BASIC FORTRAN IV PROGRAMMING

REVISED EDITION

DONALD H. FORD

Basic FORTRAN IV Programming

DONALD H. FORD
University of North Dakota

1974
Revised Edition

RICHARD D. IRWIN, INC. Homewood, Illinois 60430
Irwin-Dorsey International London, England WC2H 9NJ
Irwin-Dorsey Limited Georgetown, Ontario L7G 4B3

Revised Edition
First Printing, January 1974

ISBN 0-256-01580-5
Library of Congress Catalog Card No. 73–87258

Printed in the United States of America

Basic FORTRAN IV Programming

Preface

This revised edition of *Basic FORTRAN IV Programming*, like the previous edition, has been written specifically as a textbook for a first course in computer programming. It is designed to teach FORTRAN IV in particular rather than the art of programming in general. The only mathematical background required is a year of high school algebra. I have taken special care to select illustrative examples and problems that the student already understands so that only the art of communicating to the computer need be learned.

My purpose in undertaking this revision has been twofold: (1) to make the subject matter even easier to teach and to learn, and (2) to expand the coverage. To accomplish these objectives, three chapters have been completely rewritten and the remainder have been given detailed editorial attention, illustrations have been improved, and new material has been added.

Coverage in the previous edition was restricted to Basic FORTRAN IV, a subset of the full version of the language. I have expanded this edition to include several of the more useful features of the full version of the language which are not available in the subset (e.g., logical IFs and DATA statements) but, in each instance, the reader is cautioned that a more powerful compiler is required to process such instructions. The version covered in this text equals or exceeds the minimum American National Standards Institute (ANSI) Standard X3.5–1966. It is an ideal version for beginners to learn because it can be used on any IBM System/360 or /370

and, with minor adjustments, it can also be used on almost any modern computer system.

The order of presentation of each topic is that of a general introduction followed by a specific discussion and a series of to-the-point illustrations and examples. American National Standard Institute symbols are used in all flowchart illustrations. Whenever appropriate, previously presented material is briefly reviewed and compared to the topic under discussion. Programming problems have been carefully correlated with the text material and range from the easy to the difficult. Also, an adequate number of problems is provided so that the instructor can alternate program assignments.

Very elementary problems, which should be programmed and processed, are provided at an early point in the text to motivate the student. The purpose of a few easy programs is to develop confidence on the part of the student as well as to provide a background for more sophisticated problems. When the student receives his "first-run" output from the computer center, Appendix A should be a valuable aid. It explains the debugging process in general and illustrates with actual computer output almost every type of error the beginner will encounter. It explains what the error is, the probable cause, and how to correct it.

For the benefit of those students who punch their own program or correction cards, another Appendix explains the operation of a keypunch machine. IBM System/360 and /370 control cards and the composition of a typical elementary "job deck" are also briefly covered in an Appendix.

The first five chapters develop the required technical vocabulary and provide the background necessary to write a variety of complete elementary programs. Beginning in Chapter 5, the material is developed so that each succeeding chapter introduces new "short-cuts" as well as more sophisticated programming techniques. This approach should encourage students to read advance chapters before they are assigned.

It is difficult for a beginning programmer to appreciate the real power of a computer unless he has had experience with arrays. Chapter 8 explains one-dimensional arrays in nontechnical terms and includes a series of illustrative array routines. Chapter 9 on two- and three-dimensional arrays is relatively short but includes enough information so the student can appreciate what they are, how they work, and when they should be used.

The final chapter deals at length with the four types of subprograms but purposely stays away from "high-powered" mathematics. Instead, the emphasis is on the purpose and power of subprograms, how they are written and used, and the fact that many are available to solve a wide variety of problems. A complete listing of the IBM built-in subprograms is included in an Appendix.

It is the intent of this book to teach FORTRAN as directly and quickly as possible and to whet the student's appetite for even more knowledge of

the language. Anyone who has mastered the details of this book will be able to read and use the otherwise incomprehensible FORTRAN IV publications available from the various computer manufacturers.

Grateful acknowledgment is hereby made to the many users of the previous edition who offered valuable suggestions for its improvement. Constructive comments by students, instructors, and reviewers have had a significant impact on the final product.

