UPDATED PRINTING

Gordon B. Davis
Thomas R. Hoffmann

FORTRAN 77
A Structured, Disciplined Style

THIRD EDITION

THIRD EDITION

FORTRAN 77: A Structured, Disciplined Style

Gordon B. Davis

Thomas R. Hoffmann

University of Minnesota

McGraw-Hill Book Company

New York St. Louis San Francisco Auckland Bogotá Caracas Colorado Springs Hamburg Lisbon London Madrid Mexico Milan Montreal New Delhi Oklahoma City Panama Paris San Juan São Paulo Singapore Sydney Tokyo Toronto

FORTRAN 77: A Structured, Disciplined Style

234567890HESHES89321098

ISBN 0-07-015905-x

This book was set in Clearface by Better Graphics, Inc.
The editors were David Shapiro and Sheila Gillams;
the cover was designed by Caliber Design Planning, Inc.;
the production supervisor was Denise L. Puryear.
New drawings were done by Caliber Design Planning, Inc.
D. B. Hess was printer and binder.

Library of Congress Cataloging-in-Publication Data

Davis, Gordon Bitter.
FORTRAN 77: a structured, disciplined style.

Includes index.
1. FORTRAN (Computer program language)
2. Structures programming. I. Hoffmann,
Thomas Russell, (date) II. Title.
QA76.73.F25D385 1988 005.13'3 88-2993
ISBN 0-07-015905-X

Preface

The FORTRAN language is over 30 years old. It was first released to customers of IBM 704 computers in April 1957. At that time, it was a powerful innovation. Because of the introduction of other languages, many have predicted that FORTRAN would fade and die. It is a testament to its basic value that it continues to flourish. It is the language of choice for supercomputers (Cray released CFT 77, full FORTRAN 77 implementation for its supercomputers in 1986). The FORTRAN implementations for microcomputers continue to improve; in 1987, Microsoft released a full FORTRAN 77 for IBM PC–compatible microcomputers. There are competitor languages that are superior to FORTRAN for some types of problems, but FORTRAN continues to be an excellent language for a wide variety of mathematical problems. For other areas such as robotics and engineering design, it is the major language.

The third edition maintains the direction of the second edition but with several changes to reflect the current environment for programming in FORTRAN. FORTRAN has not changed; FORTRAN 77 is still the current standard for all FORTRAN compilers. However, the environment for programming has changed significantly.

This revision retains the innovative pedagogical approach of the first two editions in which the programming concepts and statements are described in part *a* of each chapter and the concepts are illustrated in part *b* of each chapter by two complete program examples. There is an emphasis in both the general explanation and the example notes on a disciplined, structured style for programming in FORTRAN. Students learn and use the disciplined style from their very first small program. Experience has shown that students who learn with this pedagogical approach write disciplined programs and never develop bad habits which they must unlearn later.

The use of the complete program examples, plus frequent self-testing quizzes with answers at the end of the chapter, allows the text to be used for self instruction or for a course that emphasizes student initiative in learning the language. It is excellent for a traditional lecture course because it describes and illustrates features in good program design. This allows the instructor to focus on general programming concepts, algorithm development, purposes and uses of FORTRAN statements, and reinforcement of both theoretical and practical principles of FORTRAN program design.

The major change in the third edition is in the explanation of how programs are entered into the computer and how data items and records are provided for program execution. Changes in the programming and execution environment for FORTRAN programs are reflected in the following changes in the text:

1. The Introduction, "Preparing to Program in FORTRAN," explains the various alternatives for preparing and entering a FORTRAN program into the computer and the alternatives for providing data for program execution. The introduction explains the differences between programming in FORTRAN with a microcomputer and using a mainframe computer. There are some very simple program examples that readers can use to explore the capabilities of the FORTRAN compiler and the computer available to them.
2. All references to use of punched cards for input and output have been removed because punched cards are no longer used.
3. The text explanations describe the two major methods for entering a program into the computer: using a text editor or using a terminal or workstation in interactive programming mode. The introduction explains the use of a text editor and interactive programming.
4. Explanations include FORTRAN on both mainframe and personal computers. Appendix A describes differences in implementations of FORTRAN and includes a section on FORTRAN for a personal computer.
5. The two example programs in part *b* of each chapter have been redone to illustrate the two alternatives for execution of the program: batch execution using a previously prepared data file or interactive execution in which data items are entered at a keyboard when requested by the program. In most cases, one program is designed for batch execution and the other is designed for interactive execution. The notes describing the example explain the reason for the differences in program design.
6. Additional scientific and engineering problems have been added to each chapter that specify interactive execution.

