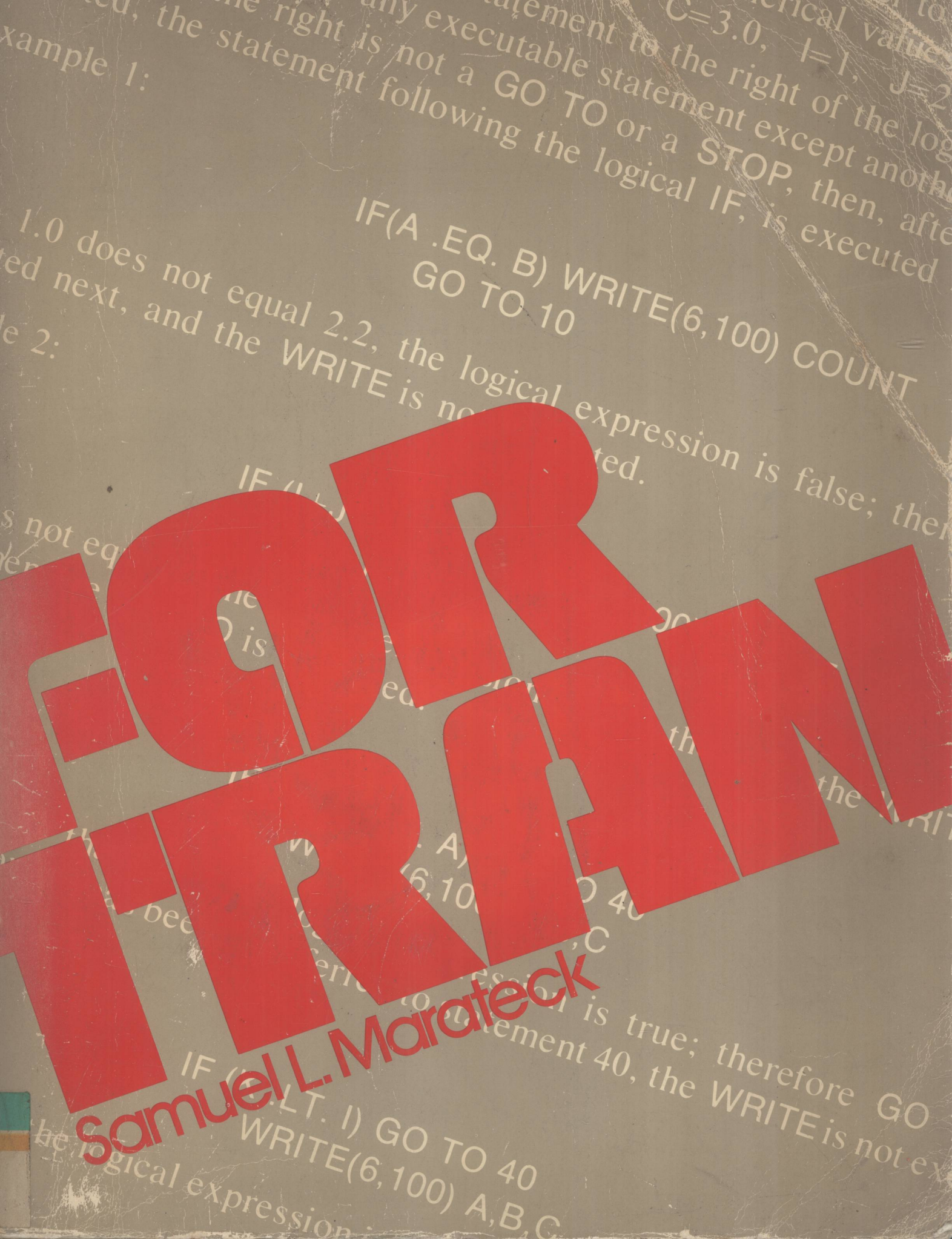
FORTRAN
Samuel L. Marateck

FORTRAN

Samuel L. Marateck
New York University

ACADEMIC PRESS NEW YORK SAN FRANCISCO LONDON
A Subsidiary of Harcourt Brace Jovanovich, Publishers

ACADEMIC PRESS, INC.
111 Fifth Avenue, New York, New York 10003

United Kingdom Edition published by
ACADEMIC PRESS, INC. (LONDON) LTD.
24/28 Oval Road, London NW1

Library of Congress Cataloging in Publication Data

Marateck, Samuel L
FORTRAN.

Includes index.
1. Fortran (Computer program language). I. Title.
QA76.73.F25M37 001.6′424 76-45991
ISBN 0–12–470460–3

PRINTED IN THE UNITED STATES OF AMERICA

To my parents Harold and Rita

FORTRAN

Preface

This book is an outgrowth of notes the author uses in a course in FORTRAN that he teaches to undergraduates at New York University. Its purpose is to teach the student how to program using the FORTRAN language, and it is written for students who have no prior knowledge of computers or programming.