December 1973 DONALD H. FORD

Contents

1

Introduction

COMPUTER programming can be defined as the art of preparing a plan to solve a problem and of reducing this plan to an explicit sequence of machine-sensible instructions. Programming is essential to the use of computers because computers without programs can do nothing.

Learning to program a computer can be compared to learning to ride a motorcycle. A new motorcyclist must learn to balance on the machine, to start, steer, and stop. He must acquire skill in the use of the various switches, pedals, and controls provided by the manufacturer. He must learn to communicate with the machine to use it as an effective means of transportation. But it is possible to become a skilled motorcyclist without understanding the principle of the internal-combustion engine or the technical aspects of the mechanical and electrical systems. Similarly, the beginning programmer need not understand the many technical concepts of computer design and electrical circuits; he must, however, learn to communicate with the machine to obtain the desired results. To communicate machine-sensible instructions, he must use a programming language and an acceptable communication medium such as punched cards.

This chapter provides an introduction to computer programming languages in general, a brief description of a specific computer system, and a discussion of the punched card.

COMPUTER LANGUAGES

A computer can perform various arithmetic computations such as addition, subtraction, multiplication, and division. It can perform various logical

functions such as comparing two values and determining if the first value is less than, equal to, or greater than the second value, and such as distinguishing plus from minus and zero from nonzero. It can move data from one location in the computer to another. It can also read data and write results. It can do all these things, and more, at fantastic speeds but it cannot think. It is a robot. It must be told when to start, what data to use, what steps to take, what to do with the results, and when to stop. To perform a specific task, it must be given a detailed series of instructions, called a *program,* in a language it is designed to read and obey.

Machine Languages

There are many ways to design a computer or machine to produce an optimum system for specific types of applications. As a result, with few exceptions, each make and model has its own internal coding system and each is designed to recognize or understand one unique language. Thus, a program written in the language of one machine cannot be processed by another.

The unique language which a computer is designed to recognize is called *machine language.* This language consists of machine-sensible instructions which usually take the form of long strings of numbers. These long strings of numbers may be in binary, decimal, octal, hexadecimal, or some other representation depending upon the model of machine. For example, 2A0405 is a valid hexadecimal notation "add command" for an IBM/360 computer and 210042800857 is a valid decimal notation "add command" for an IBM/1620. Some machines are designed to use fixed-length instructions with each instruction requiring the same number of digits; others use instructions of variable length.

The basic characteristics of a machine language are:

1. It is specific in meaning. It is unlike human language where words and phrases may have different meanings in different contexts. With rare exceptions, if a single character in a machine-language instruction is changed, its entire meaning is changed.
2. It has a relatively small vocabulary. This is understandable when one considers the complex mathematical calculations that can be performed using combinations of only the four instructions for add, subtract, multiply, and divide.
3. It is concise. Lengthy human language verbal instructions can be reduced to a few digits.
4. It is machine-oriented rather than human-oriented. It has no relationship to the English language.

Programs written in machine language are called *object programs.* Writing object programs is difficult and tedious. Machine-language pro-

grammers must have a thorough knowledge of both the language and of the internal operations of the computer to be programmed.

Human Oriented Languages

Suppose a Norwegian, who knew only his native tongue, wished to write two letters—one to a German and the other to a Japanese. How would he do it? Perhaps the most impractical approach would be to learn both the German and Japanese languages before he attempted to communicate. Obviously, a much faster and easier method would be to write both letters in Norwegian, then hire professional translators to convert the letters into the other languages.

A person wishing to write instructions to a computer may be faced with the same problem as the Norwegian. He knows how to express his instructions in the English language, but the machine language understood by the computer is foreign to him. He could, of course, employ a machine-language programmer to perform the translation, thus solving his communication problem. But, consider the advantages if he had a computer program which was designed to translate English language, accurately and without error, directly into machine language! In a sense, he could then "talk" to a computer.

