ranging sampling of ways to use computers without any conventional programming at all: mathematical and engineering packages, symbolic mathematics, bibliographic searching, NOMAD as an example of a fourth-generation language, and text editing and formatting.

In sum: The book is—among other things—a text on Fortran programming, and is intended for use in a conventional course on that subject. I, for one, have no doubts about the legitimacy of such a course for future engineers and scientists. But the reader will be left with a clear understanding that when the computer is viewed as a tool of the practicing engineer or scientist, there are increasingly attractive alternatives to actual programming for getting applications done. That statement is true even today, and will be an unremarkable commonplace in the working careers of students now in school. The book thus provides a solid core of skills for use in today's world of computer applications, and a bridge to tomorrow's.

An extensive Instructor's Manual gives answers to all exercises that do not have answers in the book. It also contains suggestions for newer teachers on how to teach programming, an annotated model syllabus, sample exams, and a summary of the very few programming changes that are needed when the book is used in the WATFIV environment.

Daniel D. McCracken

New York
October, 1983

ACKNOWLEDGMENTS

Like most programming, producing a book is a team effort. It is a pleasure to acknowledge the many and varied contributions of the following people.

Gregory P. Williams, Masstor Systems Corporation, an old friend from General Electric days, served as a faithful reviewer and provided early guidance that influenced the shape of the book in fundamental ways.

Charles L. Baker, Science Applications, Inc., supplied much good advice on programming style matters and on the way programming is really done.

Jeffrey R. Sampson, University of Alberta, and Paul P. Clement, Advanced Systems, Inc., were the most thorough and helpful reviewers I have ever worked with. (And that's saying something, because I value the contribution of reviewers very highly indeed, and have worked with many good ones.) Their contributions ranged from catching programming slips, to improving style, to suggesting major revisions in early drafts.

John L. Lowther, Michigan Technological University, also did a fine job of reviewing, and suggested the Fortran 77 implementation of the WHILE construct that I have used. He disclaims credit for inventing it, but I hadn't seen it and it is much better than what I had been planning to use.

Michael J. Clancy, University of California, Berkeley, and Leon Levine, University of California, Los Angeles, were particularly helpful in the early planning stages, drawing on their extensive experience with the kind of course for which this book is intended.

My editors at Wiley, first Gene A. Davenport and now Carol Beasley, capably assisted by Judith Watkins, were uniformly helpful and supportive. High on the long list of things they did to assist me was the provision of an astonishing number of highly talented reviewers, without whose services I simply would not know how to approach this kind of project. Most of these reviewers worked anonymously; I am delighted that all of them said "yes" when my request to give them their due credit here was passed along.

The Wiley reviewers: J. Mack Adams, New Mexico State University; Donald E. Burlingame, State University of New York, Potsdam; Donner A. Dowd, Jr., Lake Superior State College; Henry A. Etlinger, Rochester Institute of Technology; Elaine N. Frankowski, Honeywell Information Systems; Robert M. Graham, University of Massachusetts; Charles E. Hughes, University of Central Florida; Thomas E. Kurtz, Dartmouth College; Richard J. LeBlanc, Georgia Institute of Technology; Hans E. Lee, Michigan State University; George L. Miller, North Seattle Community College; Frederick J. Mowle, Purdue University; Steven S. Muchnick, University of California, Berkeley; James T. Perry, San Diego State University; Franklin Prosser, Indiana University; Woodrow E. Robbins, North Carolina State University; Bernard H. Rosman, Framingham State College; J. Denbigh Starkey, Washington State University; David B. Teague, Western Carolina University; John F. Wakerly, Stanford University;

Jerry M. Waxman, Queens College; Lloyd Weaver, Purdue University; R. A. Williams, University of Akron.

Chapter 9 contains several examples drawn from the subroutine library of IMSL, Inc. Thomas J. Aird and Granville Sewell were generous in their assistance. Dr. Sewell also created the input and produced the graph using IMSL's TWODEPEP in Chapter 10.