The text adheres to the 1977 American National Standard (ANS) FORTRAN. Most of the current FORTRAN compilers follow this standard, but many small computers will have compilers that implement only a subset version of the 1977 standard FORTRAN. The full 1977 FORTRAN is presented in the text, but features in the full version not included in the 1977 subset FORTRAN are noted in the chapters and summarized in Appendix A.

Color is used to highlight key explanations and reference boxes in the text.

Key features of the second edition that are continued in the third edition are:

1. Easily accessible reference material, such as a list of intrinsic functions on the inside front cover and a reference list of features, each with an example statement showing its use, following the index as the last pages in the text.
2. Use of the inside back cover for recording and easily referencing specifications for the FORTRAN compiler being used.
3. A large number of problems suited to different disciplines. These are tested problems; a solution has been written for each one to check for completeness of the problem statement.
4. The use of list-directed input and output for the first two chapters. This allows students to write a program without learning the FORMAT statement. The

emphasis is on writing complete, well-structured programs starting with the first program.

5. The emphasis on using features that reduce the possibility of programming errors.
6. Material on debugging presented in each chapter as the level of difficulty of programs increases.

There is an Instructor's Manual that contains a sample solution for each of the over 120 programming problems. There are also suggestions for using the text.

We would like to express our thanks for the many useful comments and suggestions provided by colleagues who reviewed this text during the course of its development, especially to, Joyce Brennan, University of Texas at Austin; Henry Etlinger, Rochester Institute of Technology; David Franklin, Southern Technical Institute; George Main, Bellvue Community College; Robert Dourson, California Polytechnic State University; and Rebecca Rutherford, Southern Technical Institute.

The revision has been aided by Janice DeGross who did manuscript typing and preparation of figures and by Timothy Hoffmann who provided assistance in reviewing the manuscript and added his insight as a commercial FORTRAN user.

Gordon B. Davis
Thomas R. Hoffmann

FORTRAN 77: A Structured, Disciplined Style

Contents

Introduction: Preparing to Program in FORTRAN

The purpose of this introductory chapter is to prepare for programming in FORTRAN. There is a short background section on the FORTRAN language. This is followed by a survey of hardware, software, files, and programming languages as background for programming. The text emphasizes the value of a structured, disciplined approach to programming, and the next section explains the objectives and features underlying it. The different methods for entering and executing computer programs are then explained in order to assist the beginner in understanding the characteristics of the computer system and the FORTRAN system being used.

There are two sections describing information needed by a programmer in order to enter FORTRAN programs and execute them. Finally, a simple seven-statement FORTRAN program is given to aid in testing your understanding of the FORTRAN system you are using.

The FORTRAN Language

FORTRAN (an acronym for FORmula TRANslator) is the most widely used of a class of high-level languages called scientific or algebraic languages. (The meaning of *high-level language* will be explained in the chapter.) It is available for use on almost all computers. Although not limited to mathematical problems, it is especially useful for problems that can be stated in terms of formulas or arithmetic procedures. This covers a wide range of problems. For example, FORTRAN is suitable for such diverse problems as analysis of sales statistics in a business and analysis of structural stress for designing a building.

Development of FORTRAN

FORTRAN was developed in 1957 by IBM in conjunction with some major users, but it is now used by all computer vendors. FORTRAN has changed and evolved. This evolutionary process resulted, during the development period, in several FORTRANs of increasing complexity. Major versions were called FORTRAN, FORTRAN II, and FORTRAN IV. Each new version made a few changes in the basic instructions and included additional features. In 1966, a voluntary FORTRAN standard, American National Standard (ANS) FORTRAN, was adopted. The International Standards Organization (ISO) also defined standard FORTRAN.