Unfortunately, no computer programs exist that will translate ordinary English language into machine-language instructions. Attempts have been made to write such programs but not with complete success. These attempts, however, have resulted in translation routines that can make the programmer's job much easier. Human-oriented languages have been developed that are composed of letters, symbols, and numbers, grouped into various combinations to form a limited vocabulary of English and pseudo-English words and expressions. These languages are much easier to learn and to use than the machine language of the computer. The development of human-oriented languages has had a significant influence on the rapid advancement of computer technology.

Translators. A program written in a human oriented language is called a source program. The *translator* is a special program, usually supplied by the manufacturer, that converts a human-oriented source program into a unique machine-language object program for the computer model on which it is translated. Source programs, for example, translated on an IBM/1620 computer result in /1620 object programs; source programs translated on an IBM/360 computer result in /360 object programs. This may appear to involve more work because a program must be written in one language, then translated into another language before it can be processed. It does involve more work, but the extra work is done by the computer at electronic speed. Use of a translation routine results in at least two major advantages. First, the programmer may be required to learn only one

language rather than a different language for each computer make and model used. Second, it is faster and easier to write programs in a human-oriented language.

Currently, two broad classes of translators exist—one is called an assembler, the other a compiler.

Assemblers. An assembler is used to translate a low-level human-oriented language, called a *symbolic* language, into machine language. Symbolic languages generally use mnemonic (meaning *memory aid*) codes such as A for add, S for subtract, M for multiply, etc. An assembler is usually designed for a specific computer model. It generally converts symbolic language source programs into machine-language object programs on a one-for-one basis; that is, for each symbolic language instruction the assembler generates one corresponding machine-language instruction. This one-for-one translation makes it possible for the programmer, if he chooses, to match his symbolic language instructions to the translated machine-language instructions to see what occurred.

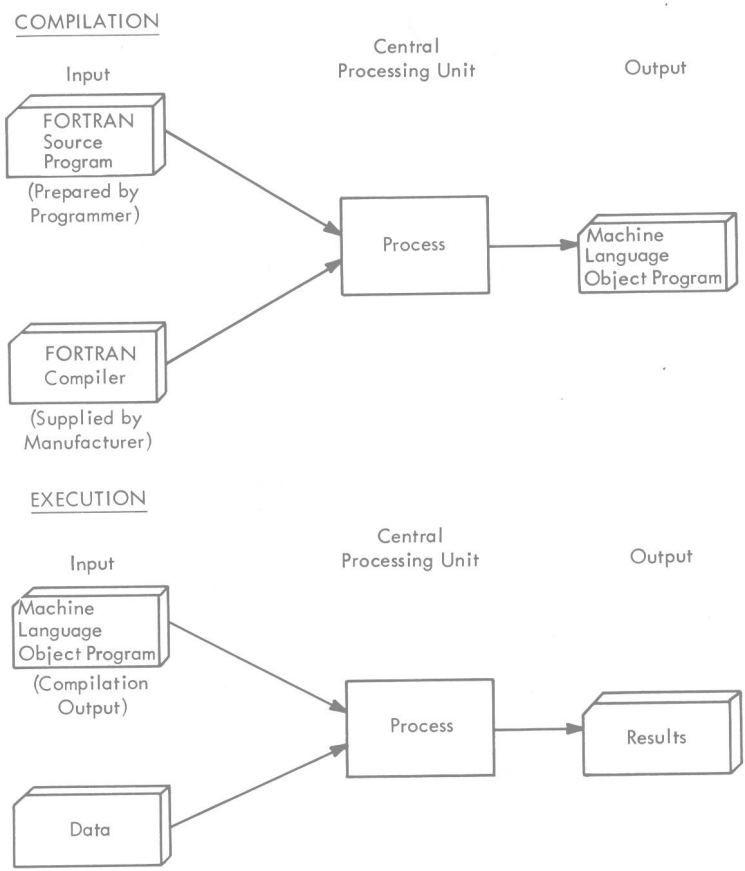
Compilers. A *compiler* is used to translate a high-level human-oriented language called a *problem-oriented language*. A compiler is more powerful than an assembler. It may generate or "compile" a list of many machine-language instructions for each problem-oriented language instruction. Thus, it is not an easy task to compare a series of high-level language instructions with the translated machine-language instructions, but, fortunately, such a comparison is rarely necessary. The use of high-level language does not require a thorough knowledge of the intricacies of the machine. Problem-oriented languages are not designed for a specific computer; they are designed to be used in solving a special class of problem. They are written in pseudo-English (example: ADD A, B GIVING C) or in common algebraic notation (C = A + B) rather than in mnemonic code.