Chapter 10 includes a section on muMATH, a symbolic mathematics package from the Soft Warehouse, Honolulu. David R. Stoutemyer made several helpful suggestions in getting this material right.

Chapter 10 also contains a section on bibliographic searching based on the facilities of DIALOG, Inc. Charles T. Meadow and Charles D. Sullivan were most helpful in adapting this material so that it both showed what is possible and remained intelligible to the reader unfamiliar with such services.

All of the text and almost all the programs were developed using the facilities of Dun & Bradstreet Computing Services, formerly National CSS, Inc. (This name change came too late, unfortunately, to be able to change the text references to National CSS.) My closest contacts there in recent years have been Nicholas A. Rawlings and Christopher Grejtak, with much help on VS Fortran from Lloyd E. Fuller and Walter H. Horowitz. Mr. Rawlings also reviewed much of the manuscript and assisted in a variety of other ways, especially with the NOMAD material in Chapter 10. The letter reproduced on page 320 attempts to express my appreciation to some of the many others now or formerly at the company to whom I am indebted.

A few of the programs were run on the facilities of the City University of New York, and I wish to express my appreciation for that support. My own institution, the City College of New York, is a component of the City University, and it is a pleasure to express my appreciation for the support given to me by my colleagues and by my chairman, George G. Ross. My appreciation also goes to the director of the CCNY computer center, George W. Elder, and to Paul Fortoul, a systems programmer who has been a major factor in my education in the past two years.

Too little credit is given to the people who turn a manuscript into a book: the production staff, with their wide range of talents and responsibilities. Their usual thanks is that if they do their job right, nobody notices! The production people I worked with at Wiley were uniformly competent, cooperative, and effective. I am happy to name them: Deborah Herbert, Madelyn Lesure, Eugene Patti, Elaine Rauschal, and Ruth Sandweiss. I appreciate all of their efforts.

Finally, I wish to say "thank you" to my students at City College. I suppose it's been said before, but I think I have learned more from them than they have learned from me. They make teaching a pleasure.

D.D.M.

COMPUTING FOR ENGINEERS AND SCIENTISTS WITH FORTRAN 77

CONTENTS

CHAPTER ONE
GETTING STARTED IN COMPUTING AND FORTRAN 1

CHAPTER TWO
THE ASSIGNMENT STATEMENT AND RELATED MATTERS 17

CHAPTER THREE
CONTROL STRUCTURES 43

CHAPTER FOUR
SUBPROGRAMS, I 87

CHAPTER FIVE
PROGRAM DEVELOPMENT AND TESTING 107

CHAPTER SIX
FORMATTED INPUT AND OUTPUT 131

CHAPTER SEVEN
THE DO STATEMENT AND ARRAYS 153

CHAPTER EIGHT
DOUBLE PRECISION, COMPLEX, LOGICAL, AND CHARACTER VARIABLES 185

CHAPTER NINE
SUBPROGRAMS, II 221

CHAPTER TEN
NONPROCEDURAL APPROACHES TO APPLICATION DEVELOPMENT 263

APPENDIX
FORTRAN INTRINSIC FUNCTIONS 321

ANSWERS TO STARRED EXERCISES 327

INDEX 357

ONE

GETTING STARTED IN COMPUTING AND FORTRAN

The uses of computers

Electronic computers are widely used in the solution of the problems of science, engineering, business, and education. This use is based upon their ability to operate at great speed, to produce accurate results, to store large quantities of data, and to carry out long sequences of operations without human intervention.

Here are some examples of the kinds of applications that we assume to be of interest to readers of this book.