A revised American National Standard (ANS) FORTRAN was adopted in 1977. This 1977 standard adds features to the previous 1966 standard FORTRAN, clarifies some ambiguities, and makes a few minor changes. The standard also defines two different levels for FORTRAN implementation: subset FORTRAN and full FORTRAN. Subset FORTRAN is a compatible subset of the comprehensive full FORTRAN. This text is based on the 1977 standard adhered to by almost all implementations of

FORTRAN in use today. The text concentrates on the most-used features of FORTRAN; some advanced or little-used features will be summarized in less detail in Chapter 7.

Concurrent with the development of standard FORTRAN has been the development of special teaching-oriented FORTRAN compilers. The best known of these, developed at the University of Waterloo (Waterloo, Canada), are termed WATFOR and WATFIV. There are similar teaching FORTRANs for many other large computers. The teaching-oriented systems were designed to provide excellent error-diagnostic messages for students and to do fast execution of small student programs. For production applications, FORTRAN is available on all large-scale systems by IBM, Control Data Corporation, Unisys, Digital Equipment Corporation, etc. For supercomputers, such as those from Cray Research, FORTRAN is a primary language.

Although almost all versions of FORTRAN adhere to the 1977 standard, they frequently provide extra nonstandard statements and features. This text takes the view that programmers should generally avoid use of nonstandard features so that their programming knowledge is not limited to a particular FORTRAN implementation.

FORTRAN is available for personal computers. The most common is by the Microsoft Corporation (often termed Microsoft FORTRAN). It is also based on 1977 standard FORTRAN but contains a few extra features to deal with the limitations or take advantage of a personal computer. Although early versions did not implement all the features of standard FORTRAN, Microsoft Version 4.0 does so. Other fully standard micro-based FORTRANs include IBM's Professional FORTRAN, F77L by Lahey Computer Systems, and FORTRAN by Ryan-McFarland Corporation. The 1977 American National Standard FORTRAN is therefore recommended as the basis for all FORTRAN programming, by students as well as by professional programmers.

How to Study FORTRAN Using This Text

FORTRAN is a machine-independent language for instructing a computer. In other words, the programmer writing FORTRAN does not need to know any machine-level details for the computer being used. The language is procedure-oriented—designed for instructing the computer in a problem-solving procedure. The language consists of a vocabulary of symbols and words and a grammar of rules for writing procedural instructions. The symbols, words, and rules utilize many common mathematical and English-language conventions so that the language is fairly easy to learn and to understand. The rules are, however, precise and must be followed with care. In other words, learning FORTRAN is like learning a special-purpose language. There are rules of construction and vocabulary to learn, and one becomes proficient by doing rather than by much reading.

The objective of the text is to assist you in learning to write clear, understandable, error-free FORTRAN programs. To achieve this objective, you need to do each of the following:

1. Learn the instructions and other elements of the language.
2. Learn how to apply the language rules and to use the FORTRAN instructions to compose a clear, understandable program.
3. Learn how to enter a FORTRAN program (at a terminal or PC) and how to specify that your program be translated (compiled) and executed (using the computer that is available to you).

The first objective is met by part A of each of the chapters that follow this introduction. There is a description of instructions or rules, and illustrations of their use are provided. To assist in learning, there are self-testing exercises after every major unit in each chapter. At each self-testing exercise, answer the questions and check your responses against the answers at the end of the chapter.

The second objective is achieved by part B of each chapter, which contains complete programs. You should study the example programs, noting the style, the error-control features included in the program, and the documentation describing the programs. These examples of good programming style provide a pattern to follow in writing your programs.

The third objective requires the programming of a problem, carrying out the procedures for entering the program, instructing the computer to compile and execute, and removing errors. A variety of different programming problems is provided with each chapter. In carrying out the writing, coding, running, etc., of a program, the text provides the following aids:

1. A reference list of all FORTRAN language features. (List of 1977 American National Standard FORTRAN Statements and Specifications follows the Index.)
2. A place to record specifications and features you need to know about the FORTRAN system for the computer you will use (including how to enter a program and how to instruct the system to compile and execute it). The form to record these is on the inside of the back cover. The specifications should be filled in from material furnished by your instructor or obtained by consulting the FORTRAN reference manual for your computer.
3. A reference list of all FORTRAN intrinsic functions on the inside of the front cover (to be explained in Chapter 2). Check off those functions that are available to you (you can wait to do this until after completing Chapter 2).