The design of the book has special significance and deserves special comment. The right-hand pages contain pictorial material on programming which most students will readily understand; this is described in detail in "To The Reader." The left-hand pages contain the explanatory text. You will see that strict adherence to this design has resulted in a number of partially filled pages. It has been the author's experience with a book on BASIC, which he wrote and designed in the same way, that in many cases students who visit the computer center and have studied only the right-hand pages, have been able to write and run programs on their first visit. The student should, in order to understand all facets of the programming techniques described, also read the text on the left-hand pages.

There are other features that should help the beginning student. The author usually introduces only one new programming concept per program. Thus many of the programs in the book are first written as a series of smaller programs, each of which serves as a step in understanding the entire larger program.

Easy to understand programs have been used to illustrate the various programming techniques discussed in the book. These programs are the solutions to problems drawn from various disciplines, and all students, whatever their major field, should understand them without difficulty. After students have been presented these programs, they will have the ability to read optional sections in which these same techniques are applied to more advanced programs.

The author also includes examples of common programming mistakes made by beginning students when they are not explicit enough in translating their thoughts into programming instructions. The author has found this type of example to be an effective teaching technique.

Since one of the most difficult parts of learning FORTRAN is the mastering of the input/output statements, these statements are introduced early in the book. Their ramifications are explained as the programs demand it. Thus the reader is exposed to input/output concepts gradually. Then in Chapter 8 the author explains the input/output capability of FORTRAN in great detail.

Sound programming techniques, including programming style and its logical extension, structured programming, aid the programmer in all stages of program writing: the

design, writing, debugging, and maintenance of programs. Sound programming techniques are emphasized from the beginning, and they are introduced as they are needed and as the statements being discussed allow their use. Chapter 10 is devoted to the application of structured programming in the design and writing of programs. Also discussed in that chapter are top-down design, top-down testing of programs, and the HIPO diagram.

At some point the FORTRAN programmer should understand round-off errors and significance as they relate to the bit configuration of the computer being used. The greater part of Chapter 12 is devoted to a discussion of these concepts, although the effect of round-off errors is introduced early in the book.

Almost the entire book is devoted to a discussion of American National Standard (ANS) FORTRAN. Since some students will be using Standard Basic FORTRAN, which has fewer instructions and which consequently can be used on machines which have less memory, any statement that is described and is not available in Standard Basic FORTRAN is footnoted as such.

The features of the WATFOR and WATFIV compilers are described as well as features of WATFIV-S (the IF-THEN-ELSE and the WHILE-DO) along with their ANS analogs. Many features of WATFOR/WATFIV/WATFIV-S have been incorporated into the proposed ANS X3.9 FORTRAN language revision (1977), and thus even the reader who will not be using WATFOR/WATFIV/WATFIV-S might want to take more than a casual interest in sections describing these compilers because of their applications to the proposed ANS compilers.

Those teachers who do not intend to use all the chapters in the book may be interested in knowing that the last four chapters—Chapter 10 (structured programming), Chapter 11 (The COMMON and the EQUIVALENCE statements), Chapter 12 (Significance, DOUBLE PRECISION, and COMPLEX numbers), and Chapter 13 (More Input/Output)—can be read in any order.

We have included in the appendices material that all readers might not have use for, e.g., control cards for the IBM 360/370 and time sharing.

It is a pleasure to thank Professor J. T. Schwartz and Professor Max Goldstein for their friendship, their many kindnesses and for their constant support while I was writing this book; and H. David Abrams and Professor Carl F. R. Weiman for acting as a sounding board for many of my ideas. It is also a pleasure to thank Jeffrey Akner and his computer facility staff for their cooperation and Professor Robert Richardson and Professor George Basbas for granting me free time on their computer.

To the Reader

This book has been written on the premise that it is at times easier to learn a subject from pictorial representations supported by text than from text supported by pictorial representations. With this in mind, beginning with Chapter 2 we have used a double page format for our presentation. On the left-hand page (we call it the text page) appears the text, and on the right-hand page (we call it the picture page) appears the pictorial representation, consisting mostly of programs and tables.

Each picture page was written to be as self-contained as possible, so that the reader, if he so desires, may read that page first and absorb the essence of the contents of the entire double page before going on to read the text. The text page consists of a very thorough discussion of the programming techniques presented on the picture page. It refers to parts of the programs and tables on the picture page; when reference is made on the text page to a given line of print on the picture page, that line—whenever it is feasible to do so—is reproduced in the text to promote readability. Students who have a previous background in programming languages and others who understand the picture page completely may find that in some chapters they can skip the text (left-hand) pages and concentrate on the picture pages.

The following techniques are used as aids in making the picture page self-contained:

1. As many as possible of the ideas discussed in the text are illustrated in the programs and tables. The captions beneath these capsulate much of what is said in the text.

2. Words underlined in the captions describe lines underlined in the figures. To illustrate this, part of Fig. 2.2a is reproduced below.

```
VAR1 = 11.4
VAR2 = 20.2
STOP
END
```

Figure 2.2a. In the program the number 11.4 is assigned to the variable VAR1 and 20.2 to VAR2 in the assignment statement.