- Designing a chemical plant requires calculations of capacities, operating conditions, and yields, under a variety of circumstances. Determining the optimum operating conditions, taking into account technical and economic factors, requires large amounts of computer time.
- Weather prediction studies involve large amounts of data and the solution of equations that, although not inherently difficult, call for vast amounts of computation.
- Statistical studies of the relationships among various factors that affect a person's learning ability often require computers. The computations may be as modest as a student research project, or as complex as a study following millions of people for many years.
- The communications industry uses computers to store, process, and disseminate information. Telephone systems make intensive use of computers for billing, network management, and—in increasing numbers—within the telephones themselves. In a different kind of communication, all the text for this book was processed by computer, with drafts produced by one computer and the photocomposition of the final book by another. More and more computing involves information of a textual sort or digitized voice data, rather than what we ordinarily think of as "numbers."
- The investigation of the possible structure of a complex organic compound could involve a combination of computations of binding energies, interatomic distances, and so on, with an elaborate computer program to present the results in a graphical form, often in color.

■ The design of an airplane uses thousands of hours of computer time to investigate the interrelated requirements of structures, aerodynamics, power plants, and control systems as they would operate under various flight conditions. After a prototype has been built, flight testing generates voluminous data that must be captured and analyzed, the latter often using statistical techniques. Then, during manufacturing, many applications that would usually be called "data processing" come into play: project control, materials requirements planning, inventory management, purchasing, and quality control, among many others. Finally, operating such a fleet of equipment requires computers to order spare parts and keep track of them, schedule crew assignments and aircraft maintenance, and plan flights.

The computer techniques needed to work with applications as diverse as these depend to a certain extent on the subject matter of the task. But the person using a computer in any of them would need to know something about how to specify the desired processing to a computer, which is essentially what this book is about. And while we are learning to "talk to" a computer, we shall also look at small but representative examples of a variety of applications that are typical of the way computers are used in engineering and science.

The steps in solving a problem with a computer

There is much more to solving a problem with a computer than the actual use of the computer. It is instructive to outline the complete process of setting up a technical problem for computer solution to see just what people do and what the computer does.

Problem identification and goal definition

A computer cannot decide for us what we want to do. *We* have to decide what the system under development is supposed to accomplish, what goal or combination of goals it must satisfy, under what conditions it must operate, and what general approach to solving the problem is to be taken. In some applications this step may be trivial; in others it may take months or years. In any case, the step obviously demands full knowledge of the problem area; there is usually little the computer can do to help us with it.

Mathematical description

In many although not all of the kinds of applications that will be considered in this book, it is necessary to formulate a mathematical description of the process under study. This can generally be done in a variety of ways; an approach must be chosen or a new one developed if no standard method applies. This step, in which the computer is not involved, requires full knowledge of the problem and of the relevant branches of mathematics.

Numerical analysis

The mathematical formulation of the problem may not be directly translatable to the language of the computer, since the computer can only do arithmetic on rational numbers and make simple decisions. Differential equations, integrals, and trigonometric functions, to name a few common examples, must be expressed in terms of arithmetic operations. Furthermore, it must be established

that any errors inherent in the data, or introduced by such operations as expressing continuous functions in terms of finite approximations, do not invalidate the results.

This entire branch of modern mathematics is largely outside the scope of this book. We shall assume that the reader approaches the problem-solving process with a method of solution—in this sense—in hand. We shall, however, illustrate quite a number of elementary numerical methods in the course of demonstrating various programming concepts.

Algorithm formulation

The next step is to devise a precise and unambiguous statement of exactly what we want the computer to do, expressed as a finite sequence of the operations of which it is capable. A computer cannot follow the order "solve this equation"—at least not in any *direct* sense, in Fortran. (But see the short discussion in Chapter 10 of muMATH, where we shall see a few examples of operations on equations in symbolic form.) It *can* follow the order "square the number identified by the variable named X2REAL," or "go back to the beginning if the current value of the variable named EPS is greater than 0.0001," or "divide SUM by N." Furthermore, and this is crucial, the exact sequence of actions must be specified in complete detail in advance, especially at all points where the computer is required to make a "decision" based on relationships among values in the computation. An unambiguous definition of the actions to be carried out in solving a problem is called an *algorithm.* An algorithm might be expressed in English sentences, or in a computer language such as Fortran, or in a notation called *pseudocode.*

Computer programming

The next task, assuming the algorithm was not originally expressed in a programming language, is to express it so. Fortran is one language among many others used for this purpose. It is the language most commonly used for the types of technical problems that we address in this book. Fortran was developed in the mid-1950s by IBM and some of its customers, with a team led by John W. Backus.