The text is based on the generally accepted 1977 American National Standard FORTRAN, but because the fundamental features of all versions of FORTRAN are the same or nearly the same, it is possible to understand and modify programs written using the older 1966 FORTRAN standard. Appendix A assists in understanding differences in these versions of FORTRAN. Implementations of FORTRAN in a personal computer generally include all the features of standard subset FORTRAN and a few additional features. Differences between standard FORTRAN and some microcomputer versions are also described in Appendix A. Before proceeding, scan the front and back endpapers, the two appendixes, and the list of 1977 American National Standard FORTRAN Statements and Specifications (follows the Index).

Instructing a Computer

Before starting the detailed explanation of the FORTRAN language, it may be helpful to review how a computer is instructed. A computer system requires both hardware and software. The *hardware* consists of all the equipment; the *software* includes the programs of instructions that direct the operations of the computer equipment. The computer hardware and software make use of files that contain programs and data.

Hardware, Software, and Files

A computer system can range in size from a large one serving hundreds of users concurrently to a personal computer with one user. A large computer system, shown in Figure 1, has input terminals connected to it over communication lines, a central processing unit (CPU), internal storage (memory), and output units (such as a

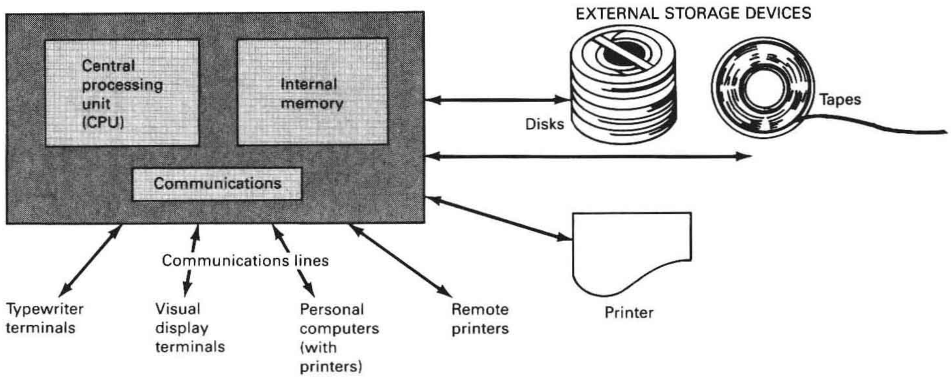

Figure 1 Large computer hardware system.

printer). It also has external storage devices, such as magnetic disks or magnetic-tape units (also called secondary or auxiliary storage). In a typical processing job, data comes from the input unit (and perhaps from external storage) into the central processor where computation and other processing is performed. After processing, the results are sent to the output device (say, a printer or terminal) or to a secondary storage device to be held for later output or further processing. Note that a terminal may be used for both input and output. The terminal may be a typewriterlike device, a visual display device with a televisionlike output display and a keyboard for input, or a personal computer equipped with communications capability. The visual display terminal will tend to be used for small amounts of output; larger volumes of output will generally be printed.

A personal computer (often called a PC or desktop computer) has the same basic parts as a large computer, but it is smaller and more compact (Figure 2). The cabinet contains the central processing unit, internal storage, and, usually, secondary or auxiliary storage (devices for "floppy" diskette and/or "hard" disk storage). There is a display device with keyboard for input and output and a separate printer for hard-copy (printed) output. For FORTRAN, the PC may be used by itself or as a terminal to a larger computer.

The CPU can perform operations such as read, write, add, etc., but the sequence in which these operations are to be performed and the specific input and output units to be used are specified by a set of instructions stored in the computer memory. The general term applied to these computer processing instructions is *software.* The instructions are organized into sets called routines and programs. A *routine* refers to a set of instructions that directs the performance of a specific task, such as calculating the square root of a quantity or producing an error message when an error is encountered in input data. A *program* consists of one or more routines that direct the solution of a complete problem.

There are several major types of software. Three types especially relevant to the study of FORTRAN are application programs, compilers, and operating systems.