The beginning programmer might ask, at this point, why all programs are not written in a high-level language. The answer to this question is that high-level languages have inherent disadvantages that, in some applications, may outweigh the advantages. In general, a program initially written in machine or assembler language will use more efficient code, will make better use of specific machine input/output capabilities, and will tend to use instructions that will execute faster at object time than a compiler-translated program.

FORTRAN, the subject of this book, is a high-level language. Thus, programs written in FORTRAN must go through a "double run" on the computer. The first run is the *compilation* or translation run. This results in a FORTRAN source program being converted into a machine-language object program. The second run is the *execution* or computation run. This results in a machine-language object program processing data and pro-

ducing the desired results. This can perhaps best be illustrated schematically. (See Figure 1–1.)

FIGURE 1–1
Compiling a FORTRAN Source Program and Executing the Object Program



Large-scale computers can accept both a source program and problem data in one run. The intervening steps are handled automatically, but the logical procedures are the same as illustrated in Figure 1–1.

FORTRAN

Many human-oriented languages are in existence. This book is concerned with the one developed by IBM and originally published in 1957. Called FORTRAN, an acronym for FORmula TRANslation, it is probably

the most widely used problem-oriented language. It is often referred to as a "scientific language" and is designed to permit complex mathematical expressions to be stated similarly to regular algebraic notation. It has become an extremely popular language because of the ease with which it can be learned and because of the wide variety of applications for which it has been found suitable.

FORTRAN is not a dead language—it is subject to change. During its brief life-span, FORTRAN has been improved by modifications, additions, and deletions. As a result, FORTRAN, like ordinary human languages, suffers from the existence of several versions and dialects; this book uses a most current version called FORTRAN IV.

The American National Standards Institute, Inc. (ANSI), has established a FORTRAN IV standard that is intended as a guide for manufacturers, consumers, and the general public. Conformance to this standard is not required; furthermore, there are many computer makes and models which differ both in size and in engineering design. Thus, some ANSI features are not present in all compilers whereas others go beyond the standard. It should be noted, however, that ANSI has had a major influence on the development of FORTRAN. As a result, any programmer who has mastered one current version of the language (such as FORTRAN IV for IBM System/360 and /370, which is used in this book) will find it relatively easy to learn the slight variations in other versions. Thus, in general, a FORTRAN programmer can communicate with any computer having a FORTRAN compiler. For this reason, FORTRAN is said to be "machine-independent" in that it may be written without regard for the specific make or model of machine on which it will be processed.

A COMPUTER SYSTEM

As previously indicated, one of the advantages of FORTRAN, from the beginning programmer's point of view, is that a thorough knowledge of the intricate details of various computer systems and designs is not required; but it is useful to have at least a general understanding of the machine to be programmed. This section illustrates what an IBM System/360 looks like and briefly describes how it works. Although a specific machine is used for illustrative purposes, the general concepts apply to all modern computers, regardless of make or model.

Hardware

The physical equipment that makes up a computer system is called *hardware*. A minimum configuration of hardware consists of a central processing unit, one input device, and one output device. Generally speak-

ing, the more input and output devices a computer system has, the more jobs it can do. Figure 1–2 illustrates the IBM System/360.

FIGURE 1–2
IBM System/360

Photo courtesy of IBM Corp.

Central Processing Unit (CPU). The CPU is the actual computer. It has three basic internal components. The *"memory" unit* (described in the next section of this chapter) serves as a storage area for both the information being processed and for the program instructions. Mathematical calculations are performed within the *arithmetic unit*. The *control* or *supervisory unit* directs the overall operations of the computer system and serves as a coordinator between the other internal units and the input and output devices. One end of the CPU exterior has a control panel with various switches and dials used to operate the machine. A light panel instantly indicates to the expert what the computer is doing, or it can be used to indicate the contents of any particular location within the computer.