A computer language imposes restrictions of its own in terms of what kinds of commands it can "understand" and carry out; different languages have different capabilities. Furthermore, the exact form in which our orders to the computer are expressed is prescribed for each language, and the rules are generally rather inflexible.

One major purpose of this book is to enable you to construct correct Fortran programs to solve problems of interest to you. Even if, as will probably be the case, you do most of your computer work using programs written by others, knowing Fortran will give you a basis for understanding what computers can (and cannot) do. It will also give you a vocabulary for talking about your needs with computing experts.

Program testing

There are so many opportunities to make mistakes in programming that most programs do not operate as intended when first tried, because of errors in any of the steps listed above. Mistakes must be located and corrected, and the pro-

gram must be thoroughly tested to establish as fully as possible that it actually does what the programmer meant it to do. The computer is used heavily in this step, which can easily take longer than writing the program in the first place.

We shall place considerable emphasis in this book on writing programs in a way that minimizes programming errors and facilitates locating those that do occur. The techniques suggested will also greatly facilitate the maintenance of programs. (All programs that are used over an extended period of time have to be modified as requirements and computer equipment change.) Programs written with maintenance in mind are very much easier to maintain than those where this consideration has been ignored.

Production

Now, finally, the program can be used to process "real" data, that is, data other than the sample values used in program testing. This may be done in a variety of ways, depending upon the computer and the application. Sometimes data is entered from a computer "terminal" (a typewriterlike device that may also have been used to prepare the program) with results presented on a video display or other device that is part of the terminal. Sometimes many sets of data values are punched into cards and all of the cases run consecutively, with results printed on a paper listing. In other cases the computer is *on-line* as part of a process control system. Here, the data is obtained by direct input from sensors located in the process equipment; the output is a combination of printed logs and of signals controlling the operation of the process equipment.

Interpretation

Except in the process control situation just noted, results produced by the computer in response to our program do not always constitute an "answer" to the problem. The computer user must often now interpret the results to see what they mean in terms of the combination of goals of the proposed system. It is often necessary to repeat some or all of the preceding steps before the problem is really "solved."

Several conclusions may be drawn from this discussion. First, the computer does not, by itself, solve problems. It only follows exactly the computational procedures (i.e., the program) given to it. Second, a computer does not relieve the user of the responsibility of planning the work carefully; in fact, use of the computer demands more careful planning than noncomputerized methods ordinarily do. This is an important secondary benefit of using the computer. Third, a computer does not in any way eliminate or even reduce the need for a full and detailed understanding of the problem area, or for a thorough knowledge of the related mathematics.

The emphasis in the first part of this book is on the programming step. Problem identification, goal definition, and mathematical formulation are in the province of the technical area under consideration: electrical engineering, physics, statistics, operations research, or whatever. Numerical analysis is a branch of modern mathematics in its own right. Program testing is discussed, but not at great length and mostly in the context of how to write programs that minimize the probability of programming errors. Production is not ordinarily the programmer's responsibility, except possibly in the case of student exercises. The interpretation of results brings us back into the specialized field of the problem area.

But don't write a program at all if one already exists!

Anyone who needs to know Fortran should also know that there are often alternatives to writing Fortran programs. For a great many of the things that a user might wish to do with a computer, other people have already written programs which, possibly with slight modification, will serve the need. Programs of this type fall into two categories: software tools and applications packages.