The statements VAR1 = 11.4 and VAR2 = 20.2 are underlined to show that they are

described by the words underlined in the caption. Thus they are both assignment statements.

3. To the right of most programs appears a table that describes what effect certain statements in the program have on the computer's memory. For instance, the following table describes the effect that VAR1 = 11.4 has on the memory:

DESCRIPTION	VAR1
VAR1=11.4	11.4

We see from the table that this statement caused the number 11.4 to be associated with VAR1 in the computer's memory. The line-by-line analysis afforded by these tables should aid the reader in understanding the program.

Contents

3

Calculations and the READ Statement

4

Functions and the IF Statement

5

The DO Loop, the IF-THEN-ELSE and the WHILE Loop

6

Subscripted Variables, the DATA Statement, and the Implied DO Loop

7

Doubly Subscripted Variables and Matrix Multiplication

8

Input/Output

9

Functions, Subprograms, and Subroutines

10

Structured Programming

11

The COMMON Statement and the EQUIVALENT Statement

12

Significance, Double Precision, Complex Numbers

13

More Input/Output

Appendix

1

Introduction to Computers and Programming

1.1. General Remarks

If you wished to categorize the times we live in, in terms of the technological advance that most affects our lives, you would call our age the age of the computer; the computer is all encompassing. For instance, in business some of the uses of computers are for billing, check writing, and inventory control; every large organization uses computers to process its records. In another realm—science and engineering—there are two spectacular examples of the use of computers: (1) The multitude of calculations that enabled the space program to put a man on the moon were done by computers. (2) The pictures taken on Mars by the Viking lander were reconstructed on earth by computers.

In order to solve a problem, the computer follows a set of instructions called a *program*. The people who write these programs are called, appropriately, *programmers*. The form that the instructions in the program take depends on the programming language used. It is the purpose of this book to teach you to program in FORTRAN. The name FORTRAN is taken from FORmula TRANslation. FORTRAN was developed in the 1950s; as the name implies, it was devised as a language for solving mathematical and mathematics-related problems. FORTRAN is used today to solve problems in mathematics, the physical sciences and engineering, the social sciences and linguistics, and other related fields.

The type of computer used in an overwhelming number of applications solves problems and processes information by manipulating digits; hence, this type of computer is called a *digital computer*. A FORTRAN program is run on a digital computer.

So that you may understand the relation of FORTRAN to the computer, we first briefly describe computers. One way of picturing a computer is as a maze of on-off electrical switches connected by wires. Thus you might imagine that if a programmer wished to instruct a computer to do something, he would have to feed it a program composed of a series of on-off types of instructions. As a matter of fact, the first programs were written like this. The type of language that uses this form of instruction is called *machine language*. In its most primitive form, a program written in machine language consists of strings of 0's and 1's, where a zero represents an open switch, and a one

represents a closed switch. As you can surmise from this brief description, learning to write programs in machine language can be very difficult. Moreover, even once you master it, writing in machine language can be very tedious. For this reason, computer languages closer in form to the spoken word—in our case, English—and to algebra, have been devised. The most widely used of these languages is FORTRAN. Computer languages closer to the machine are called *low level languages*. An example of a low level language is machine language. Computer languages similar in form to how we express ourselves, either by the spoken word or by mathematical symbols, are called *high level languages*. FORTRAN is a high level language.

A program written in FORTRAN cannot be directly understood by the computer; it must first be translated into machine language. A special program does this. It is called a *compiler*, and is already present in the machine when we feed the computer our program. Once a program has been translated, we say that it has been *compiled*. The original FORTRAN program is called the *source program*, and the translated program is called the *object* program.

FORTRAN has grammatical rules that must be followed by the programmer. These rules are similar to those in English that govern the sequence of words in a sentence, the punctuation, and the spelling—we shall learn these rules in later chapters. Before the compiler translates your program, it checks whether you have written your program instructions according to the grammatical rules. If you make grammatical errors in writing an instruction, don't worry; the compiler has been written so that it will inform you of these. Programmers refer to grammatical errors as *compilation errors* or *compile-time errors*. Your program must be free of compile-time errors before the compiler will translate your program.

We now describe the main components of the computer. Essentially, the computer consists of an *input unit*, a *memory unit*, a *logical unit* (or *arithmetic unit*), an *output unit*, and a *control unit*. Our program is communicated to the computer through the input unit. All the mathematics and decisions in the program are done in the logical unit. The program itself and the numbers it processes are stored in the memory unit. The computer communicates the results of our program to us through the output unit. The control unit directs the activities of the other four units. The control unit and the logical unit are referred to collectively as the *central processing unit* (abbreviated as CPU). The word *hardware* is used to describe the physical components of the computer, such as these units, whereas the word *software* is used to describe the programs. We shall use the word *system* to describe the programs, such as the compiler, that process the programs you write.

1.2. The Keypunch

The first, and still the most widely used, means of communicating a program to the computer is to punch the program instructions on a card. The device we use to do this is called a keypunch. In Fig. 1.1 we show a typical keypunch; and in Fig. 1.2, the keypunch keyboard.