1. *Application programs* Programs that direct the processing for an application of computers, such as preparing payroll, preparing checks, calculating a rate of return for a proposed project, calculating stress factors for a building structure, etc. The FORTRAN examples in this text are application programs. Application programs may be written by personnel in organizations needing them, or they may be purchased from software vendors who prepare and sell programs for frequently encountered applications.
2. *Compilers* Programs that translate instructions written in a high-level, general-

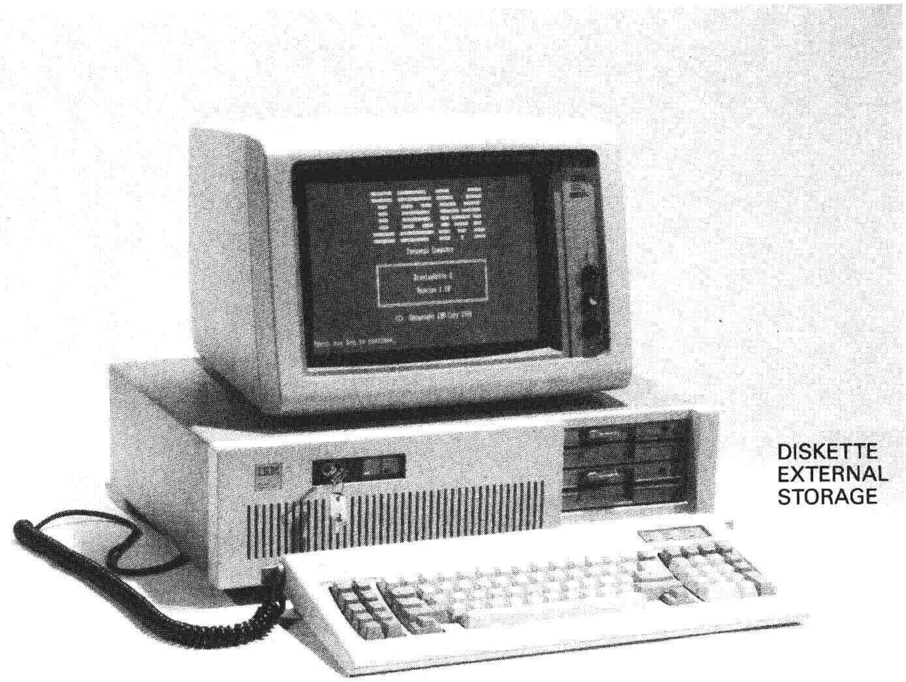

Figure 2 Personal computer.

purpose language, such as FORTRAN, into a set of machine instructions specific to the computer being used.

3. *Operating system* A set of routines that directs and manages the operations of the computer. The operating system supports and directs the running of application programs. For example, if an application program has an instruction to print output on the printer but the printer is not operable, the operating system sends a suitable message to the operator.

Operating systems and compilers are generally obtained from either the hardware vendor or independent software vendors. Programs being executed are stored in the internal (main or primary) storage or memory, which is part of the central processing unit. Programs not currently being executed but which need to be available are stored as files in external storage. Some parts of the operating system remain in main storage all the time. Based on job control instructions to be described later, these operating system routines bring into internal memory the routines and programs to be executed and direct their execution. The programs to be run may be compilers, application programs, or other software.

A computer file is a set of stored data. The set of data can be text, such as a report or the lines of a computer program, or the data can be numeric. Files will be explained in detail later; the feature of a file that is important for this chapter is its use as a way of storing a program or saving a set of data to be used by a program. Each file is assigned a name by the person who creates it. The program file is retrieved by referring to its name.

Machine-Level Languages

A program in main memory must be in machine language to be executed. A machine-language instruction is represented as a string of binary digits called *bits* (represented by 1s and 0s), which identify the operations to be performed and the data, etc., to be used. The machine-level instructions differ for different series of computers and different manufacturers. As an example, a typical instruction for a large IBM computer has the following form (with 1 standing for a 1-bit and 0 for a 0-bit in storage):

0101101000110000101110100100000

Even though the internal machine representation is in this form, it would be very difficult and could lead to error if a programmer were required to deal with such instructions.