Examples of software tools:

- A *text editor* permits the entry, modification, and other processing of textual data, such as letters, reports, and computer programs. A related *text formatter* can be used to print documents in an attractive form or used to drive photocomposition equipment.
- The output of a program is often most useful if displayed graphically. A *graphics package* produces pictures on a video display terminal or plotter. It is seldom in the user's best interest to write his or her own graphics programs, since they can be exceedingly complex and since such programs already exist.
- There are many hundreds of *databases* on a vast range of subjects such as, new chemical compounds, abstracts of publications in psychology, stock and bond prices, or the AP and UPI news wires. Access to these databases is usually through a *time-sharing network,* with a telephone connection between the user's terminal and the database.

Examples of applications packages:

- An electrical engineer wishing to study the performance of a proposed electrical network containing various kinds of components, including nonlinear devices such as diodes and transistors, can choose from among perhaps a dozen different packages that deal with such problems. Some of the packages analyze various special aspects of the network, such as the effect of electrical noise, or the identification of the critical components in terms of a worst-case analysis of component characteristic variations.
- The designer of an optical lens can turn to any of several packages that may "know" more about the intricacies of lens design than the average designer does.
- A chemical engineer wishing to study the probable performance of a proposed piping network can choose the package that best fits his or her precise needs: Transient or steady-state? Compressible or incompressible? With or without loops in the piping?
- The designer of a new integrated circuit chip has available a number of packages that help lay out the components so as to minimize fabrication problems, power consumption, time delays due to capacitance, or whatever else may be of concern.

There are literally thousands of applications packages of the general sort suggested by these examples. When the problem to be solved fits an existing package, it almost always makes sense to buy or lease it rather than writing a program yourself. The package, in most cases, will have been written by top experts in the field. It will have been thoroughly checked out—in part by other users!—and therefore be much less likely to contain significant errors than whatever you might write yourself. And there will be an organization standing behind

it for maintenance if problems do develop, or if modifications are needed to make the package more effective or to make it work on new computing equipment.

So why learn Fortran?

Then what is the purpose of learning Fortran at all, if you will more commonly use software tools and applications packages instead of writing Fortran programs? Four reasons:

- You need to know what a computer is capable of, and, sometimes just as important, what it is *not* capable of. Learning a programming language and solving some problems yourself is the best way to gain an understanding of the power and limitations of a computer.
- It will often happen that the program someone else has written will not *quite* do what you want. What then? In some cases, it will be your task to modify the package to fit your needs.
- If the modifications are beyond your nonprofessional abilities, or if no existing program is suitable and a new one must be written, you will need to call upon the services of experts. To interact effectively with them you will need to know not only what is reasonable to ask (in terms of what a computer can do) but also enough about *their* field to be able to talk intelligently. Learning Fortran and writing some programs yourself provides that background.
- Many students, especially those in electrical engineering, will need to take further courses in computing to be able to design systems that incorporate microcomputers. Studying Fortran is one good way to get started learning enough about computers and computing to be able to do such design work.

This book deals with both aspects of getting work done with a computer, i.e., writing Fortran programs yourself and using programs that others have written. In the first eight chapters of the book you are given enough information about Fortran to enable you to write meaningful programs. Using this knowledge, you will be able to write programs in your other courses (if you are in school), and this experience will serve your need to know what a computer can do and how to talk with computer experts. Chapter 9 emphasizes the use of preprogrammed libraries whenever possible. The last chapter is a sampling of various ways to get work done with a computer without doing any "programming" at all, at least not as "programming" has conventionally been understood.

A simple program

Let us begin the study of Fortran programming by considering a simple example of a program, one where the required processing is so short and easily stated that the algorithm is a matter of one sentence.

The task is to compute the value of a certain fourth-degree polynomial for a value of `X` that is to be obtained from the computer terminal or from a punched card. Let us turn to the program shown in Figure 1.1 to see how this job might be done.

NOTE! The presentation of this first program is designed only to give you