Because binary notation is difficult to use, a programmer who codes programs in machine-level instructions generally uses a symbolic assembly language. Symbolic assembly languages as a class are often referred to as *low-level languages.* These languages are machine-oriented because each symbolic assembly instruction is converted into one machine-language instruction. The preceding machine-language instruction coded in symbolic assembly language might be: A 3,PAY where, for example, A means "add." The computer cannot directly execute the symbolic instructions, so these must be translated into machine-language instructions. This is done by a program called a *symbolic assembly system* which converts each symbolic instruction into an equivalent machine-level code instruction. Machine-oriented programming is very useful for some applications because instruction coding can be very machine-sensitive and thus obtain very efficient use of the computer. However, an assembly language program is relatively difficult to code, and logic errors are difficult to find. It is also difficult and time-consuming to change. A program in a low-level, machine-oriented language also has limited transferability (portability) from one computer to another.

High-Level Languages

A *high-level language* is oriented to problem solution or processing procedures rather than to the machine-level instructions of a particular computer. The instruction statements use words, phrases, and symbols that are similar to those commonly used to describe solution or processing procedures. Another major difference between a high-level instruction and a symbolic assembly instruction is that one high-level instruction is translated into many machine-language instructions.

There are a number of different high-level languages for different types of problems. Each of these languages consists of a grammar (set of rules) and predefined words for writing instructions. A program called a *compiler* is used to translate the program written in the high-level language (the source program) into machine-level instructions (the object program) for the computer on which the program is to be run. Since machine-level instructions differ among computer series, there must be a unique compiler program for each computer series. For example, there are FORTRAN compilers written for each series of the IBM computers, others written for the series of VAX computers, and yet others written for personal computers.

There are two important advantages of high-level languages over symbolic assembly languages: They are machine-independent in the sense that programs written in a high-level language can be compiled and run on any computer (for which there is a compiler) with few or no changes, and they are relatively easy to learn. Today, these languages are generally so powerful and efficient that they have virtually eliminated the need for symbolic assembly language coding except for a few specialized applications. It is also relatively easy to standardize methods of programming with high-level languages. Organizations having a concern with program accuracy and a desire for programming discipline have strongly influenced the trend toward use of high-level languages.

The two most common high-level languages are FORTRAN and COBOL: FORTRAN is best suited for formula-type mathematical problems, while COBOL is the dominant language in business data processing. Other high-level languages having significant use are BASIC, Pascal, PL/1, Ada, and APL.

A Structured, Disciplined Style in FORTRAN Programming

The title of this text indicates that there can be a structured, disciplined style in writing FORTRAN programs. Since the text follows this approach, it will be useful to understand the reasons for the approach and the basic methods to implement it.

Computer programs frequently do not meet user requirements, are not produced on time, cost considerably more than estimated, contain errors, and are difficult to maintain (to correct or change to meet new requirements). These difficulties have been observed with such frequency that many organizations have attempted to change the practice of programming in order to improve performance. The revised approach can be termed a *programming discipline*—well-defined practices, procedures, and development control processes. A student should not merely learn to code FORTRAN statements. It is equally important to learn how these statements are combined into a high-quality program—one which is easy to understand (and change, if necessary) and which uses computer resources efficiently.

Objectives of a Disciplined Approach to Programming

Because programming discipline is an underlying philosophy for this text and because of the importance of programming discipline to industry, it will be useful to summarize the major objectives of this approach to programming:

1. *Meet user needs* A program has a purpose, such as to produce an analysis or to compute a set of statistics. An assignment to prepare a program is a failure if the program is not used because the potential users of the application find it too complex or too difficult. A disciplined approach to program design includes a careful analysis of requirements before programming.
2. *Development on time within budget* Estimates of time and cost for writing computer programs have frequently been substantially in error. By the use of a more structured, disciplined approach, installations have achieved dramatic improvements in productivity and have improved their ability to estimate time to complete.
3. *Error-free set of instructions* It is generally considered that all large-scale computer programs contain errors, and it may be impossible to remove every single error from a large set of programs. However, using a disciplined, structured approach, programs may be designed and developed in a manner that minimizes the likelihood of errors and that facilitates detection and correction of errors in testing. The result can be virtually error-free programming.
4. *Error-resistant operation* A program may produce erroneous results, due either to program errors or to incorrect input. The program should be designed so that errors will, whenever possible, be detected by the program itself during execution. The design features to assist in programmed detection of errors are:
   (*a*) Input validation. This is a process of testing all input data items to determine whether or not they meet the criteria set for them. For example, data input may be tested for: