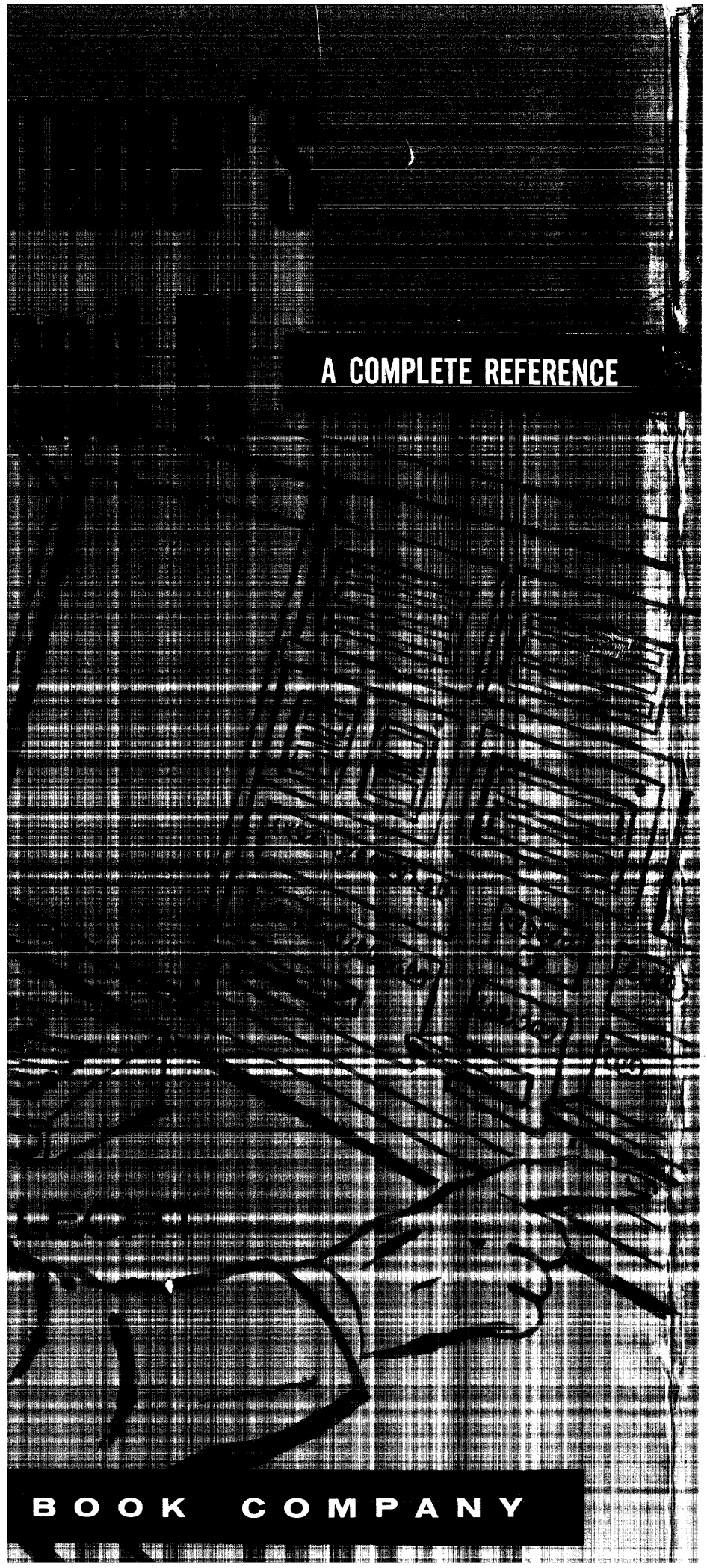

$7.95

The Programmer's FORTRAN II and IV

A COMPLETE REFERENCE

By CHARLES PHILIP LECHT

Director, Advanced Computer Techniques Corporation

Here is the most complete guide available on the FORTRAN II and FORTRAN IV programming languages. A thoroughly detailed reference, it presents both languages in such a way as to define each independently and to indicate differing characteristics. The book is especially geared to the needs of the practicing computer programmer, but is also useful as a learning tool.

Since the FORTRAN language has relatively few instructions in it, the problem in learning to program in the language does not rest in simply memorizing the format and meaning of each instruction. It consists rather of the tremendous detail the programmer must remember about the characteristics of the instructions. This essential book makes all this information readily and constantly accessible.

Written by an expert in computer documentation, the book shows clearly and concisely the full extent, meaning, and limitations of each type of statement. It describes what an action will *not* do as well as what it will do, and points out possible misuses of information which can be caused by inference, plausibility considerations, or even logical deduction.

(continued on back flap)

(continued from front flap)

The core of the book is FORTRAN II. FORTRAN IV information is provided and labeled as such when it extends or limits the scope of a particular FORTRAN II concept . . . or changes the concept significantly. Thus FORTRAN II material may be considered FORTRAN IV material except when otherwise indicated.

The clear, easy-to-read format and unusual accuracy of detail make this book ideally suited for use as a reference. As an added aid, words requiring definition are marked with an asterisk and are defined in a glossary at the end of the book.

Photo by Richard C. Robey

CHARLES PHILIP LECHT, founder, President, and Technical Director of ADVANCED COMPUTER TECHNIQUES CORPORATION, received his start in the computer field in 1955 as a programmer for IBM. From there he joined M.I.T.'s Lincoln Laboratory as a technical Staff Member, where he worked on the systems design, analysis, and programming of the SAGE data reduction effort. Subsequently he held the positions of technical Staff Member, The MITRE Corporation; Chief, Programming Division, and Chief, Mobilization Application Division, Ordnance Industrial Data Agency; Executive Vice President, ACT.

Among his many professional achievements have been an election prediction program, two ballistic missile and satellite simulation models, systems analysis and programming of the Particle-In-Cell hydrodynamics model operating on the UNIVAC LARC, and supervision and design of various operating systems.

THE PROGRAMMER'S FORTRAN II AND IV

THE PROGRAMMER'S FORTRAN II AND IV: A Complete Reference

CHARLES P. LECHT
Director, Advanced Computer Techniques Corporation

With a foreword by
ROBERT BEMER, *General Electric Company*

New York
San Francisco
Toronto
London
Sydney

McGRAW-HILL BOOK COMPANY

THE PROGRAMMER'S FORTRAN II AND IV

 Library of Congress Catalog Card Number 66-15182

36965

123456789HD721069876

To Eric and the Colonel

FOREWORD

I would hope that most readers of this book are already familiar with FORTRAN, more familiar than just knowing that it is an acronym for FORmula TRANslation. This is for a reason opposite to that which may come first to mind. It is not that this book is so difficult that only someone already grounded in its usage can comprehend it easily. No, my reason is that FORTRAN is widely used and but narrowly understood.

I recall that nagging worry with which I surveyed certain fellow students in my undergraduate mathematics courses. They seemed to assimilate each new concept without effort, while I was struggling to relate to numerical and spatial examples. As a working mathematician, however, I discovered that I had been subconsciously synthesizing and integrating, whereas they had been merely adding layer after layer.

We see this superficiality when the beginning computer user is first exposed to the power of both FORTRAN and the machine. The language is relatively easy to learn, for much of it is a simple mapping from common mathematical terminology and usage. The beginner becomes drunk with the power at his fingertips, but let him not forget that it is expensive power, easily squandered unwittingly. Should Parkinson need additional proof for his theories, the FORTRAN community exists.

One of the best things about this book is that it was written by an expert in computer documentation. This is a significant statement, for less than 1 per cent of the world's programmers document carefully and correctly. Those of us that do may be motivated by the mileage to be gotten out of what we conceive; the programmer who documents poorly is soon discarded in disgust, and so are his achievements. A main principle of programming documentation is what we call "positive negation." The computer forces us to this. It isn't enough to say what a certain action will do; one must also state precisely what it won't do, and positively (i.e. to point out possible contextual misusesof the information, the occurrence of which may be caused by inference, plausibility considerations, or even logical deduction). This may well be the essence of this book. It is not a text to teach you to program in FORTRAN--you may learn this as a byproduct. The intent is to show clearly and concisely the full extent, meaning and limitations of each type of statement in the FORTRAN language, so that you may synthesize rather than add layer after layer.

There are many programmed processors for the various computers, each of which translates a certain variant of FORTRAN to machine

language with varying efficiency in the running program created. But in each variant there are many ways of expressing the problem solution, from efficient to inefficient, much as one can get to a point two blocks away by walking twelve, or stutter and be circumlocutive rather than concise. It is the responsibility of the fabricators of these processors to explain such idiosyncrasies to the user in texts companion to this volume. The purpose here is rather to prepare the user in the most general way to understand the effect of these variations, as applied to specific computers.

There exists but one brief history of FORTRAN, given by W. P. Heisling in the 1963 March issue of The Communications of ACM, pages 85 and 86. As one familiar with the intimate history, I should like to set it down here in a less formal way.

The computer gradually impressed upon its early users something they should have realized all along--that it was indeed a device of unlimited applicability, and that one very important application might be to operate upon the expression of an algorithm (or problem solution) in a language convenient to humans and render it into a language convenient to machines. Remember that the natural language of humans is imprecise. It gains understandability from many other devices, such as inflection, hand-waving, redundancy, relation to previous conditions, alternate phrasing and the like. A compromise language between humans and machines, of which FORTRAN is one, must have certain artificial characteristics which make it deterministic without needing such devices.

It is typical of the computer age that very few innovations are the product of a single person. It is rather as if the very nature of computers led us all along an inevitable path of understanding. There are few developments in computer languages or processors that cannot be found in embryonic form in the earlier programs of a dozen people. Some first glimmerings came when Dr. Grace Murray Hopper and her associates produced the A0 compiler (1952 May) for the UNIVAC® I, further extending it to A2 (1953 August) and then to AT3 (or Math-Matic, 1956 June), which had a limited form of algebraic statement. Perhaps an even earlier worker was Dr. Heinz Rutishauser of Switzerland who, unknown to U.S. workers, developed a FORTRAN-like compiler for the Zuse 4 computer in 1951, although it did not receive any extensive usage. R. A. Brooker's Autocode for the Manchester (Ferranti) Mercury handled statements of a limited arithmetic type. Laning and Zierler developed an algebraic system for the M.I.T. Whirlwind somewhere around 1953 or '54, although I have not been able to place an operative date for it. Also setting the stage were many interpretive mathematical systems for the IBM 650, 701 and the Datatron 205.

Dr. Charles DeCarlo, then Director of Applied Science for IBM, was impressed enough to set up a development group under John Backus, who had done the Speedcoding system for the 701. This group was organized in the summer of 1954 and brought its work to a usable condition in 1957 January. Other members were Dr. David Sayre, Robert Beeber, Sheldon Best, Dr. Richard Goldberg, Lois Haibt, Harlan Herrick, R. A. Nelson, Peter Sheridan, Harold Stern and Irving Ziller, together with Roy Nutt of United Aircraft and Robert Hughes of the Livermore Radiation Laboratories.

This group was charged not only with building an algebraic compiler, but also with proving that a compiler could produce optimized object code (running programs) comparable in efficiency to those of the best hand coders. As a result, the original effort took 25 man years of effort over a two and one half year period, at an initial cost of a half million dollars. Today we can do better at an eighth of the cost, but the 707 flies faster than the Wright airplane, too. One cannot say enough for Dr. DeCarlo's vision, for he insulated and protected the FORTRAN group for this long period when the use of computers was expanding so rapidly that any good programmer was desperately in demand. It was good judgment, too, for today IBM has a yearly income from FORTRAN alone in excess of 300 million dollars, which was almost their entire income in 1957 when FORTRAN was introduced without fanfare or too much confidence.

I did not have the opportunity to participate in the development of 704 FORTRAN, as the project was halfway along when I joined IBM in 1955 December. However I did see the development, being on another project in the same room. During this time John Backus was appointed Manager of Programming Research for IBM, under the grand repository of computer wisdom -- John McPherson.

Only a month or so after the introduction of FORTRAN, Dr. Alan Perlis completed an algebraic compiler for the 650, called IT (Internal Translator). IT was originally conceived for the Datatron 205, but that processor suffered a delay when Dr. Perlis left Purdue for Carnegie Tech, not becoming operational until the summer of 1957. Although the names of the variables were very limited in form, the method of translation for such a lesser machine was most ingenious, and I was by this time enthusiastic enough about the possibility of machine-independent languages to ask Dr. Perlis for permission to use his system imbedded in a FORTRAN system. He agreed, and a preprocessor was constructed to translate from slightly limited FORTRAN statements to IT statements (which produced SOAP statements, which were <u>then</u> compiled). This project was led by Dave Hemmes,

who qualifies as a real documentor, with Florence Pessin, Otto Alexander, and Leroy May. Mrs. Pessin, a double-crostic addict, named the system FORTRANSIT, having a three-way meaning for 1) FOR TRANSITion, 2) FORTRAN, Soap, and IT, and 3) it FORTRANS IT. Although later replaced by a real FORTRAN processor for the 650, this jury-rig device did add visibly to the proof for machine-independence, particularly since one machine was binary, the other decimal.

At the time 704 FORTRAN was to be put into the field, IBM formed the Applied Programming Department under Jack Ahlin. As Manager of Programming Systems, the actual field operation of FORTRAN became my worry, while the Backus group continued in Research to develop what was to become FORTRAN II, which is pretty much what this present book describes. And worries I had, for at around 25,000 instructions this was a very complex program for that period. The complexity is best indicated by the departure of Sheldon Best for a position at M.I.T. before the processor was quite completed. It took Drs. Sayre and Goldberg three months of day and night work to figure out just what he had done in his section 5.

The 704 tape units presented major difficulties. It seemed impossible to run FORTRAN on more than the test machine. Finally a squad was sent to the West Coast to work with the customer engineers. When FORTRAN finally would run on a different machine, the C.E.s took careful note of just what they did and prepared a special writeup on how to tune the computer so it would run FORTRAN. This marked the end of an era for engineering diagnostic programs, for they said the machine was OK when FORTRAN said it wasn't. Besides, who cared if a component was faulty if the programming system didn't use it? Armed with this argument, Hemmes and I marched on Poughkeepsie Product Test, where Mr. G. A. Hemmer was quite willing to use FORTRAN as a major component of the factory test and acceptance program.

It then became time to produce a FORTRAN for the new 709. In the meantime we had learned many things about the operational requirements for such a processor. One installation, for example, figured that they translated and tested FORTRAN programs on the average of 50 times before they became correct, operational, and complete. Compiling time was outstripping production time and 80 percent of that was for optimizing index register usage for incorrect programs. Obviously the capability of switching off the optimization was necessary.

Since the new version of FORTRAN (FORTRAN II) became available for the 704 in 1958 June, this was the version built for the 709, becoming operational in 1959 June. The major difference

from the original language was the ability to compile independent subroutines written in either FORTRAN or the assembly language, and have these utilized by main FORTRAN programs compiled at a different time. This would not only save machine time, it was logically far sounder. I have always thought it a development equivalent in importance to the original FORTRAN.

The follow-on to FORTRAN II was to be XTRAN, mainly to remove certain restrictions in the language rather than introduce radical concepts. However, XTRAN was more or less swallowed up by ALGOL 58 in the first cooperative international effort on programming languages. My frustrations on losing the initiative on XTRAN were compensated by being able to edit ALGOL 58 into a reasonably clean form.

The first non-IBM FORTRAN processor was done for the Philco 2000, becoming operational in 1960 April; however, they called it ALTAC. The first processors to actually use the name FORTRAN were those for the UNIVAC Solid-State 80 and the CDC 1604 in 1961. As of 1965 between 60 and 100 FORTRAN processors have been implemented for very many machines. The FORTRAN Committee of SHARE has been a stabilizing influence, at least throughout the IBM line. The assembly language now commonly associated with FORTRAN (called FAP) was introduced through this committee in 1960 September. It was the product of the Western Data Processing Center at UCLA.

The latest production stage in FORTRAN is the FORTRAN IV language. This is a further loosening of the restrictions and an addition of new features along the lines of XTRAN and ALGOL. The SHARE FORTRAN Committee accepted this for the 7094, realizing that it was in several ways incompatible with FORTRAN II. However, the differences were mechanically convertible. This time there was no pussy-footing; UNIVAC set out to build FORTRAN IV for the 1107 and actually managed to get it into operation before that for the 7094. At present there are some 10 to 12 new changes proposed by the SHARE Committee for an improved FORTRAN IV. However, in the face of NPL (new programming language) for the IBM 360 these may not be implemented.

The worldwide effect and sway of FORTRAN is amazing. In his short history Heising said that over 228,000 manuals have been distributed. It is my hope that Mr. Lecht's book will be recognized as one of the most important of these contributions. It certainly fills a void to which I have been acutely sensitive.

ROBERT BEMER

PREFACE

The contents of this book have found their way to print through many years of FORTRAN programming by its author and a group of computer programmers who, if they did not take part in FORTRAN's original design, did spend significant portions of their lives "in residence" with it. Thus, as stated within its INTRODUCTION, this book "reflects more of what the language is than what it may have been intended to be."

The wide variety of computing machines for which at least one version of the FORTRAN compiler has been written prevents the material within this book from being all and always applicable on every specific computer. However, this lack of total applicability should cause significantly more concern within international committees on programming standards within the computer industry, than to the reader concerned with utilizing FORTRAN as a day to day working tool.

It is recognized that *total* FORTRAN compatibility between computing machines is not a reality, yet there is sufficient compatibility so that it may be stated that for equivalent classes of computers, the majority of the information presented within this book is common.

It may be further stated that with minor or non-standard FORTRAN statement exceptions, there is no "living" FORTRAN the statement repertory of which is not, at least, imbedded within this book's contents. The minor exceptions include the author's purposeful intent to bid adieu to the FREQUENCY statement which was mostly ignored in the past and is no longer implemented. Other exceptions include statements which are usually considered special features of *some* FORTRAN compilers.

The compilation of the various levels of symbols, combinations of symbols, sets of combinations of symbols, etc. which exist on the pages of this book and the process of arranging these "marks" in such a way as to have meaning to a wide variety of FORTRAN users was far beyond the single-handed capabilities of the author. It is then important that others who helped support the author in his attempt to overcome the idolization of a single symbol or arrangement of symbols to the exclusion of all others be mentioned.

Maximum recognition must be given to Mrs. Winnie Schare (of ACT) for collection of much of the material and editorial comment. The author is equally indebted to Mrs. Elizabeth Holberton, whose encouragement and editorial consul proved invaluable, as well as to her associate Mrs. Nora Taylor (both of whom are top technical experts at the U. S. Navy, David Taylor Model Basin).

Miss Joan Gildea and Miss Carol Ott sharing technical typing duties were both tireless and first-rate.

Finally, the encouragement of Mr. Robert Bemer, while he was Director of Systems Programming at UNIVAC proved the "straw that broke the camel's back" and perhaps the *raison d'etre* for it all.

CHARLES P. LECHT

AUTHOR'S NOTE

Guide to the Efficient Use of This Book

This book contains a complete description of the FORTRAN II and IV languages, where 'complete' is defined within the fourth paragraph of the INTRODUCTION

Due to the manner in which the descriptive material has been organized, the differences between FORTRAN II and IV are easily seen.

The general organization of the book is made clear by a review of its TABLE OF CONTENTS except for the following protocol.

> The kernel of the book is FORTRAN II. FORTRAN IV information is provided and labeled as such only when it
>
> i. extends the scope of a particular FORTRAN II concept,
>
> ii. limits the scope of a particular FORTRAN II concept, or
>
> iii. changes the concept, where limitations or extensions are academic.
>
> Thus, the FORTRAN II material may be considered FORTRAN IV material except when modified as above. Modifications to instructions immediately follow the description of each instruction. Other FORTRAN IV material is clearly labeled as such.

When not followed by a II or IV, the word FORTRAN means FORTRAN II and FORTRAN IV.

In presenting the material contained within this book, strict formatting rules were followed. These rules were developed with two purposes in mind.

The first and foremost was to effect a presentation of the material with great precision and completeness of information content.

The second, and not unimportant reason, was to overcome the usual equivocation to which past authors have been

party in presenting a well defined language description in a not so well defined form. Thus, this book does not suffer the problem of being part learner's text and part reference guide, but rather it is specifically intended to be the latter.

The motivation for presenting the material in the form contained within this book is then suggested above. The FORTRAN language is trivial to learn by the average programmer. However, in actual use, he must constantly confirm his knowledge through reference to the language description. This comes about through both his desire to perform properly and the high cost of making a mistake in the writing of a computer program.

The usual text which has been used as a reference guide in writing computer programs has suffered from containing much irrelevant and too much introductory material. Because of the appearance of this material the meaningful descriptions of the language statements themselves have not been presented either in sufficient detail or clarity That problem has been overcome in this book.

Users of this book will find it valuable to read its INTRODUCTION at least once in order to gain a feel for the declarative writing style which has been used throughout. It would also be of value to study the manner in which the TABLE OF CONTENTS has been organized.

The contextual use of the material is self-evident except, perhaps, for the following.

Definitions of all words in the book which are marked by an asterisk (*) appear within the GLOSSARY. In the case of hyphenated words which are asterisked, their definition will appear in the GLOSSARY alphabetized by the first word. The GLOSSARY contains words which are themselves asterisked and which, in turn, follow the rules described above.

CONTENTS

THE PROGRAMMER'S FORTRAN II AND IV

I. *Introduction*

FORTRAN

INTRODUCTION:

This book is intended for programmers.

It is intended to be self-contained.

It is a detailed description of the FORTRAN languages which may be appropriately termed, FORTRAN II and FORTRAN IV. The word 'termed' is used because the details of the languages presented in this book have been derived through the study and use of various versions of the languages as implemented on a variety of widely used computing machines.

This description of FORTRAN reflects more of what the language is than what it may have been intended to be. In that regard, users of this book will not suffer from language usage which reflects unimplemented intentions nor seemingly logical deductions which are inadmissible.

This book is to be used for reference purposes. It is not a "self-teaching" device.

A. Definition of FORTRAN:

FORTRAN is a set of symbols* and a set of rules.

FORTRAN is a language.

FORTRAN is intended for (although not limited to) use in scientific problems involving a significant amount of numerical calculation. In this regard, it is called a "scientific language.

FORTRAN is a language in which computer programs may be written.

There is a compiler associated with the FORTRAN II language and with the FORTRAN IV language. It is called the FORTRAN II or FORTRAN IV COMPILER respectively. The FORTRAN COMPILER translates computer programs written in the FORTRAN language into the basic language (machine code) of a computer.

B. *The symbols* of FORTRAN:*

Symbols Used to Write the Language

Alphabetic Characters:*	ABCDEFGHIJKLM NOPQRSTUVWXYZ
Digit Characters:*	0123456789
Special Characters:	/ *slash* + *plus* - *minus* * *asterisk* . *point* , *comma* = *equal* *blank* (*left parenthesis*) *right parenthesis*

(NOTE: Throughout this book the character 'blank' will be specified by a 'b')

By following certain rules, the symbols of FORTRAN may be put together to form statements.*

Symbols Which a FORTRAN Compiler can Recognize

This includes all the symbols listed above, plus additional special characters which may vary for each computer.

C. *Types of FORTRAN Statements*

Control:

used to alter the normal sequential execution of statements;
used to make logical decisions.

Arithmetic:

used to perform numerical calculations.

Input/Output:

used to cause transfer of data between I/O units and internal storage locations.

C. *Types of FORTRAN Statements (contd.)*

Subprogram:*

used to permit the writing of subroutines or of logical sections of instructions as separate programs to be joined with a main-program* when the problem is run.*

Specification:

used to give information to the FORTRAN compiler.

D. *Form of a FORTRAN Program*

The FORTRAN language has many statement usage rules which are either dependent upon or independent of a problem for which the language is being written.

The rules which are dependent upon specific problems for which the language can be used are not the subject matter of this book.

The rules which are independent of any specific problem for which the language can be used but must be followed for all problems where usage is attempted can be broken down into two categories:

Rules for forming FORTRAN statements

Rules of relationship between FORTRAN statements

The rules for forming FORTRAN statements comprise the main text of this book; Sections II and III.

The rules of relationship between FORTRAN statements are presented below.

D. *Form of a FORTRAN Program (contd.)*

These statement relationships arise through the design and development of the FORTRAN language itself as a problem solving tool.

There are three such relationships some or all of which govern all FORTRAN statements

ORDER	-	*a strict precedence relationship which demands that if a certain statement has been used, an associated statement must preceed it either physically or logically.*
REFERENCE	-	*a dependence relationship which demands that if a certain statement is used, another one exists.*
POSITION	-	*a physical relationship which demands that if a certain statement is used, the location of another statement or group of statements is determined.*

There are no other such relationships.

Thus each program written is a sequential list of statements, where the statements are related to one another as defined above.

An example of a program written in the FORTRAN language is included here to present a visual and conceptual model of the use of the language. Thus, this model establishes a pattern of statement use which suggests the form of all FORTRAN programs.

D. Form of a FORTRAN Program (contd.)

*Statement Number**		*FORTRAN Statement*
C		THIS IS A SAMPLE PROGRAM WHICH
C		WILL DETERMINE THE REAL ROOTS
C		OF A QUADRATIC
4		READ 1,A,B,C,TITLE,DATE
		CALL HEAD(TITLE,DATE)
		DISCR=B**2-4.*A*C
8		IF(DISCR)3,2,2
2		AROOT=(-B+SQRTF(DISCR))/(2.*A)
		BROOT=(-B-SQRTF(DISCR))/(2.*A)
10		PRINT5,A,B,C,AROOT,BROOT
		GO TO 12
1		FORMAT(3E14.5,A6,A6)
5		FORMAT(3E14.5,2E18.6)
3		PRINT7,A,B,C
7		FORMAT(3E14.5,5X,15HIMAGINARY ROOTS)
12		STOP
		END

In the above program,

control statements are - *8, 12, and one statement after 10 (i.e., 10+1)*

arithmetic statements are - *2, 2+1, 4+2*

input/output statements are - *4, 10, 3*

specification statements are - *1, 5, 7, and 12+1*

subprogram statement is - *4+1*

comments are - *the first three lines of the program.*

E. *Writing a FORTRAN Program*

A program is composed of a series of statements and comments.*

A statement may be translated by the compiler into many computer operations or into no operations at all.

Those statements which result in specific computer operations are termed executable and the sequence in which they are given is important. Control, Arithmetic, Input/Output, and Subprogram statements are executable. Specification statements are not executable and in general may be placed anywhere in the source-program.*

Program execution begins with the first executable statement written and continues through each executable statement in order of appearance unless a control type statement directs the program away from the next sequential statement.

The statements of a program are written on FORTRAN coding forms, each line corresponding to a memory record.

Positions 1-72 may be used.

Positions 73-80 are not interpreted by FORTRAN but may be used by the programmer.

Positions 1-5 are used for statement-numbers. A statement only needs a number if reference is made to it. Statement numbers need not be in any order. The statement number is purely a device for referencing statements, and has no other significance. Integers* from 1 to 32767 may be used for statement numbers.*

The statement begins in position 7.

Position 6 is normally blank.

E. Writing a FORTRAN Program (contd)

Statements may be continued on up to 9 additional records in FORTRAN II and 19 additional records in FORTRAN IV. In either case, these are called continuation records. Position 6 of all continuation records must contain a character other than 0 or b.

Explanatory comments within a program are designated by a 'C' in position 1 of each comment record. These comments play no part in the execution of the program.

With the exception of one FORTRAN II statement and two FORTRAN IV statements, blank positions are ignored and may be used freely to improve the readability of a FORTRAN program.

The exceptions indicated above are the FORMAT *statement in FORTRAN and the* DATA *statement in FORTRAN IV.*

II. *FORTRAN Statements*

The description of each statement in the FORTRAN language is presented in the following sections. Each statement starts on a new page with the format of its descriptive material given as shown below:

Instruction Name (for left-hand page)

Instruction Name (for right-hand page)

USE: (a brief statement of the purpose of the instruction)

(Form of the Instruction)

(Definition of Symbols used in the Form line)

RULES:

(A list of rules governing the correct usage of the instruction; includes restrictions, suggestions, etc.)

EXAMPLES:

(A list of examples illustrating the use of the instructions; includes examples of incorrect usage.)

As many pages as needed describe an instruction.

If there are any differences between the FORTRAN IV use of an instruction and its FORTRAN II counter-part, an extra page(s) labeled FORTRAN IV is included which only presents those attributes of the instruction which differ.

USE: *To perform a numerical calculation.*

a=e

a: simple variable; single value of a subscripted-variable*.*
e: an expression.*

RULES:

1. *The equal sign (=) means "is replaced by" (i.e., 'a' is replaced by 'e'.)*

2. *The value of the expression on the right of the equal sign is computed and this value replaces the previous value of 'a'.*

3. *The quantity stored will be an integer* if 'a' is defined as an integer variable.*

4. *The quantity stored will be floating-point* if 'a' is defined as a floating point variable.*

5. *Calculations will be performed in the integer mode* if the expression on the right side of the equal sign contains all integer quantities. (NOTE: Exception: In an integer expression a floating point quantity may appear as the argument* of a function.)*

6. *Calculations will be performed in floating point mode if the expression contains all floating point quantities. (NOTE: Exception: In a floating point expression an integer quantity may appear as an exponent, a subscript*, and as an argument of a function.)*

EXAMPLES:

1.	`X=Y*Z`	*product of Y and Z replaces variable X.*
2.	`A=I/J`	*calculation is in integer mode ;if I=3 and J=2, result is 1; (fractional part is truncated) stored in A as a floating 1(1.).*

EXAMPLES: (contd.)

3.	`I=A*B`	*calculation is in floating point mode; if A=1.2 and B=2.4, result is 2.88; stored in I as the integer 2. (factional part truncated).*
4.	`J=J+1`	*illustrates meaning of = sign. J is incremented by 1 and the new value replaces the old.*
5.	`A=I`	*converts I to a floating point number and stores it in A.*
6.	`K=B`	*truncates B to an integer and stores it in K.*
7.	`X=T+(U*V)**2-EXPF(R-C(3))` `/(3.1416*P(J)*T**5)`	*shows the ability to make many calculations using a single statement.*

RULES:

1. *The statement is also used to perform logical operations.*
2. *The words "floating point" as used in FORTRAN II have been replaced by the word "real*" in FORTRAN IV.*
3. *The quantity stored will be integer, real, double-precision*, complex* or logical* (.TRUE. or .FALSE.) depending on the type of variable 'a'.*
4. *If 'a' is a logical variable*, then 'e' must be a logical-expression*.*
5. *If 'a' is a complex variable, 'e' must be a complex expression.*
6. *If 'a' is a real variable, 'e' may be a real, integer, or double precision expression.*
7. *If 'a' is an integer variable, 'e' may be a real, integer, or double precision expression.*
8. *If 'a' is a double precision variable, 'e' may be a real, integer or double precision expression.*

EXAMPLES:

1.	`X = Y*Z`	*If X and Y are real and Z is double precision, Y*Z uses double precision arithmetic. The most significant part of the result is stored in X.*
2.	`C = D.AND..NOT.E`	*C, D and E must be logical quantities. C will be .TRUE. only when D is .TRUE. and E is .FALSE.. In all other cases C will be .FALSE..*
3.	`W = V.LE.17.2`	*W is a logical variable. V is either real or double precision. W will be .TRUE. if V ≤ 17.2; otherwise W is .FALSE..*

```
4.   COMPLEX P,Q(10),R
     P = Q(J)-(4.2,51.7)-CSIN(R)
```

Integer J appears in a complex expression as a subscript.

arithmetic statement function

USE: *To define a function* in a single arithmetic statement*; to make a subroutine* out of a series of calculations which recur in a single program.*

$$name\mathrm{F}(a_1,a_2,.....,a_n)=e$$

name: *a function name* followed by a terminal F.*

$a_1,a_2,..,a_n$: *non-subscripted-variables* which are dummy-arguments*.*

e: *an arithmetic expression* not involving subscripted variables.*

RULES:

1. *Arithmetic statement functions must precede the first executable statement in a source-program*.*
2. *The function is compiled as a closed-subroutine*.*
3. *The function may be used any number of times.*
4. *'e' may contain other functions that are available to the program. (i.e., built-in, library, or previously defined arithmetic statement functions.)*
5. *'e' may contain variables not stated as arguments. These are treated as parameters and at the time the function is called their current values will be used.*

EXAMPLES:

1. `SOMEF(X,Y) = X**2-2.*Y` *defines a simple calculation.*
2. `ANYF(D,E,F) = D-E*SQRTF(F)` *SQRT is a library function.*

EXAMPLES: (contd.)

```
3.  ONEF(A)          = 4.*A-SINF(A**2)
                                          definition of ONEF.
                                          (dashes represent
                                           other coding.)
          T          = ONEF(S-U**3)/      use of ONEF in the
                       ONEF(R-5.)         program

4.  FIRSTF(P(I),Q) = Q*P(I)/(Q-1.)        illegal! P(I), a sub-
                                          scripted-variable*, is
                                          not valid as a dummy-
                                          argument*.
          D          = S-FIRSTF(C(4),R)

4.  (alt)
    FIRSTF(P,Q)      = Q*P/(Q-1.)         a dummy argument is
                                          always a simple vari-
                                          able* whereas an
                                          actual argument may
          D          = S-FIRSTF(C         be a constant*, a
                       (4),R)             simple variable, a
                                          subscripted variable,
                                          or an expression*.

5.  BIGF(G,H)        = G/2.*R-H**3

        R = A*B+C

          T = BIGF(X,Y)-S+5.3             current value of R
                                          will be used.
```

arithmetic statement function

FORTRAN IV

The form of the statement is as follows:

$$name(a_1,a_2,\ldots,a_n) = e$$

There is no terminal F *following the function name.*

RULES:

1. *Type statements (if any) for the dummy arguments must precede the arithmetic statement function.*

USE: *To give a statement-number* value to an integer variable*; to set up the path which a subsequent assigned* GO TO *will take.*

ASSIGN *n* TO *j*

n: statement number
j: integer variable

RULES:

1. *An* ASSIGN *statement* must be executed in the object-program* prior to the assigned* GO TO *statement to which it refers.*

2. *An assigned-variable* (j) may be used as an ordinary variable only if it is redefined.*

EXAMPLES:

1. ASSIGN 33 TO N — *for the next assigned* GO TO, *N will refer to statement number 33.*

2. ASSIGN 6 TO NKLMN

BACKSPACE

USE: *To cause a tape unit to backspace one tape record.*

BACKSPACE *i*

i: unsigned integer constant or an integer variable* which is a logical-tape-unit* designation.*

RULES:

1. *Any number of tape records may be backspaced by giving successive* BACKSPACE *statements.*

2. *If the* BACKSPACE *statement is given for tapes written with the* WRITE TAPE *statement, one logical-tape-record* is backspaced.*

EXAMPLES:

1.
```
BACKSPACE 5
BACKSPACE 5
BACKSPACE 5
```
will position logical tape unit 5 three records back

USE: *To transfer control to a subroutine subprogram*; to give arguments* to the subroutine subprogram.*

CALL *name* $(a_1,a_2,\ldots,a_n)$

*name: a subprogram name**
$a_1,a_2,..,a_n$: *actual arguments*

RULES:

1. *An argument in the* CALL *statement may be a. constant*, b. variable*, c. subscripted-variable*, d. name of an array* without subscripts, e. arithmetic expression*, f. Hollerith-string*.*

2. *The arguments in the* CALL *statement must agree in number, order, and mode* with the arguments in the* SUBROUTINE *statement.*

3. *There need not be any arguments in the* CALL *statement if all the pertinent data are assigned locations in common* and thereby transmitted implicitly.*

EXAMPLES:

1. `CALL MATMPY (X,5,10,Y,7,Z)` *(arguments illustrating a,b)*

2. `CALL QDRTIC (P*9.732,Q/4.536,R-S**2.0,X(1),Y,2)` *(a,b,c,e)*

3. `CALL SAM (1,2.14,L,K(2),Z,Y(1,2,3),ARRAY,A*B,3HABC)` *(illustrates all)*

4. `CALL CALC` *(no arguments)*

CALL

FORTRAN IV

RULES:

1. *An argument may also be a logical-expression*, or the name of a function, or a subroutine subprogram.*

EXAMPLES:

1. CALL SUBR (SIN,A.GT.B)

USE: *To cause designated data items to be allocated storage in a fixed portion of storage called common*; to permit these items to be shared between programs.*

COMMON *v, v, ..., v*

v: each 'v' is a variable or non-subscripted array* name*

RULES:

1. *The items are allocated storage in the sequence in which they appear in the* COMMON *statement*.*

2. *There may be any number of* COMMON *statements in a program. (Each starts assignment where the previous one left off.)*

3. *Ordering in common may be altered by an* EQUIVALENCE *statement. (See* EQUIVALENCE *example 3.)*

4. *A* COMMON *statement may be placed anywhere in the source-program* except as the first statement in the range of a* DO.

5. *If* COMMON *statements appear in two or more programs (i.e., in a main-program* and a subprogram*, or in several subprograms) which are to be run jointly, the exact common statements must be used to assure proper correspondence between variables in each program. These programs will share the designated area of common.*

6. *This permits the transfer of arguments* between a main program and subprogram without having them appear explicitly in the argument list following the subprogram name*. All that is necessary is to have the dummy-argument* and the actual argument occupy corresponding positions in common.*

7. *All the arguments of a subroutine subprogram may be transmitted through common. However, there must be at least one explicit argument for a function subprogram.*

RULES: (contd.)

8. *If the name of an array appears in a* COMMON *statement, it must also appear in a* DIMENSION *statement in the same program.*

EXAMPLES:

1.
```
COMMON X, Y, Z

DIMENSION X(1), Y(1,2), Z(1,2,3)
```

In common storage

X(1)
Y(1,1)
Y(1,2)
Z(1,1,1)
Z(1,2,1)
Z(1,1,2)
Z(1,2,2)
Z(1,1,3)
Z(1,2,3)

2.
```
COMMON A,B,C
COMMON J,K,L
COMMON P,Q
```

A
B
C
J
K
L
P
Q

3.

Main Program

```
COMMON A,B,C

DIMENSION B(3)
```

Subprogram

```
COMMON S,T,U,V,W

DIMENSION V(2)
```

In common storage

A, S
B(1), T
B(2), U
B(3), V(1)
C, V(2)
W

A more general form of the COMMON *statement is permitted.*

COMMON/a_1/v,v,...,v/a_2/v,v,...,v/.../a_n/v,v,... v

v: *each 'v' is a unique variable* which may be subscripted* with dimension information*

$a_1, a_2, \ldots, a_n$: *variables which are common block names*, or blanks*

RULES:

1. *FORTRAN II Rule 3 does not apply.* EQUIVALENCE *will not reorder common. However, the length of a common block may be increased by an* EQUIVALENCE *statement (see* EQUIVALENCE*).*

2. *FORTRAN II Rule 8 does not apply. An array may appear in a* COMMON *statement with its dimension information. Therefore it must not appear in a* DIMENSION *statement in the same program or in a type statement with its dimension information.*

3. *Elements may be placed in separate areas of common called blocks. These blocks are given names which appear enclosed in slashes (//) in a* COMMON *statement.*

4. *A common block name may appear in several* COMMON *statements of a program. Assignment of variables will be in the order of the* COMMON *statements.*

5. *Ordinary common or blank common is indicated by two consecutive slashes or by omitting the block name. See example 1.*

6. *Blocks in different programs which have the same names will share storage space. All blocks of the same name must have the same length.*

7. *If a double word variable is placed in common, its high order part must be located an even number of storage locations away from the first common element.*

COMMON

FORTRAN IV

EXAMPLES:

1. COMMON /ABC/BOB,J, R(3),W(2)//X,Y,Z(2,2)/M/P,Q,S

In common storage

M	*blank*	*ABC*
P	*X*	*BOB*
Q	*Y*	*J*
S	*Z(1,1)*	*R(1)*
	Z(2,1)	*R(2)*
	Z(1,2)	*R(3)*
	Z(2,2)	*W(1)*
		W(2)

2. COMMON F,G,H(2)/CAT/A,C,D,I//BOY,DOG
   COMMON /CAT/T,U,V//M,N

In common storage

blank	*CAT*
F	*A*
G	*C*
H(1)	*D*
H(2)	*I*
BOY	*T*
DOG	*U*
M	*V*
N	

3. DOUBLE PRECISION ANS
   COMMON A,B,C,D,ANS,E

In common storage

A
B
C
D
ANS (high order)
ANS (low order)
E

USE: *To be the last statement* in the range of a* DO, *when the last statement would otherwise have been a* GO TO *or an* IF; *to furnish a reference point which may be given a statement-number*.*

```
n    CONTINUE
```

*n: statement-number**

RULES:

1. *The statement number n of the* CONTINUE *statement provides a transfer point for an* IF *or a* GO TO *that is intended to begin another repetition of the* DO *range.*

2. CONTINUE *is a dummy statement and creates no object-program* instructions.*

3. *When not used as part of a* DO, *any transfer to this statement causes the statement following to be executed next.*

EXAMPLES:

1.
```
10 DO 12  I=1,10
   IF (ARG-VALUE(I))  12,13,12
12 CONTINUE
```

2.
```
   DO 98  J=1,15,2
   IF (A(J)-TEST)  96,97,96
97 K=K+1
   GO TO 98
96 L=L+1
98 CONTINUE
```

3.
```
     -
     -
     -
   GO TO 25
     -
     -
     -
   IF (A*B-C)  1,25,1
     -
     -
     -
25 CONTINUE
   C=D*X(4)
```

If a statement has been deleted but it is not desirable to change all statements which referenced it,* CONTINUE *may be used.*

(dashes represent other coding)

DIMENSION

USE: *To tell the FORTRAN compiler how much storage to set aside for arrays*; to specify a maximum size for each array used in the program.*

DIMENSION *v,v,v,.....,v*

v: each 'v' is a subscripted-variable with 1, 2, or 3 unsigned integer constant* subscripts**

RULES:

1. *Each array must appear in a* DIMENSION *statement.*

2. *The* DIMENSION *statement must precede the first appearance of the subscripted variable in the source-program* and may not be the first statement in the range of a* DO.

3. *The subscripts represent the maximum number of elements possible in the array. No reference to the array should be made in which the value of a subscript exceeds this maximum.*

4. *Arrays may not be more than 3-dimensional.*

5. *Any number of arrays separated by commas may be listed in a single* DIMENSION *statement.*

6. *There may be any number of* DIMENSION *statements in a program.*

7. *'v' may not be the name of the program itself, or of any program which it calls.*

8. *Each 'v' in all the* DIMENSION *statements of a single program must be unique.*

9. *Arrays are stored in descending storage locations with the first subscript varying most rapidly, and the last varying least rapidly.*

EXAMPLES:

1.	`DIMENSION A(50)`	*sets aside 50 positions of storage for a one-dimensional floating-point* array named A.*
2.	`DIMENSION X(5),Y(2,10),I(30)`	*several arrays in one* DIMENSION *statement.*
3.	`DIMENSION C(12),D(10),` `SQRT(40),C(12)`	*illegal! See RULES 7. and 8. SQRT is a library routine.*

DIMENSION

FORTRAN IV

RULES:

1. *The v's may have variable subscripts if the* DIMENSION *statement appears in a subprogram*.*

2. *If a subprogram array has variable dimensions the array name and its dimensions must be arguments of the subprogram.*

3. *The actual dimensions must appear in the calling program* DIMENSION *statement and these are the dimensions which must be passed to the subprogram.*

4. *If an array name appears in a* COMMON *or a type statement with its dimension information, the array name must not appear in a* DIMENSION *statement.*

5. *FORTRAN II Rule 9 does not apply. Arrays are stored in ascending storage locations with the first subscript varying most rapidly and the last varying least rapidly.*

EXAMPLES:

1.

Main Program

```
DIMENSION X(5,10),Y(7,7)
    —
    —   (dashes represent
    —    other coding)
    —
CALL MAT(X,5,10)
    —
    —
    —
CALL MAT(Y,7,7)
```

Subprogram

```
SUBROUTINE MAT (A,N,M)
    —
    —
    —
DIMENSION A(N,M)
    —
    —
    —
```

USE: *To execute repeatedly a series of statements*; to cause looping* having initializing, incrementing and testing of the loop all done by a single statement.*

DO n $i = m_1, m_2, m_3$

n: *a statement-number**
i: *a non-subscripted integer variable**
m_1, m_2, m_3: *unsigned integer constants* or non-subscripted integer variables*

RULES:

1. *The* DO *statement is a command to execute repeatedly the statements which follow, up to and including statement number n. This sequence of statements is called the range of the* DO.

2. *i is called the index of the* DO*; m_1 is the initial value for i, m_2 is the maximum value for i, and m_3 is the increment for i.*

3. *i is set equal to m_1 for the first execution of the statements in the range of the* DO. *For each succeeding time through the range, i is increased by m_3.*

4. *After the statements in the range of the* DO *have been executed with the index of the* DO*, (i), equal to the highest number which does not exceed m_2, control passes to the statement following the last statement in the range of the* DO*, i.e., the statement after statement number n. The* DO *is then said to be 'satisfied'.*

5. *m_2 must be greater than or equal to m_1.*

6. *If m_3 is not specified, it is assumed to be one.*

7. *The range of the* DO *can consist of any number of statements*, including only one statement.*

8. *The index of the* DO *need not be used in the range of the* DO*, but it is usually used in computations or as a subscript*.*

RULES: (contd.)

9. *If* DOs *are within* DOs *(this is called a nest of* DOs*), the range of the inner* DO *must be within the range of the outer* DO. *(Note: More than one* DO *of the nest may end at the same statement.)*

10. *Transfer of control from within a* DO *to outside its range is permitted.*

11. *Transfer of control from outside the range of a* DO *to within its range is not permitted, except when returning from a temporary exit (see RULE 10).*
NOTE: Statements executed during this temporary exit are considered to be logically within the DO *range and therefore RULE 12. applies.*

12. *No statement is permitted within the range of a* DO *which re-defines the index i, the initial value* m_1, *the terminal value* m_2, *or the incremental value* m_3.

13. *The first or last statement in the range of a* DO *must not be a non-executable FORTRAN statement; i.e.,* DIMENSION, FORMAT, COMMON, *etc.*

14. *The last statement in the range of a* DO, *(statement number n), must not be another* DO, *a transfer (i.e.,* IF *or* GO TO*), or a non-executable statement. See* CONTINUE.

15. *The value of i, the index of the* DO, *is available upon leaving the range of a* DO *only if executing a* CALL *or transferring out by* IF *or a* GO TO*; and not when passing to the statement after statement-number* n (i.e., when the* DO *has been satisfied).*

16. *If an assigned* GO TO *statement is in the range of a* DO, *all the statements to which it may transfer must either be in the range or all must be outside the range.*

EXAMPLES:

1.
```
    DO 5  I=1,3
  5 A(I)=A(I)+B(I)
```
index used as subscript; m_3=1 is assumed; range of* DO *is only one statement*

2.
```
    DO 89  J=2,12,3
```
this DO *is executed 4 times with j=2,5,8,11*

3.
```
    DO 35  K=70,100,10
    DO 31  I=1,5
 31 S(I)=T(I)-U(I)
 35 A   =K*J+L
```
a nest of DOs*; index used as a subscript and in computation*

4.
```
    DO 130  L=1,100,3
    IF(X(L)-50.)  77,130,130
130 CONTINUE
```
possible transfer out of range of DO*; use of* CONTINUE *since* IF *cannot be the last statement in range of* DO

5.
```
    N=1
     -
     -
     -
     -
     -
    DO 76  L=1,N,1
 76 J(L)=J(L)*3
```
(dashes represent other coding)

this DO *is executed <u>once</u>*

6.
```
    GO TO 111
    DO 11  I=1,500,50
111 A=B*C
 11 Z(I)=37.0
```
illegal! see RULE 11.

7.
```
    DO 26  M=3,9,3
    DIMENSION T(12)
    T(M)=A(M)/3.28
 26 CONTINUE
```
illegal! see RULE 13.

8.
```
    DO 17  L=1,15,2
    A(L)=B(L)-C(L)
    DO 17  K=3,10,2
 17 J(K)=K*L
```
nest of DOs *ending at statement 17*
statements repeat from the inner DO *until it is finished. Then they repeat from the outer* DO*, with inner* DO *index, K, starting again at its initial value*

9.
```
    DO 6  I=1,3
    IF(C(I))  6,6,8
  8 DO 6  J=I,6
    A(I,J)=B(J)*C(I)
  6 CONTINUE
```
the CONTINUE *is needed to end the nest of* DOs *in case the* IF *condition requires that the inner* DO *be bypassed*
index of inner DO *uses current value of outer* DO *index for starting point*

DO

FORTRAN IV

RULES:

1. *FORTRAN II Rule 4 has one exception. The range of a* DO *may be ended with a logical* IF *statement. See* IF, *logical for an explanation of what occurs in this case.*

USE: *To provide a shorthand notation to be used in the list* of an input/output operation for manipulation of data arrays.*

$$(((v,v,\ldots,v,i=l_1,l_2,l_3),v,v,\ldots,v,$$

$$j=m_1,m_2,m_3),v,v,\ldots,v,k=n_1,n_2,n_3)$$

v: *each 'v' is a subscripted-variable* with 1, 2, or 3 subscripts*, or a non-subscripted variable*

i, j, k: *non-subscripted integer variables**

l_1, l_2, l_3,

m_1, m_2, m_3,

n_1, n_2, n_3: *unsigned integer constants* or non-subscripted integer variables*

RULES:

1. *In the format shown above, a left parenthesis represents the beginning of an implied* DO *statement. The corresponding right parenthesis represents the end of that implied* DO.

   Implied DO*'s may be nested. i, j, k are indexes of the implied* DO. l_1 *is the initial value of the index,* l_2 *the maximum,* l_3 *(if stated) the increment, as explained under* DO*; m and n are similar.*

2. *An implied* DO *can only be used in the list of an I/O statement.*

3. *An implied* DO *may appear anywhere in a list.*

4. *Any number of implied* DO*'s may appear in a single list.*

RULES: (contd.)

5. *The implied* DO *must be separated by commas from the other list elements.*

6. *There may be any number of 'v's in an implied* DO *including only one 'v'.*

7. *There need not be any 'v's at all in the two outer sets of parentheses.*

8. *If a 'v' is a non-subscripted variable, the subscripts i, j, and k will have no effect on its transmission, but it will be repeated according to the rules for the* DO.

9. *'i', 'j', and 'k' represent the variable subscripts* of the 'v's which have not been previously defined.*

10. *There may be one, two, or three sets of parentheses, depending on whether there are one, two, or three undefined variable subscripts.*

11. *If any* $'l_1'$, $'l_2'$, $'l_3'$, $'m_1'$, $'m_2'$, $'m_3'$, *or* $'n_1'$, $'n_2'$, $'n_3'$, *is variable, it must have*

    a) *been previously defined, or*

    b) *appeared in the current list prior to the implied* DO
    (Note: This is only valid for an input list.)

12. *Implied* DO*'s covering the entire array are assumed if an array name, unsubscripted, (for which a* DIMENSION *statement appeared previously) is given in an I/O list.*

RULES: *(contd.)*

13. *Execution is that of a nest of* DO*'s with the innermost parentheses representing the innermost* DO*; i.e., referring to the general form of the instruction, page 37.*

```
      DO 1  k=n1,n2,n3
      DO 2  j=m1,m2,m3
      DO 3  i=l1,l2,l3
      v
      v
      .            (variables in innermost
      .            parentheses)
      .
   3  v
      v
      v
      .            (variables in middle
      .            parentheses)
      .
   2  v
      v
      v
      .            (variables in outermost
      .            parentheses)
      .
   1  v
```

EXAMPLES:

		Order of transmission
1.	`READ 5, (A(I),I=1,10)`	*A(1), A(2), ..., A(10)*
2.	`PUNCH 1, ((B(I,J),I=1,5), J=1,7)`	*B(1,1),B(2,1),...,B(5,1), B(1,2),...,B(5,2),..., B(1,7),...,B(5,7)*
3.	`WRITE OUTPUT TAPE 3,7, ((C(I,J),D(I,J),J=1,5), I=1,4)`	*C(1,1),D(1,1),C(1,2), D(1,2),...,C(1,5),D(1,5), C(2,1),D(2,1),...,C(4,1), D(4,1),...,C(4,5),D(4,5)*
4.	`PRINT 3,A(3),D,I,((X(K,L), K=1,5),Y(L),L=1,6), Z(1),Z(2)`	*A(3),D,I,X(1,1),X(2,1), ...,X(5,1),Y(1),X(1,2), ...,X(5,2),Y(2),..., X(5,6),Y(6),Z(1),Z(2)*

EXAMPLES: (contd.)

5.
```
READ INPUT TAPE 1,4,(((T
   (I,J,K),I=1,6),U(J,K),
     J=3,7),V(K),K=1,4)
```
Order of transmission

*T(1,3,1),T(2,3,1),...,
T(6,3,1),U(3,1),T(1,4,1),
...,T(6,4,1),U(4,1),...,
T(6,7,1),U(7,1),V(1),
T(1,3,2),...,T(6,7,2),
U(7,2),V(2),...,T(1,3,4),
...,T(6,7,4),U(7,4),V(4)*

6.
```
READ 2,K,(A(I),B(I,K),
   I=1,10),((C(I,J),I=1,10,
     2),D(J,3),J=1,K)
```
*K,A(1),B(1,K),A(2),
B(2,K),...,A(10),B(10,K),
C(1,1),C(3,1),...,C(9,1),
D(1,3),C(1,2),C(3,2),...,
C(9,2),D(2,3),...,C(1,K),
...,C(9,K),D(K,3)*
Note: There are 2 implied DO*'s; value of K read in will be used for B and D arrays.*

7. *The list in Example 6 with explicit* DO *statements:*

```
      K
      DO 4  I = 1, 10
      A(I)
4     B(I,K)
      DO 6  J = 1, K
      DO 5  I = 1, 10, 2
5     C(I, J)
6     D(J, 3)
```

8.
```
DIMENSION A(10)
READ 5, A, X, Y
```
A(1), A(2), ..., A(10), X, Y
Illustration of Rule 12.

RULES:

1. *FORTRAN II Rule 2 does not apply.*
   An implied DO *may be used in the list of a* DATA *statement as well as in an I/O statement*

END

USE: *To specify the physical end of a source-program* during a FORTRAN compilation; to specify an action to be taken by the FORTRAN compiler at the end of compilation with regard to the setting of the individual Sense-Switches*.*

END (I_1,I_2,...,I_5)

I_1,I_2,...,I_5: 0, 1, 2

RULES:

1. *Sense Switch options are specified by the installation.*
2. *The* END *statement* must be physically the last statement of every program.*
3. END *creates no object-program* instructions.*

EXAMPLES:

1. END (2,2,2,2,2)
2. END (0,1,2,0,1)

END

FORTRAN IV

RULES:

1. *Sense Switch options do not apply. The form is simply*

```
END
```

END FILE

USE: *To write an end of file indication on a tape unit.*

END FILE *i*

i: unsigned integer constant or an integer variable* which is a logical-tape-unit* designation*

RULES:

1. *All output tapes should have end-of-file indications.*

EXAMPLES:

1. END FILE 4 — *writes end of file mark on logical tape 4*

USE: *To conserve storage by having several variables* use the same location when the logic of the program permits.*

EQUIVALENCE (*a,b,c,...*),(*d,e,f,...*)
,...,(....)

a,b,c,d,e,f: *variables* which may be followed by a single integer constant* in parentheses*

RULES:

1. *An* EQUIVALENCE *statement* may be placed anywhere in the source-program*, except as the first statement in the range of a* DO.

2. *This statement causes all of the variables in each parenthetical expression to be assigned the same storage location in the object-program*.*

3. *Any number of equivalences (i.e., sets of parentheses) may be given.*

4. *Locations in storage can be shared only among variables, not among constants.*

5. *Variables brought into common* through an* EQUIVALENCE *statement may reorder common as set up by the* COMMON *statement, because the order of the variables in the* EQUIVALENCE *statement takes precedence. (See Example 3.)*

6. *If an unsubscripted variable which is an array* is given in an* EQUIVALENCE *statement, the implied subscript* is one.*

7. *The integer constant* in parentheses specifies how many locations after the beginning of an array* the equivalence is to be made. (i.e., c(p) is the (p-1) location after c(1), or c(1,1), or c(1,1,1))*

EQUIVALENCE

EXAMPLES:

1. EQUIVALENCE (A,B,C) — *these are simple variables* all assigned to the same storage location*

In storage A,B,C

2. EQUIVALENCE (A,J),(B(3),D)
DIMENSION A(4), B(2,2), D(6)

In storage

A(1), J		*subscript* assumed one; floating-point* quantities can be equivalenced to integer* quantities*
A(2)		
A(3)		
A(4)		
B(1,1)		
B(2,1)		
B(1,2),	*D(1)*	
B(2,2),	*D(2)*	
	D(3)	
	D(4)	
	D(5)	
	D(6)	

3. COMMON A,B,C,D
EQUIVALENCE (C,G),(E,B) — *variables in* EQUIVALENCE *statement take precedence in common**

In common storage

C,G
B,E
A
D

RULES:

1. *FORTRAN II Rule 5 does not apply. Variables brought into common through an* EQUIVALENCE *statement will not reorder common. They may, however, increase the size of the common block as set up by the* COMMON *statement (see example 2).*

2. *Arrays may not be brought into common via an* EQUIVALENCE *statement in such a way as to cause common to extend backward.*

3. *FORTRAN II Rule 7 does not apply. Subscripted variables must be listed in* EQUIVALENCE *statements with the correct number of subscripts as specified by the* DIMENSION *statement.*

4. *When using a double word quantity in an* EQUIVALENCE *statement, care must be taken to assure that the high order part is an even number of locations away from the start of all other two word quantities which may be linked via* EQUIVALENCES.

EXAMPLES:

1.
```
COMMON A,B,C,D
EQUIVALENCE (C,G),(E,B)
```

In common storage

A	*no reordering*
B,E	*of common*
C,G	
D	

2.
```
COMMON X(2),Y,Z
EQUIVALENCE (Y,R(1))
DIMENSION R(3)
```

In common storage

X(1)	*the length of common*
X(2)	*was extended due to*
Y,R(1)	*the* EQUIVALENCE
Z,R(2)	*statement*
R(3)	

EXAMPLES: (contd.)

3.
```
COMMON J(2,2)
COMPLEX P,Q
EQUIVALENCE (P,J(2,1)),(Q,J(2,2))
```

In FORTRAN II the EQUIVALENCE *would have been (P,J(2)), (Q,J(4))*

In common storage

J(1,1)
J(2,1), P (high order)
J(1,2), P (low order)
J(2,2), Q (high order)
Q (low order)

4.
```
DIMENSION T(6)
COMMON U,V(2),W
EQUIVALENCE (V,T(4))
```

In common storage

U , T(3) *Invalid! See Rule 2*
V(1), T(4)
V(2), T(5)
W , T(6)

USE: *To specify the arrangement of data on the input or output medium; to specify the type of conversion to be made between the internal and the external format.*

n FORMAT (*Specification*)

n: *a statement-number**
Specification: *a series of codes separated by commas to determine how numeric and alphanumeric fields* should appear externally*

RULES:

General

1. *The input statements*:* READ, *and* READ INPUT TAPE, *and the output statements:* PUNCH, PRINT, *and* WRITE OUTPUT TAPE *must reference a* FORMAT *statement-number**.

2. FORMAT *statements are not executed; their function is to supply information to the object-program**. *They may be placed anywhere in the source-program* except as the first or last statement in the range of a* DO.

3. *The entire specification must be enclosed in parentheses.*

Numeric Fields

4. *For numeric fields, the following codes must be used:*

   a. *the letter* E, F, *or* I, *specifying the type of conversion to be performed;*

   b. *a number (w) specifying the width of the field on the external medium;*

   c. *a number (d) specifying the number of digits* to the right of the decimal point externally. (Only for floating-point-numbers*.)*

RULES: (contd.)

5. *Numeric specifications are written* Ew.d, Fw.d, *or* Iw.

6. Ew.d

   a. E *is used in the* FORMAT *specification when a floating-point-number* is/or will be represented externally as a decimal fraction followed by the letter E and a 2-digit decimal exponent. (e.g., 0.538E-12)*

   b. *w in this case must allow for a sign, a leading 0, a decimal point, the fraction, and four positions for the exponent.*

   c. *d represents the precision desired. (e.g., .00325 using* E10.3 *will appear as b0.325E-02)*

7. Fw.d

   a. F *is used in the* FORMAT *specification when a floating-point-number* is/or will be represented externally as a true decimal number. (e.g., -75.324)*

   b. *w in this case must allow for a sign, a decimal point, and all the digits desired.*

   c. *d represents the fractional digits. (e.g., 52.75318 using* F9.4 *will appear as bb52.7532)*

8. Iw

   a. I *is used in the* FORMAT *specification when an integer quantity is/or will be represented externally as a decimal integer*. (e.g., 4327)*

RULES: (contd.)

8. (contd.)

b. w in this case must allow for a sign, and all the integral digits. (e.g., -325 using 15 *will appear as b-325)*

9. For output, w may be larger than the actual field to provide blanks to the left of the number. If the sign is +, a blank will be inserted in the space reserved for the sign.*

Data Preparation for Numeric Fields

10. E, F, *and* I *formats apply to both input and output; however, some restrictions are eased for preparation of input.*

a. Plus signs may be omitted and no blank need be provided.

b. E *notation exponent need not have four positions. (i.e., exponent of* 10^2 *may be written* E2, E02, E 2, E 02, +02, +2*)*

c. If a decimal point is not punched, the specification will indicate where it should be placed. If it is punched, its position overrides the specification.

Scale Factors

11. It is permissible to attach a scale factor to a number which is output with E *or* F *conversion.*

*The form is n*PE*w.d or n*PF*w.d.*

P is the letter P.

n is an integer specifying the power of 10 by which the number is to be multiplied. In the case of F *conversion, the value is actually changed. For* E *converison, the scale factor merely moves the decimal point and corrects the exponent.*
(e.g., 375.417 will appear with specification of 2PF8.1 *as b37541.7*
with specification of 2PE14.5 *as bb37.54170Eb01)*

RULES: (contd.)

11. (contd.)

n may be + or - for F *conversion.*

n may only be + for E *conversion.*

12.

a. Scale factors are assumed to be zero. however, once a scale factor appears, it holds for all subsequent E *and* F *conversions within the same* FORMAT *statement.*

b. To reset the scale factor to zero, 0P *must be written.*

Alphanumeric Fields

13. For alphanumeric fields, there are two possible specifications. Both result in storing the alphanumeric information internally.*

a. A*w for Hollerith-strings* addressable by the object-program*. These are given names* and may be referred to for processing and/or modification.*

b. w H *for Hollerith strings which are used only for input and/or output. They may not be referred to or manipulated in storage in any way.*

14. A*w*

a. A is the letter A.

b. w is the field width externally. It is normally 6 since each Hollerith string variable represents six characters*.*

c. If w > 6, only the rightmost six characters will be transmitted. If w < 6, blanks are appended to the right to make six characters.

RULES: (contd.)

15. wH

a. w is an integer specifying the number of characters in the Hollerith string which appears immediately following the H.

b. H is the letter H.

c. For output, the next w characters of the Hollerith string are transmitted to the output medium.

d. For input, the next w characters on the input medium replace the Hollerith string. Subsequent use of the FORMAT *statement for output will use the new characters.*

e. There need not be a comma between the Hollerith string and the next specification.

16. Blank is a valid character in a Hollerith string and must be included in the count for Aw *and* wH.

Blank Fields

17. To specify the number of positions to be skipped for input, or of blanks to be created for output, the specification is wX.

a. w is the field width.

b. X is the letter X.

c. The next w characters are skipped on input, or the next w characters will be blank for output.

d. The X *specification need not be followed by a comma.*

RULES: (contd.)

Correspondence to List Elements

18.

a. Correspondence between E, F, I, *and* A *specifications and variables in the list of the I/O statement is simply by order of appearance.*

b. Each numeric specification must correspond to a variable in the list* of the related I/O statement.* E *and* F *specifications must apply only to the floating-point-numbers*;* I *specification must apply only to integers.*

c. Each A *specification must correspond to a variable* which represents alphanumeric information.*

d. The H *specification corresponds to no variable in the I/O list.*

e. The X *specification corresponds to no variable in the I/O list.*

Field Repetition

19. If a particular E, F, I, *or* A *specification is to be repeated for successive fields*, the specification need be written only once preceded by a number indicating how many successive fields are involved (i.e.,* 3E10.3 *is the same as* E10.3, E10.3, E10.3*).*

20. RULE 19. can also be applied to a group of specifications. Parentheses are used to show what is to be repeated (i.e., 2(F10.5, E14.4, I3) *is the same as* F10.5, E14.4, I3, F10.5, E14.4, I3*).*
NOTE: Only one level of parentheses is permissible.

Ending a FORMAT *Statement*

21. When a FORMAT *statement is used and the end of the specification is reached before the list* is terminated, the* FORMAT *statement is used again. It repeats from the last open parenthesis.*

RULES: (contd.)

22. *Each time the end of the* FORMAT *statement is reached, an input or output record is terminated.*
NOTE: Maximum record sizes:
for card - 80 characters
for line of print - 120 characters
for tape record - 120 characters.

23.

a. *A record may be terminated within the* FORMAT *statement by the appearance of a slash (/) instead of a comma between two specifications.*

b. *Consecutive slashes will cause skipping of records. (n+1 slashes will bypass n records.)*

24. *If the* FORMAT *statement has more specifications than the list* requires, the unneeded specifications will be ignored.*

Printer Carriage Control

25. *When a* FORMAT *statement is used to set up a line of print, (i.e., with* PRINT *and* WRITE OUTPUT TAPE *statements), the first character is for carriage control, and is not printed.*
NOTE: This character must be included in the character count for the record.

b causes a single space.
0 causes a double space.
1 skips to a new page.
These are usually written 1H , 1H0, 1H1

EXAMPLES:

1.
```
      READ 4,A,B,K
    4 FORMAT (E10.3,F7.1,I5)
```

2.
```
      READ 1,A,B,C,D,E
    1 FORMAT (F7.1,E13.2,A6)
```
See RULE 21.

3.
```
      WRITE OUTPUT TAPE 2,7,Y,Z,T
    7 FORMAT (E14.4,F7.1,I5)
```
Specification for T is incorrect; should be E *or* F.

4.
```
      PUNCH 5,P,Q,R(3)
    5 FORMAT (2E14.5,F10.5)
```
See RULE 19.

5.
```
      PRINT 10,V,W,ANS
   10 FORMAT (20X,2HV=F10.3,2HW=F9.3,7HANSWER=F12.3)
```
Mixing alphabetic and numeric information; also see RULE 17.

6.
```
      PUNCH 5,(A(I),B(I),I=1,10)
    5 FORMAT (E17.2,E19.3)
```
See RULE 21.

7.
```
      READ INPUT TAPE 9,12,(JOB(K),K=1,50)
   12 FORMAT (5I7)
```
This assumes 5 values per record; also see RULE 19.

8.
```
      PRINT 3
    3 FORMAT (40X,18HLIST OF QUANTITIES)
```
Printing of alphabetic information only, do not need a list.

9.
```
      PRINT 6,A(1),B(1),A(2),B(2),C,D
    6 FORMAT (30X,2(F10.3,10X,E14.5)/26X,E14.4,F8.2)
```
See RULES 20. and 23.

10.
```
      WRITE OUTPUT TAPE 2,7,I,J,K,(T(N),Q(N),N=1,20)
    7 FORMAT (I5,I3,I7/(E13.6,E15.6))
```
See RULES 21. and 23a.

11.
```
      PRINT 4,W,X,Y,Z,IND
    4 FORMAT (F10.3,3PF9.2,E15.2,F11.3,I6)
```
W has no scale factor; X, Y, and Z have a scale factor of 10^3*; IND has no scale factor.*
See RULE 12.

RULES:

Numeric Fields

1. *An additional code is provided.*

   Dw.d

   a. D *is used in the* FORMAT *specification when a double-precision-number* is/or will be represented externally as a decimal fraction followed by the letter* D *and a 2-digit decimal exponent (e.g. 0.75123482179D-03).*

   b. *w in this case must allow for a sign, a leading 0, a decimal point, the fraction and four positions for the exponent.*

   c. *d represents the precision desired (e.g. .000095143728561 using* D18.11 *will appear as b0.95143728561D-04).*

2. *Complex-numbers* will be treated as 2 separate real-numbers* and should be transmitted with 2 successive specifications (either* E *or* F*), (e.g. 12.71 + 3.8521i using* F6.2, F8.2 *will appear as b12.71bbbb3.85).*

Data Preparation For Numeric Fields

3. *FORTRAN II Rule 10 also includes* D *format.*

   a. D *notation exponent need not have four positions (same rules as for* E*).*

Scale Factors

4. *Scale factors may be used for both input and output.*

   a. *For input they apply only to* F *conversion (e.g. if the number 32.751 is read in with specification* -2PF6.3, *it will be used internally as .32751).*

   b. *For output scale factors apply to* E, F *and* D *conversion. The rules which apply to* E *format also apply to* D *format.*

5. *FORTRAN II Rule 12a also includes* D *conversion.*

Logical Fields

6. *The specification* Lw *is used for data which is being transmitted to or from logical variables*.*

   a. L *is the letter* L.

   b. *w is the field width externally.*

   c. *Input - a value of .TRUE. is stored if the first non-blank character in the data field is a T. A value of .FALSE. is stored if the first character is F or if all the characters are blank.*

   d. *Output - a value of .TRUE. or .FALSE. in storage causes a T or F, respectively, to be written in the rightmost position of the field. The rest of the field is filled with blanks.*

Correspondence To List Elements

7. a. *FORTRAN II Rule 18a includes the* D *and* L *specifications.*

   b. *Rule 18b should read "*E, F *and* D *specifications must apply only to real-numbers*."*

   c. *The* L *specification must correspond to a variable which represents a logical value (i.e. .TRUE. or .FALSE.).*

Field Repetition

8. *FORTRAN II Rule 19 includes the* D *and* L *specifications.*

Ending a FORMAT Statement

9. *FORTRAN II Rule 22 "maximum record sizes" does not apply. In FORTRAN IV they are as follows:*

for card-on line	-	*72 characters*
for card-off line	-	*80 characters*
for line of print-on line	-	*120 characters*
for tape record	-	*size of the printed line of the off-line printer*

Printer Carriage Control

10. *FORTRAN II Rule 25 should refer to the* WRITE *(i,n) list statement which replaces the* WRITE OUTPUT TAPE *statement of FORTRAN II.*

Object Time FORMAT *Statements*

11. *A* FORMAT *statement may be read in when the object program is run.*

    a. *The* FORMAT *statement is read into an array. The I/O statement which references that* FORMAT *statement uses the array name. (e.g.* READ (2,FORM) A,B,C) *FORM is the name of the array which contains the* FORMAT *statement describing* A, B *and* C.

    b. *Prior to executing the above* READ *statement the* FORMAT *statement must be read in. (e.g.*

```
      READ(2,1)(FORM(I),I=1,3)
    1 FORMAT(3A6))
```

NOTE: The entire statement excluding the word FORMAT *must be read into the array. Thus storage might appear as follows:*

FORM	(	F	1	0	.	3
+1	,	E	1	4	.	5
+2	,	F	7	.	1	)

    c. *The array containing the* FORMAT *information must be dimensioned even if its word size is one. (e.g.* DIMENSION FORM (3))

EXAMPLES:

1.

```
      LOGICAL Y
      DOUBLE PRECISION DOUBLE
      COMPLEX X
      READ (5,7) ANGLE, X, Y, DOUBLE
    7 FORMAT (F10.3,2E9.2,L5,D20.11)
```

X is read in with two specifications, E9.2, E9.2; *DOUBLE is read in with a* D *specification. Y is read in with an* L *specification.*

EXAMPLES: (contd.)

```
2.      DIMENSION WORD (2)
        READ (1,6) WORD
      6 FORMAT (2A6)
        -
        -   (dashes represent other coding)
        -
        READ (1,WORD) (X(I),Y(I),I=1,50)
```

In storage after executing the first READ statement

WORD	*(*	*6*	*E*	*1*	*2*	*.*
+1	*3*	*)*	*b*	*b*	*b*	*b*

USE: *To define a function-type subprogram*; to define its name*; to define its arguments*.*

FUNCTION $name(a_1,a_2,a_3,\ldots,a_n)$

name: *a subprogram name*
$a_1,a_2,a_3,\ldots,a_n$: *an argument; a non-subscripted variable*; an array* name*

RULES:

1. FUNCTION *must be the first statement* of a function subprogram.*

2. *The arguments may be considered dummy-arguments* which are replaced at the time the object-program* is executed by the actual arguments from the calling program.*

3. *Data may be passed through common*; however, there must be at least one argument explicitly stated in the argument list.*

4. *A function-type subprogram is referenced by using its name as an operand* in an arithmetic expression*. (See EXAMPLE 2.)*

5. *A function-type subprogram returns a single value result to the FORTRAN calling program. This result in the function subprogram is identified by the subprogram name used in the* FUNCTION *statement.*

6. *If an argument is an array name, it must appear in a* DIMENSION *statement in the subprogram. It must have the same maximum size as the actual argument in the calling program.*

EXAMPLES:

1.
```
      FUNCTION DEV(X)
      -
      -                          (dashes denote other
      -                           coding)
      DEV=X**2/PREV+3.714
```

EXAMPLES: (contd.)

2. *Subprogram*

```
FUNCTION CALC(A,B,I)
```

Calling Program

```
      _
      _
      _
S=T-CALC(R(3),Q,4)*P**2
```

Additional forms of the FUNCTION *statement are:*

REAL FUNCTION *name* $(a_1, a_2, \ldots, a_n)$
INTEGER FUNCTION *name* $(a_1, a_2, \ldots, a_n)$
DOUBLE PRECISION FUNCTION *name* $(a_1, a_2, \ldots, a_n)$
COMPLEX FUNCTION *name* $(a_1, a_2, \ldots, a_n)$
LOGICAL FUNCTION *name* $(a_1, a_2, \ldots, a_n)$

RULES:

1. *The above forms of the statement define the function to be of the type specified. These forms override the naming convention of I-N for integer functions and A-H and O-Z for real functions.*

2. *An argument may also be the dummy name of a* FUNCTION *or* SUBROUTINE *subprogram. In this case, the actual argument, which is the subprogram name, must appear in an* EXTERNAL *statement in the calling program. See type statement.*

3. *In FORTRAN II Rule 6, sentence two does not apply if the* DIMENSION *statement in the subprogram has variable dimensions. See* DIMENSION *statement. However, if a dummy argument is dimensioned, the actual argument must also be dimensioned.*

4. *A dummy argument may not appear in an* EQUIVALENCE *statement in the subprogram.*

GO TO, *assigned*

USE: *To indicate which of several statements* is the next one to be executed by using a preset path.*

GO TO n,(n_1, n_2, n_3, ..., n_m)

n: *assigned-variable**
$n_1, n_2, n_3, \ldots, n_m$: *statement-numbers**

RULES:

1. *In a previously executed* ASSIGN *statement, the integer variable* 'n' must have been assigned the value of one of the statement numbers in the parentheses.*

2. *The next statement to be executed will be the one whose statement number in the parentheses has the same value as the integer variable 'n'.*

3. *Any number of statement numbers may appear in the parentheses. (The limit exceeds all practical requirements.)*

4. *Assuming RULE 1., an assigned* GO TO *may appear anywhere in the program, except as the terminal statement of a* DO.

5. *The assigned* GO TO *statement has limited uses within a* DO *loop. (See* DO, *RULE 16.)*

EXAMPLES:

1.
```
ASSIGN 17 TO J
GO TO J,(4,17,1,3)
```
statement 17 is executed next

2.
```
ASSIGN 20 TO K
GO TO K, (20,21)
  20 STOP
```
a statement-number within the parentheses may be the next sequential statement*

EXAMPLES: (contd.)

3.

```
      ASSIGN 30 TO N
   10 GO TO N, (30,40,50)

   30 A=B
      ASSIGN 40 TO N
      GO TO 10
         -
         -
         -
   40 C=B
      ASSIGN 50 TO N
      GO TO 10
         -
         -
         -
   50 D=B
      STOP
```

coding example

(dashes denote other coding)

GO TO, *computed*

USE: *To indicate which one of several statements* is the next one to be executed; this is determined by using a previously computed value.

GO TO $(n_1,n_2,n_3,\ldots,n_m),i$

$n_1,n_2,n_3,..,n_m$: *statement-numbers**

i: *a non-subscripted integer variable**

RULES:

1. *The value of "i" can be changed throughout the program from integer* value one to integer value m.*
2. *If the value of "i" is two, the next statement to be executed will be n_2; if the value of "i" is four, the next statement to be executed is n_4, etc.*
3. *The value of "i" must be known to the object-program*when the computed* GO TO *statement is encountered.*
4. *Assuming RULE 3., a computed* GO TO *statement may appear anywhere in the program, except as the terminal statement of a* DO.
5. *Any number of statement numbers may appear in a computed* GO TO.
6. *"i" should never attain a value greater than "m".*
7. *There is no restriction on other uses for the variable "i".*

EXAMPLES:

1. `GO TO (30,45,50,9),K` — *If K=3, transfer to 50*
2. `GO TO (14,14,14,76,86,96),MIN` — *If MIN=1, 2, or 3, go to 14*

EXAMPLES: (contd.)

3.

```
      A=3
      B=4
      C=5
      K=0
    1 K=K+1
      GO TO (11,22,33),K
        -
        -
        -
   11 F=A-B
      GO TO 1
   22 E=A-C
      GO TO 1
   33 E=B-C
      GO TO 100
        -
        -
        -
  100 STOP
```

Sample Use first time will transfer to 11; second time will transfer to 22; third time will transfer to 33.

(dashes represent other coding)

4.

```
      J=1
      GO TO (10,20,3,76),J
   10 STOP
```

Any of the statement-numbers can be the next sequential statement.*

GO TO, *unconditional*

USE: *To indicate the next statement* to be executed.*

GO TO *n*

*n: statement-number**

RULES:

1. *An unconditional* GO TO *statement may appear anywhere in the source-program*, except as the terminal statement of a* DO.

EXAMPLES:

1.
```
GO TO 16
```

2.
```
GO TO 1999
```

3.
```
 5 A(I)=A(I)+C(I)
   IF (I-100)  3,2,2
 3 I=I+1
   GO TO 5
```
A coding example.

4.
```
   GO TO 35
35 STOP
```
The statement number can be the next sequential statement.

USE: *To change the sequence of statement* execution, depending upon the value of an arithmetic expression*; to make a logical decision.*

IF (a)n_1, n_2, n_3

*a: an expression**
*n_1,n_2,n_3: statement-numbers**

RULES:

1. *If a $<$ 0, control is transferred to n_1.*
2. *If a = 0, control is transferred to n_2.*
3. *If a $>$ 0, control is transferred to n_3.*
4. *This statement may not appear as the terminal statement of a* DO.

EXAMPLES:

1. `IF (A)5, 10, 15` — *A valid expression.*

2. `IF (X*Y+Z-W(2,3))5,5,10` — *The "n's" need not be all different.*

3.
```
IF (SAM)5,6,6
5  A=A+1
6  B=A-C**2
```
One or more of the statement numbers can be the next sequential statement .

IF

FORTRAN IV

In FORTRAN IV this statement is referred to as the arithmetic IF.

IF ACCUMULATOR OVERFLOW

USE: *To change the sequence of statement* execution, depending upon whether or not an overflow condition has occurred; to make a logical decision.*

IF ACCUMULATOR OVERFLOW n_1,n_2

n_1,n_2: *statement-numbers**

RULES:

1. *If an overflow condition has occurred since last tested, a transfer of control is made to* n_1.
2. *If an overflow condition has not occurred, a transfer of control is made to* n_2.

EXAMPLES:

1.
```
IF ACCUMULATOR OVERFLOW  17,10
```
If an overflow condition has occurred, transfer to 17, if an overflow condition has not occurred, transfer to 10.

2.
```
IF ACCUMULATOR OVERFLOW  36,37
36 STOP
37 K=K+1
```
One of the statement numbers can be the next sequential statement.

IF ACCUMULATOR OVERFLOW

FORTRAN IV

*This statement is not part of the FORTRAN IV language. However, the overflow indicator may be tested by using the subroutine subprogram** OVERFL *(provided by FORTRAN) as follows:*

```
CALL OVERFL(j)
```

*j: integer variable**

RULES:

1. *If an overflow condition has occurred since the last* CALL OVERFL, *j is set equal to 1.*
2. *If an overflow condition has not occurred, j is set equal to 2.*

EXAMPLES:

1.
```
CALL OVERFL(IND)
```
IND is set to 1 or 2, according to whether an overflow condition has or has not occurred.

2.
```
CALL OVERFL(K)
GO TO (3,17),K
```
If overflow, transfers to 3; if no overflow transfers to 17.

IF DIVIDE CHECK

USE: *To change the sequence of statement* execution, depending upon whether or not a divide check condition has occurred; to make a logical decision.*

IF DIVIDE CHECK n_1, n_2

n_1,n_2: *statement-numbers**

RULES:

1. *If a divide check condition has occurred since last tested, a transfer of control is made to* n_1.
2. *If a divide check condition has not occurred, a transfer of control is made to* n_2.

EXAMPLES:

1.
```
    IF DIVIDE CHECK  12,47
```
If a divide check condition has occurred, transfer to 12; if a divide check condition has not occurred, transfer to 47.

2.
```
    IF DIVIDE CHECK  38,55
38  STOP
55  K=K+1
```
One of the statement numbers can be the next sequential statement.

IF DIVIDE CHECK

FORTRAN IV

*This statement is not part of the FORTRAN IV language. However, a divide check condition may be tested by using the subroutine subprogram** DVCHK *(provided by FORTRAN) as follows:*

CALL DVCHK(*j*)

*j: integer variable**

RULES:

1. *If a divide check condition has occurred since the last* CALL DVCHK, *j is set equal to 1.*

2. *If a divide check condition has not occurred, j is set equal to 2.*

EXAMPLES:

1. CALL DVCHK(NAME)

   NAME is set to 1 or 2 according to whether a divide check condition has or has not occurred.

2. CALL DVCHK(LET)
   GO TO (4,12),LET

   If divide check, transfer to 4; if no divide check transfer to 12.

IF QUOTIENT OVERFLOW

USE: *To change the sequence of statement* execution, depending upon whether or not a quotient overflow condition has occurred; to make a logical decision.*

IF QUOTIENT OVERFLOW n_1, n_2

n_1,n_2: *statement-numbers**

RULES:

1. *If a quotient overflow condition has occurred, a transfer of control is made to* n_1.
2. *If a quotient overflow condition has not occurred, a transfer of control is made to* n_2.

EXAMPLES:

1.
```
IF QUOTIENT OVERFLOW  333,1
```
If a quotient overflow condition has occurred, transfer to 333; if a quotient overflow condition has not occurred, transfer to 1.

2.
```
IF QUOTIENT OVERFLOW  101,102
101 PAUSE
102 K=K+1
```
One of the statement numbers can be the next sequential statement.

IF QUOTIENT OVERFLOW

FORTRAN IV

*This statement is not part of the FORTRAN IV language. However, the overflow indicator may be tested by using the subroutine subprogram** OVERFL *(provided by FORTRAN) as follows:*

CALL OVERFL (*j*)

*j: integer variable**

RULES:

1. *If an overflow condition has occurred since the last* CALL OVERFL, *j is set equal to 1.*

2. *If an overflow condition has not occurred, j is set equal to 2.*

EXAMPLES:

1. CALL OVERFL(KAT)

   KAT is set to 1 or 2 depending on whether an overflow condition has or has not occurred.

2. CALL OVERFL(M)
   IF (7,5),M

   If overflow, transfer to 7; if no overflow transfer to 5.

IF SENSE LIGHT

USE: *To change the sequence of statement* execution, depending upon the condition of a Sense-Light*; to make a logical decision.*

IF (SENSE LIGHT *i*) n_1,n_2

i: *the number of a Sense Light*
n_1,n_2: *statement-numbers**

RULES:

1. *The Sense Lights are numbered 1 - 4.*
2. *If the Sense Light tested is on, a transfer of control is made to* n_1, *and the light is turned off.*
3. *If the Sense Light tested is off, a transfer of control is made to* n_2.

EXAMPLES:

1.
```
    IF (SENSE LIGHT 3)  31,32
```
If Sense Light 3 is on, transfer to 31; if Sense Light 3 is off, transfer to 32.

2.
```
    IF (SENSE LIGHT 1)  84,3
84  PAUSE
    K=K+1
```
One of the statement numbers can be the next sequential statement.

IF SENSE LIGHT

FORTRAN IV

*This statement is not part of the FORTRAN IV language. However, the sense lights may be tested by using the subroutine subprogram** SLITET *(provided by FORTRAN) as follows:*

CALL SLITET(*i*,*j*)

*i: integer expression**
*j: integer variable**

RULES:

1. *If the Sense Light specified by the value of i is on, j is set equal to 1, and the light is turned off.*

2. *If the Sense Light tested is off, j is set equal to 2.*

EXAMPLES:

1. CALL SLITET(3,LITE)

   LITE is set equal to 1 or 2 depending on whether Sense Light 3 is on or off.

2. CALL SLITET(K-1,N)
   GO TO (3,8),N

   If Light K-1 is on, transfer to 3; if it is off transfer to 8.

USE: *To change the sequence of statement* execution, depending upon the setting of a Sense-Switch*; to make a logical decision.*

IF (SENSE SWITCH *i*)n_1,n_2

i: *the number of a Sense Switch*
n_1,n_2: *statement-numbers**

RULES:

1. *The Sense Switches are numbered 1 - 6.*
2. *If the Sense Switch tested is on, a transfer of control is made to* n_1.
3. *If the Sense Switch tested is off, a transfer of control is made to* n_2.

EXAMPLES:

1.
```
IF (SENSE SWITCH 4) 13,15
```
If Sense Switch 4 is on, transfer to 13; if Sense Switch 4 is off, transfer to 15.

2.
```
IF (SENSE SWITCH 1) 2,3
   2 PAUSE
```
One of the statement numbers can be the next sequential statement.

IF SENSE SWITCH

FORTRAN IV

*This statement is not part of the FORTRAN IV language. However, the settings of the Sense Switches may be tested by using the subroutine subprogram** SSWTCH*(provided by FORTRAN) as follows:*

CALL SSWTCH(*i*,*j*)

*i: integer expression**
*j: integer variable**

RULES:

1. *If the Sense Switch specified by the value of i is on, j is set equal to 1.*
2. *If the Sense Switch tested is off, j is set equal to 2.*

EXAMPLES:

1. CALL SSWTCH(N-M+3,NSWT)

   NSWT is set equal to 1 or 2, depending on whether Sense Switch N-M+3 is on or off.

2. CALL SSWTCH(4,I)
   GO TO (17,24),I

   If Switch 4 is on transfer to 17; if it is off transfer to 24.

USE: *To cause a temporary halt; to cause the object-program* to halt in such a way that depressing the START key causes the program to continue execution with the next sequential statement*.*

PAUSE *n*

n: an unsigned integer constant or blanks.*

RULES:

1. *If 'n' is a constant, it will appear in a display register on the computer console.*

EXAMPLES:

1. PAUSE 1234

2. PAUSE

PRINT

USE: *To cause data to be transmitted from internal storage to the on-line printer.*

PRINT *n, list*

n: *statement-number* of the associated* FORMAT *statement.*

list:* *list of variables* whose values are to be transmitted.*

RULES:

1. *Line after line is printed until all the items on the list have been printed.*
2. *The number of characters per line and lines printed is determined by the* FORMAT *statement.*
3. *A minimum of one line is printed.*

EXAMPLES:

1. PRINT 2,((X(I,J),I=1,4),J=1,7),(Y(K),K=1,20)

2-dimensional array X and 1-dimensional array Y are printed according to FORMAT *statement 2.*

The order is:

X(1,1), X(2,1), X(3,1), X(4,1),

X(1,2), X(2,2), ..., X(4,2), ..., X(1,7),

....., X(4,7), Y(1), ..., Y(20)

USE: *To cause data to be transmitted from internal storage to punched cards.*

PUNCH *n, list*

n: *statement-number* of the associated* FORMAT *statement*
list:* *list of variables* whose values are to be transmitted*

RULES:

1. *Cards will be punched until all the items on the list have been punched out.*
2. *The number of cards punched depends on the* FORMAT *statement.*
3. *A minimum of one card is punched.*

EXAMPLES:

1. PUNCH 8,(A(I),B(I),I=1,100)

The order is A(1),B(1),A(2),B(2),.....,A(100),B(100)

100 elements of arrays A and B are punched according to FORMAT *statement 8.*

2. PUNCH 8,(A(I),I=1,100), (B(I),I=1,100)

In this case, the order is A(1),A(2),...,A(100),B(1),B(2),...,B(100)

READ

USE: *To cause data to be transmitted from the card reader to the internal storage.*

READ *n, list*

n: *statement-number* of the associated* FORMAT *statement.*
list:* *list of variables* whose values are to be transmitted.*

RULES:

1. *Cards will be read until all the elements in the list have been read in.*
2. *The number of cards read is determined by the* FORMAT *statement.*
3. *A minimum of one card will be read.*
4. *When the supply of cards is exhausted, the object-program* will halt.*

EXAMPLES:

1. READ 5,A,B,C(3),D(J) *four floating-point-numbers* will be read in according to* FORMAT *statement 5.*
   NOTE: *J must have been assigned a value previously.*

USE: *To cause data which is stored on a magnetic drum in internal machine notation to be read into storage.*

READ DRUM *i, j, list*

i: *integer constant*, or integer variable**
j: *integer constant, or integer variable*
list:* *list of variables whose values are to be transmitted*

RULES:

1. *No* FORMAT *statement is referenced.*
2. *i designates a logical drum.*
3. *j designates a location on the drum.*
4. *Values will be read starting from location j until the entire list has been read in.*
5. *The list may include only simple variables , subscripted-variables* with constant subscripts*, and array* names.*

EXAMPLES:

1. READ DRUM 3,152,A,B(3),I,X

READ INPUT TAPE

USE: *To cause data to be transmitted from a magnetic tape unit to the internal storage.*

READ INPUT TAPE *i, n, list*

i: *unsigned integer constant* or an integer variable* which is a logical-tape-unit* designation*
n: *statement-number* of the associated* FORMAT *statement*
list:* *list of variables whose values are to be transmitted*

RULES:

1. *Information is read from tape until all elements in the list have been read in.*
2. *The number of records read is determined by the* FORMAT *statement.*
3. *A minimum of one tape record will be read.*

EXAMPLES:

1. READ INPUT TAPE 17,6,K,(B(K)),J,(C(I),I=1,J)

   Reads from logical tape 17 according to FORMAT *statement 6. The newly read in values for K and J will be used.*

READ INPUT TAPE

FORTRAN IV

This statement is not part of the FORTRAN IV language. However, a more general input statement is provided. (See section on FORTRAN IV statements - READ, *with conversion)*

READ TAPE

USE: *To cause data which is already in machine notation to be transmitted from a magnetic tape unit to the internal storage.*

READ TAPE *i, list*

i: *unsigned integer constant* or an integer variable* which is the logical-tape-unit* designation.*

list:* *list of variables whose values are to be transmitted.*

RULES:

1. *No* FORMAT *statement is referenced.*
2. *Information is read sequentially from tape into locations corresponding to the variables in the list.*
3. *This statement will read at most one logical-tape-record.**
4. *Reading continues until either the list is exhausted or the end of the logical record is encountered.*
5. *If the end of the logical record comes first, no values will be entered for the remaining variables in the list.*
6. *If the list is exhausted first, the remainder of the logical record is by-passed, and the tape moves to the beginning of the next record.*
7. *Data written using the* WRITE TAPE *statement must be subsequently read in using the* READ TAPE *statement.*

EXAMPLES:

1. READ TAPE 17,ALPHA,BETA,INTER(I),INTER(2)

Values are read in for listed variables . No conversion is performed.

READ TAPE

FORTRAN IV

This statement is not part of the FORTRAN IV language. However, a more general statement is provided. (See section on FORTRAN IV statements - READ, *without conversion).*

RETURN

USE: *To redirect control from a subprogram* to the program which called it.*

```
RETURN
```

RULES:

1. *The* RETURN *statement must be the logical end of a subprogram.*

2. *There may be any number of* RETURN *statements in a subprogram.*

EXAMPLES:

1. RETURN — *There are no other forms of the* RETURN *statement.*

USE: *To cause a tape unit to move backwards to the beginning of the tape.*

REWIND *i*

i: unsigned integer constant or an integer variable* which is a logical-tape-unit* designation.*

RULES:

1. *A tape may be rewound by this statement and then read from or written upon.*
2. *If the tape has already been rewound, no operation is performed.*

EXAMPLES:

1. REWIND 5 — *logical tape unit 5 will be rewound.*

SENSE LIGHT

USE: *To turn the Sense-Lights* on or off so that they may be used as internal switches.*

SENSE LIGHT *i*

i: 0 or the number of a Sense Light

RULES:

1. *If i=0, all the Sense Lights are turned off.*
2. *If i = the number (integer constant*) of a Sense Light, that light is turned on.*
3. *The Sense Lights are numbered from 1 - 4.*
4. *The purpose of turning a Sense Light on or off is to test it by means of* IF (SENSE LIGHT *i*) n_1,n_2 *at times during the program and to take certain actions (transferring to statement-number* n_1 or n_2) depending upon its condition.*

EXAMPLES:

1.	SENSE LIGHT 0	*All lights are turned off.*
2.	SENSE LIGHT 3	*Sense Light 3 is turned on.*

SENSE LIGHT

FORTRAN IV

This statement is not part of the FORTRAN IV language. However, the Sense Lights may be turned on or off by using the subroutine subprogram SLITE (provided by FORTRAN) as follows:*

CALL SLITE(*i*)

*i: integer expression**

RULES:

1. *If the value of i is 0, all the Sense Lights are turned off.*
2. *If i = 1, 2, 3 or 4, the corresponding Sense Light will be turned on.*
3. *The purpose of turning a Sense Light on or off is to set a condition which may be tested by means of a* CALL SLITET *(i,j) statement at times during the program.*

EXAMPLES:

1. CALL SLITE(3*J-5) — *Sense Light 3J-5 is turned on.*

STOP

USE: *To cause the object-program* to halt in such a way that depressing the START key has no effect; to cause a permanent halt.*

STOP *n*

n: an unsigned integer constant or blanks*

RULES:

1. *If 'n' is a constant, it will appear in a display register on the computer console.*

2. *If the object program is being run under a monitor* system, this statement* causes control to be given to the monitor.*

EXAMPLES:

1. STOP 55

2. STOP

USE: *To define a subroutine-type subprogram*; to define its name*; to define its arguments*.*

SUBROUTINE *name*(a_1,a_2,a_3,..,a_n)

name: *a subprogram name*
a_1,a_2,a_3,..,a_n: *an argument; a non-subscripted variable*, an array* name*

RULES:

1. SUBROUTINE *must be the first statement* of a subroutine subprogram.*

2. *The arguments may be considered dummy-arguments* which are replaced at the time the object-program* is executed by the actual arguments from the calling program.*

3. *A subroutine-type subprogram is referenced by means of a* CALL *statement.*

4. *A subroutine-type subprogram may return more than one value to the FORTRAN calling program, by means of its arguments. These arguments must be given values in the subprogram.*

5. *If an argument is an array name, it must appear in a* DIMENSION *statement in the subprogram. It must have the same maximum size as the actual argument in the calling program.*

6. *If data is passed through common*, a subroutine subprogram need not have any arguments.*

EXAMPLES:

1.
```
      SUBROUTINE OMEGA(A,B)
      -
      -
      -
      -
      B=A**2/4.713
```
The answer may be returned to the calling program by the actual argument which corresponds to variable B.

EXAMPLES: (contd.)

2. SUBROUTINE POLY *No arguments.*

3. *Subprogram*

```
SUBROUTINE MATMPY(A,N,M,B,L,C)
```

Calling Program

```
      .
      .
      .
CALL MATMPY(X,4,3,
           Y,7,Z)
```

RULES:

1. *An argument may also be a dummy name for a function or subroutine subprogram. In this case, the actual argument, which is the subprogram name, must appear in an* EXTERNAL *statement in the calling program. See type statement.*

2. *FORTRAN II Rule 5, sentence two does not apply if the* DIMENSION *statement in the subprogram has variable dimensions. See* DIMENSION *statement.*

3. *A dummy argument may not appear in an* EQUIVALENCE *statement in the subprogram.*

EXAMPLES:

1.

Main Program

```
EXTERNAL MATADD, MATMPY
         -
         -
CALL MATRIX(MATADD,P,Q,5)
         -
         - (dashes represent
         -  other coding)
CALL MATRIX(MATMPY,R,S,7)
```

Subprogram

```
SUBROUTINE MATRIX(X,A,B,N)
           -
           -
CALL X(A,B,N)
           -
           -
           -
```

WRITE DRUM

USE: *To cause data to be written from storage onto a magnetic drum in internal machine notation.*

WRITE DRUM *i, j, list*

i: *integer constant* or integer variable**
j: *integer constant or integer variable*
list:* *list of variables whose values are to be transmitted*

RULES:

1. *No* FORMAT *statement is referenced.*
2. *i designates a logical drum.*
3. *j designates a location on the drum.*
4. *Values will be written starting at drum location j until the entire list has been transmitted.*
5. *The list may include only simple variables, subscripted-variables* with constant subscripts*, and array* names.*

EXAMPLES:

1.
```
DIMENSION T(100),R(10,10)
WRITE DRUM 5,200,I,J,T,R(7,5)
```

WRITE OUTPUT TAPE

USE: *To transmit data from internal storage to a magnetic tape unit.*

WRITE OUTPUT TAPE *i,n,list*

i: *unsigned integer constant* or an integer variable* which is a logical-tape-unit* designation*
n: *statement-number* of the associated* FORMAT *statement*
list:* *list of variables whose values are to be transmitted*

RULES:

1. *Information is written on the tape until all the items on the list have been transmitted.*
2. *The number of records written is determined by the* FORMAT *statement.*
3. *A minimum of one tape record is written.*

EXAMPLES:

1. DIMENSION ALIST (10,20)

   WRITE OUTPUT TAPE 14, 6, ALIST
   Writes on logical tape 14 according to FORMAT *statement 6 on the entire array*, ALIST column by column.*

WRITE OUTPUT TAPE

FORTRAN IV

This statement is not part of the FORTRAN IV language. However, a more general statement is provided. (See section on FORTRAN IV statements - WRITE, *with conversion)*

USE: *To write data onto magnetic tape without converting it from internal machine notation.*

WRITE TAPE *i, list*

i: *unsigned integer constant* or an integer variable* which is the logical-tape-unit* designation*

list:* *list of variables whose values are to be transmitted.*

RULES:

1. *No* FORMAT *statement is referenced.*
2. *A single logical-tape-record* is written consisting of all the elements in the list.*
3. *Values which are written using this statement must then be read in with the* READ TAPE *statement.*

EXAMPLES:

1. WRITE TAPE 4,G,H,I,J,K,(L(N),N=1,4),T

   All values are written in machine notation as one logical record.

WRITE TAPE

FORTRAN IV

This statement is not part of the FORTRAN IV language. However, a more general statement is provided. (See FORTRAN IV statements - WRITE, without conversion).

III. *FORTRAN IV Statements*

BLOCK DATA

FORTRAN IV

USE: *To define a block data subprogram*; to permit data to be entered into a labeled common* block during compilation.*

```
BLOCK DATA
```

RULES:

1. *This statement must be the first statement of a block data subprogram.*

2. *No executable statements are permitted in this subprogram. The only allowable statements are* DIMENSION, COMMON, DATA *and the type statements describing the variables*.*

3. *Data may be entered into any number of common blocks with a single block data subprogram.*

4. *Data may not be entered into blank common.*

5. *If a common block name appears in the* COMMON *statement of a block data subprogram, all the variables in the common block must be listed even if they are not being assigned data by the* DATA *statement.*

EXAMPLES:

1.
```
BLOCK DATA
COMMON/BLK1/X,Y,Z
DATA X,Y,Z/4.37,25.,32.2/
END
```
This subprogram enters data into the common block named BLK1.

2.
```
BLOCK DATA
DIMENSION T(50)
LOGICAL M
COMMON/ABC/K,ROB,M,WT/DEF/T,SUB
DATA ROB,M/12.3875,T/,(T(I),I=1,10)/
     10*0.0/
END
```
See Rule 5; K, WT, *and* SUB *do not appear in the* DATA *statement.*

DATA

FORTRAN IV

USE: *To cause data to be compiled into the object-program*; to eliminate the necessity of reading in data every time the object program is run.*

DATA $list/k*d_1,k*d_2,\ldots,k*d_m/,list/k*d_{m+1},k*d_{m+2},\ldots,$
$k*d_n/,\ldots/,list/k*d_p,\ldots,k*d_r/$

list : list of variables**
$d_1,d_2,\ldots,d_r$: *constants**
k : unsigned integer constant*

RULES:

1. *The d_i represent the actual data to be associated with the respective variable on the list. There must be a data item for each list element.*

2. *If a data field is to be repeated, it may be preceded by 'k*' to indicate the number of times it should appear (i.e., 4*5.3 is the same as 5.3, 5.3, 5.3, 5.3).*

3. *If the data field is to appear only once, the 'k*' is omitted.*

4. *If a d_i is a logical constant, it may be written as T, F, .TRUE., or .FALSE..*

5. *If the list contains subscripted-variables*, the subscript* must either be an integer constant, or must be a variable which is under control of an implied* DO *(see* DO, *implied) with a constant increment and constant limits.*

6. *Variables which are given values by a* DATA *statement may be redefined in the program.*

7. *Data may be entered into labeled common* by using a* DATA *statement in a block data subprogram. (For details see the* BLOCK DATA *statement).*

8. *Data may not be entered into blank common via a* DATA *statement.*

EXAMPLES:

1.
```
DATA A,B,I,D/3.495,.00593,
     5142,-7.2E4/
```
The value 3.495 is associated with A, .00593 with B, etc.

2.
```
DIMENSION H(5),X(2),Y(2)
DATA H(1)/15HDETAILEDbOUTPUT/,
     X(1),Y(1),X(2),Y(2)/
     2*10.,100.,1000./
```
The Hollerith characters will be in H(1), H(2), and H(3) (with H(3) containing 3 added blanks). X(1) and Y(1) will both contain the real number 10. X(2) and Y(2) will contain 100 and 1000 respectively.

3.
```
LOGICAL L,M(5),SWTCH
DATA (TABLE(I),I=1,20)/10*1000.,
     6*500.,4*10000./,SWTCH,M,L/
     T,2*.FALSE.,3*.TRUE.,F/
```
20 values are entered into the array TABLE; SWTCH gets a value of true; M(1), M(2) are false, M(3), M(4), M(5) are true, L is false.

4.
```
COMPLEX R(3)
DOUBLE PRECISION CURR, IND
DATA R/(2.1,3.459),2*(5.0,1.0)/,
     CURR,IND/7.5D1,238926.5871/
```

IF, *logical*

FORTRAN IV

USE: *To conditionally execute a certain statement* depending on whether a logical-expression* is true or false; to make a logical decision.*

```
IF(l) s
```

l: *a logical expression*
s: *an executable statement*

RULES:

1. *'s' may not be a* DO *statement or a logical* IF *statement.*

2. *If the value of 'l' is .TRUE., statement s will be executed.*

   a. *If s is a control type statement, it indicates the next statement to be executed.*

   b. *If s is not a control statement, the statement following the* IF *will be executed next.*

3. *If the value of 'l' is .FALSE., control is transferred to the next sequential statement.*

4. *If statement 's' is a* CALL *statement, the statement following the* IF *will be executed upon returning from the subprogram*.*

EXAMPLES:

1.
```
IF (A.GT.B.OR.K) GO TO 35
D=A-B*X
```
If either A>B or K has the value .TRUE., statement 35 will be executed next. Otherwise the following statement is executed.

2.
```
IF (.NOT.SWTCH) I=I+1
J=J+1
```
If SWTCH is .FALSE., both I=I+1 and J=J+1 are executed. If SWTCH is .TRUE., only J=J+1 is executed.

EXAMPLES: (contd.)

3.
```
IF (X.AND.Y.AND.Z)CALL SUBR1(X,Y,Z)
CALL SUBR2(Z,W)
```

If CALL SUBR1 *is executed the subprogram returns control to* CALL SUBR2.

NAMELIST

FORTRAN IV

USE: *To make possible the transmission and conversion of data without the use of an I/O list and a* FORMAT *statement; to permit greater flexibility in the formatting of object-program* data.*

NAMELIST/$name_1$/v,v,...,v/$name_2$/v,v,...,v/

.../$name_n$/v,v,...,v

$name_1$,$name_2$,...,$name_n$: *a* NAMELIST *name**

v: *each v is a non-subscripted-variable* or array* name*

RULES:

1. *Each list of variables has a name associated with it.*
2. *To reference a list in an I/O statement, only the* NAMELIST *name need be given.*
3. *Each name in the* NAMELIST *statement must be enclosed in slashes.*
4. *The name may be used only in* READ *or* WRITE *statements and it must not be the same as any other name in the program.*
5. *The* NAMELIST *statement must precede the* READ *and/or* WRITE *statement(s) which use a* NAMELIST *name.*
6. *A variable may appear in any number of lists in a* NAMELIST *statement.*
7. *If a variable appears in a* NAMELIST *statement it cannot be used as a subroutine argument.*
8. *If a* DIMENSION *statement is needed for a* NAMELIST *variable, it must precede the* NAMELIST *statement.*

RULES: (contd.)

Data Input

9. *The input record must contain*

   a) *$ as the second character (Note: the first character is always ignored)*

   b) *A* NAMELIST *name following the $*

   c) *one or more blank characters*

   d) *the data items to be converted and stored separated by commas (Note: the data may be contained in any number of records provided each record but the last is terminated by a data item followed by a comma.)*

   e) *$ to indicate the end of the data for a single list. (Note: a $ may not appear as the first character of a record.)*
   (e.g. b$* NAME*bbb.....data items.....$)

10. *The input data items may take on several forms.*

   a) $v = c$

      v is a simple variable or a subscripted variable which appears in the list of the specified NAMELIST *name.*

      c is a constant.*
      The value of the constant will be converted and stored in the variable

      ex: $A = 3.5, X(I) = 523.9, I=10$

   b) $v = c_1, c_2, \ldots, c_n$

      v is the name of an array which appears in the list of the specified NAMELIST *name.*

      c_i *is a constant.*

      The first constant is stored in v_1, *the second in* v_2, *etc. A constant may be preceded by a positive integer* k (i.e.* $k*c_i$*) to indicate repetition of the constant k times. A constant must be provided for each element of the array.*

RULES: (contd.)

10. (contd.)

b) (contd.)

ex: DIMENSION *P(10)*
*P = 5.3,2*7,10.352,6*3.1*

c) $v_i = c_1, c_2, \ldots, c_n$

v_i is a subscripted variable which appears in the list of the specified NAMELIST *name.*

c_i is a constant.
The first constant is stored in v_i, the second in v_i+1, etc. until all the specified constants are stored in consecutive array elements. As in 10b a constant may be preceded by 'k' if it is to be stored in k successive array elements. There may not be more constants than there are array elements between v_i and the end of the array.*

11. The type of the variable need not be the same as that of the corresponding data item. Integer*, real* and double-precision* constants may be associated with any variable except logical* or complex*. Conversion is done according to the type of the variable.*

*12. There may be no blanks embedded in a constant (i.e. Q(2)=5*b485.7 is not permitted).*

13. Data items need not appear in the input record in the same order as the variables in the list of the NAMELIST *name. Also, it is not necessary for all the list elements to receive data.*

Data Output

14. The form of the output record(s) are essentially the same as the input record(s).

15. Data is written out such that it may be subsequently read in simply by referencing the NAMELIST *name.*

RULES: (contd.)

16. *The names of all the variables and arrays in the list are written along with their values.*

17. *Arrays are written columnwise.*

EXAMPLES:

1.
```
NAMELIST/LIST1/I,X,A
READ (3,LIST1)
```

A data record might appear as

b$LIST1bbX=53.2,I=3,A=72.$

2.
```
DIMENSION R(5),S(20)
NAMELIST/CAL/P,Q,R,S,T
READ (1,CAL)
```

A data record might appear as

*b$CALbQ=327.1,R=3*2.5,2*7.5,*
*S(8)=2.4,9.1,7*33.3,T=5.2$*

NOTE: Data will be read into S(8)-S(16). However nothing will be read into S(17)-S(20) or into P.

3.
```
COMPLEX K
DIMENSION C(2)
NAMELIST/INIT/B,C,K
WRITE (5,INIT)
```

The output record might appear as

b$INITbB=0.54381279E4,C(1)=
0.85074312E-3,C(2)=0.14817289E2,K=
(0.40145792E10,0.44128567E9)$

READ, *with conversion*

FORTRAN IV

USE: *To cause data to be transmitted from a symbolic-input-device* to the internal storage.*

form 1 READ(*i*,*n*) *list*

i: *unsigned integer constant* or integer variable**
n: *statement-number* of the associated* FORMAT *statement*
list:* *list of variables whose values are to be transmitted*

form 2 READ(*i*,*v*)

i: *same as in form 1*
v: *a* NAMELIST *name**

RULES:

1. *Information is read from the symbolic input device designated by 'i'. (This must not be the card reader).*

2. *In form 1;*

   a) *information is read until all items on the list have been transmitted*

   b) *the number of records read is determined by the* FORMAT *statement.*

3. *In form 2 the information read pertains to* NAMELIST *name 'v'.*

4. *A minimum of 1 record is read.*

EXAMPLES:

1. `READ (3,7)K,(B(K)),J,(C(I),I=1,J)`

   Reads from input device 3 according to FORMAT *statement 7.*

2. `READ (2,ANY)`

   When a record is found on input device 3 which contains the name ANY, the subsequent data items are converted and stored.

USE: *To cause data which is already in machine notation to be transmitted from a symbolic-input-device* to the internal storage*

READ*(i) list*

i: *unsigned integer constant* or integer variable**
list:* *list of the variables whose values are to be transmitted*

RULES:

1. *No* FORMAT *statement or* NAMELIST *name is referenced.*
2. *Information is read sequentially from the symbolic input device designated by 'i' into locations corresponding to the variables in the list.*
3. *This statement will read at most one logical-record*.*
4. *Reading continues until either the list is exhausted or the end of the logical record is encountered.*
5. *If the end of the logical record comes first, no values will be entered for the remaining variables in the list.*
6. *If the list is exhausted first, the remainder of the logical record is bypassed and the input device is positioned to read the next sequential record.*
7. *Data written using the* WRITE*(i) list statement must be subsequently read in using the* READ*(i) list statement.*

EXAMPLES:

1. READ (3), ALPHA,BETA,GAMMA*(3)* *Values are read in for the listed variables. No conversion is performed.*

WRITE, *with conversion*

FORTRAN IV

USE: *To transmit data from internal storage to a symbolic-output-device**

form 1 WRITE(*i*,*n*) *list*

i: *unsigned integer constant** *or integer variable**
n: *statement-number** *of the associated* FORMAT *statement*
*list**: *list of variables whose values are to be transmitted.*

form 2 WRITE(*i*,*v*)

i: *same as form 1*
v: *a* NAMELIST *name**

RULES:

1. *Information is written on the symbolic output device designated by i.*

2. *In form 1,*

   a) *information is written until all the items on the list have been transmitted,*

   b) *the number of records written is determined by the* FORMAT *statement.*

3. *In form 2,*

   a) *the information written pertains to the* NAMELIST *name 'v',*

   b) *the values of the variables and arrays in 'v' are written with their names such that each output field contains all the significant digits,*

   c) *complete arrays are written columnwise,*

   d) *information output with a* WRITE *(i,v) statement can be read in with* READ(*i*,*v*) *as long as the same* NAMELIST *name is referenced.*

EXAMPLES:

1.	`WRITE (3,5)A,B,ALIST`	*Writes on output device 3 according to* FORMAT *statement 5.*
2.	`WRITE(1,INP)`	*Writes on output device 1 all the variables and arrays listed with the name INP in the* NAMELIST *statement.*

WRITE, *without conversion*

FORTRAN IV

USE: *To write data onto a symbolic-output-device* without converting it from internal machine notation*

WRITE*(i) list*

i: *unsigned integer constant* or integer variable**

list:* *list of variables whose values are to be transmitted.*

RULES:

1. *No* FORMAT *statement or* NAMELIST *name is referenced.*
2. *A single logical-record* is written on the symbolic output device designated by 'i' consisting of all the elements in the list.*
3. *Data written with this statement must be read in using the* READ*(i) list statement.*

EXAMPLES:

1. `WRITE (5)X,Y,(Z(I),I=1,10),Q(7),TABLE`

   All values are written in machine notation as one logical record.

USE: *To specify the type* of value which is to be associated with a variable* or a function*; also, to tell the FORTRAN processor the names of subprograms which are arguments of other subprograms.*

```
INTEGER v,v,...,v
REAL v,v,...,v
DOUBLE PRECISION v,v,...,v
COMPLEX v,v,...,v
LOGICAL v,v,...,v
EXTERNAL name1,name2,...,namen
```

v: *each v is a variable or function name**
$name_i$: *is a subprogram name**

RULES:

1. *If a 'v' is the name of an array*, it may be followed by parentheses enclosing 1,2, or 3 unsigned integer constant* subscripts* specifying the dimensions of the array. If this is done, the array may not appear in a* DIMENSION *or a* COMMON *(with dimension information) statement.*

2. *If a variable or function name appears in a type statement (except* EXTERNAL*) it overrides the naming convention of I-N for integer quantities and A-H and O-Z, for real quantities. The type may not be changed in the program.*

3. *The names of all subprograms which are used as arguments of other subprograms must appear in an* EXTERNAL *statement in the calling program.*

4. *A name may not appear in two type statements unless one of them is* EXTERNAL.

5. *A name must appear in its type statement before it is used in any executable statement or any* DATA *statement.*

EXAMPLES:

1. `REAL KAT,JOB,BOB`

   The variable BOB need not appear since its name correctly describes its type via the naming convention.

2. `DOUBLE PRECISION ABLE,BAKER,`
   `NOTE`

   This is the only means for defining double-precision values.

3. `EXTERNAL BOY,SIN,COS`

   These are subprogram names.

IV. *Related Topics*

A. FORTRAN functions

1. General Description:

A function* is a procedure to be followed, usually involving some calculation, which produces a single result.

In FORTRAN, one or more arguments* may be used in defining the procedure; however, the result must be a single value.

To use a function it must be defined and it must be called. There are several means within the language for defining functions, and each will be discussed in detail.

As part of the function definition, a name* and the number and mode of the arguments are established.

A function name consists of 3-6 alphabetic or numeric characters*. The first must be alphabetic. A terminal F must be appended to the name. The arguments listed in the function definition are called dummy-arguments*.

Any arithmetic expression* is valid as an actual argument.

To use an established function, one may call the function simply by writing the name of the function with a terminal F, followed by parentheses enclosing the actual arguments. i.e., A=B-C*SQRTF(D). (NOTE: There must be at least one argument.)

The actual arguments must correspond to the dummy arguments in number, order, and mode.

<u>FORTRAN IV Differences, Extensions, Exceptions</u>

A function name consists of 1-6 alphabetic or numeric characters. The first must be alphabetic.

A terminal F is not appended to a function name when the function is used, i.e. A=B-C*SQRT(D).

A. *FORTRAN functions (contd.)*

2. *Classes of FORTRAN functions:*

a. *Built-in functions:*

These functions exist within the compiler.

They will be inserted into the object-program as open-subroutines* whenever they are called.*

The function name determines the mode* of the answer. If the first character* is 'X', the answer is an integer*; if not 'X', the answer is a floating-point-number*.*

Examples:

1) ABSF(B-6.3*A) *The answer will be floating point.*

2) XMINOF(P,Q-4.2,Z) *The answer will be an integer.*

See Appendix 2 for a list of Built-in functions.

FORTRAN IV Differences, Extensions, Exceptions

The type of the function is determined by the FORTRAN processor. It may not be altered by a type statement.

ex. ABS(B-6.3*A) *the answer will be a real number*

MIN1(P,Q-4.2,Z) *the answer will be an integer*

DABS(C-D) *the answer will be a double-precision-number*.*

b. *Library functions:*

These functions exist in a library file and are available to the compiler.

These routines are compiled as closed-subroutines and will appear in the object program only once if called. Each reference simply produces a linkage to the routine.*

The arguments and the answers are in the floating point mode.*

A. *FORTRAN functions (contd.)*

2. *Classes of FORTRAN functions: (contd.)*

Examples:

1) SINF(ALPHA)
2) EXPF(D-E**2)

See Appendix 2 for a list of library functions.

FORTRAN IV Differences, Extensions, Exceptions

The library functions of FORTRAN II are classified in FORTRAN IV as Function subprograms which are available within the FORTRAN system.

See the section on Subprograms.

c. *Arithmetic Statement Functions:*

These functions are defined by single arithmetic statements at the beginning of the program.

They are incorporated into the object-program as closed-subroutines* - only once regardless of the number of references. The function name* determines the mode* of the answer. If the first character is 'X', the answer is an integer*; if not 'X', the answer is a floating-point-number*.*

To define an arithmetic statement function, see description page 18.

FORTRAN IV Differences, Extensions, Exceptions

The type of the function may be specified by having its name appear in a type statement.

If it is not so specified it will be considered an integer function if the name begins with I, J, K, L, M or N and a real function if the name begins with A-H or O-Z.

d. *Subprogram* functions:*

These functions are defined in a separate FORTRAN program and details appear in the section on Subprograms.

B. Subprograms*

1. General Description:

Subprograms are programs written to perform a specific job which is part of a larger problem.

Subprograms may be written either in FORTRAN or machine language.

A subprogram is compiled independently from its associated main-program*. A main program and all associated subprograms are brought together when the problem is run.

Subprograms are compiled as closed-subroutines* and will appear in storage only once.

There are two types of subprograms; Function Subprogram and Subroutine Subprogram.

2. Types of Subprograms:

a. Function Subprogram:

Defines a FORTRAN function as described in the section on FORTRAN functions.

The first statement of a function subprogram must be

FUNCTION $name(a_1,a_2,..,a_n)$

(see description page 61)

The value of the function is transmitted to the calling program by the subprogram name*.

The name may be 1-6 characters* long. The first character must be alphabetic. If the name begins with I, J. K. L, M, or N, the value returned will be an integer*. If the name begins with any other letter, the answer will be a floating-point-number*.

At some point in the subprogram the name

B. Subprograms (contd.)*

2. Types of Subprograms: (contd.)

a. Function Subprogram: (contd.)

of the function must be given a value.

ex.
```
FUNCTION      CALC(X,Y,Z)
   —
   —          (dashes represent other
   —           coding)
   —
CALC=X*Y-3.5
```

ex.
```
FUNCTION      ALPHA (D,E)
   —
   —
   —
   —
READ 4, ALPHA
   —
   —
   —
```

Function subprograms are called just like other FORTRAN functions, i.e., by using the function name followed by the actual arguments* enclosed in parentheses.*

If an argument is an array name, it must appear in a* DIMENSION *statement in the subprogram. It must have the same maximum size as the actual argument in the calling program.*

FORTRAN IV Differences, Extensions, Exceptions

If the FUNCTION *statement does not specify the type, then the first character of the function name determines the type, as in FORTRAN II.*

ex.
```
REAL FUNCTION MOD(A,B,I)
             -
             -
             -
             -
             -
MOD          =   A*B
```

If a dummy argument is an array name, it must be dimensioned in the subprogram; the actual argument in the calling program must also be dimensioned. However, only the dummy argument may have variable

B. *Subprograms* (contd.)*

2. *Types of Subprograms: (contd.)*

a. *Function Subprogram: (contd.)*

dimensions (see the DIMENSION *statement).*

FORTRAN IV provides a set of mathematical routines which are coded as function subprograms. The type of the arguments and the answers is predetermined.

b. *Subroutine Subprogram:*

Used instead of a function subprogram when there is no value or more than one value to be returned to the calling program.

The first statement of a subroutine subprogram must be

SUBROUTINE *name*$(a_1,a_2,\ldots,a_n)$

(see description page 95)

The name may be 1-6 characters long. The first character must be alphabetic.*

One or more of the arguments may be utilized to return values to the calling program.

Thus, at some point in the subprogram, values must be assigned to the designated arguments.*

ex.
```
SUBROUTINE    AREA(A,B,K,S,T)
    —
    —
S=A+B**K
```

Subroutine subprograms are called by executing a CALL *statement (see description page 23)*

If an argument is an array name, it must appear in a* DIMENSION *statement in the subprogram. It must have the same maximum size as the actual argument in the calling program.*

B. Subprograms (contd.)*

2. Types of Subprograms: (contd.)

b. Subroutine Subprogram: (contd.)

FORTRAN IV Differences, Extensions, Exceptions

If a dummy argument is an array name, it must be dimensioned in the subprogram; the actual argument in the calling program must also be dimensioned. However, only the dummy argument may have variable dimensions (see the DIMENSION *statement).*

V. *Appendices*

Appendix 1

Glossary

1. *Argument:*

A variable element of a subroutine which may be changed for each subroutine reference.*

examples:

X is the angle whose sine is to be calculated; X is an argument. SINF(X)

A, I, and C are arguments.

```
FUNCTION BOB(A,I,C)
```

2. *Array*

*An ordered group of quantities (which may appear in a multi-dimensional form); integer*array; floating-point* array.*

FORTRAN IV: Arrays may be integer, real*, double-precision*, complex* or logical*.*

3. *Assigned Variable:*

An integer variable which is given a statement-number* value by an* ASSIGN *statement*

examples:

K is an assigned variable.

```
ASSIGN 7 TO K
```

4. *Character:*

alphabetic, digit, special

alphabetic example: A,B,C,.....,Z

digit example: 0,1,2,.....,9

special example: +,-/,*,=,(,),.

5. *Closed Subroutine:*

A subroutine so designed that it can be called or referenced from more than one location in a program, can return control to the place from which it was called, and is reusable in memory.

examples:

library functions, arithmetic statement functions, subprograms (i.e., SIN, COS, MATMPY*)*

6. *Common:*

A designated internal storage area available to more than one program.

FORTRAN IV: There are two kinds of common: labeled common and blank (or unlabeled) common. Labeled common is divided into named blocks. Blocks having the same name in different programs share storage. Blank common is the same as FORTRAN II common.

7. *Complex Number:*

A number having a real and an imaginary part; it is expressed as two real-numbers separated by commas and enclosed in parentheses.*

examples:

the first number is the real part, the second is the imaginary part (this is 14.3+7.2i)	(14.3,7.2)
either part of or both may be 0	(20.1E-2,0.0)

8. *Constant:*

An integer, a floating-point-number**

example of integer constants:

valid integer	0
may have + sign	+6
if negative, must have minus sign	-19683
if positive, may be unsigned	26
must not have a decimal point	3249

8. *Constant: (contd.)*

example of floating point constants:

must have decimal point	0.0
point may be at the beginning	.328
point may be at the end	-53.
*point may be between two digits**	+4.159
negative numbers must have minus sign	-512.356
positive numbers may be unsigned	.002512
any of the above may be followed by letter E and a decimal exponent (this means 4.7×10^3 or 4700)	4.7E3
exponent may have a sign	33.1E-2
exponent has a maximum of two digits	41.33E15

FORTRAN IV: An integer, a real-number*, a double-precision-number*, a complex-number*, a logical* value.*

example of integer constants:

same as FORTRAN II

example of real constants:

same as FORTRAN II floating point constants

also: may have up to 9 digits	8327.58147E4

example of double precision constants:

must have decimal point	31.58947514
point may be anywhere	2375892146.

8. *Constant: (contd.)*	*if fewer than 10 digits must be written with the letter D and a decimal exponent (this is $29.4x10^5$ or 2,940,000)*	29.4D5
	if 10 or more digits letter D is optional	94385.17629D-3
	exponent is a maximum of 2 digits and may be signed	.375D-12
	example of complex constants: must consist of 2 real numbers separated by commas and enclosed in parentheses	(4.983,72.5)
	first number is real part of constant; second number is coefficient of imaginary part (this is 2.5+9.1i)	(2.5,9.1)
	both parts of constant may be signed	(-3.8,+1.2E4)
	example of logical constant: may be either true or false	.TRUE.
	the decimal points must always precede and follow the constant	.FALSE.
9. *Digit:*	*positive integer* less than 10 example:* 0, 1, 2, ..., 9	
10. *Double Precision Number:*	*A number written with a decimal point which is expressed internally as a decimal fraction with 10 or more significant digits times a power of 10. examples: must have a decimal point and at least 10 digits*	1.493217658

10. *Double Precision Number:* (contd.)

may have fewer than 10 digits if the letter D followed by a decimal exponent appears (this is $3.5x10^2$ or 350) 3.5D2

11. *Dummy Argument:*

A variable such as the one which appears in the argument* list of a function* definition but which is replaced by the actual argument when the function is used.*

12. *Expression:*

A sequence of constants, variables*, functions*, operators*, and parentheses indicating that a calculation is to be performed; also a single constant, variable, or function.*

Rules of Calculation:
The hierarchy of operations is as follows:

1.	*exponentiation*	**
2.	*multiplication, division*	*, /
3.	*addition, subtraction*	+, -

For operations on the same level, the order of calculation is from left to right.
examples:

single variable	T
single constant	5
single function	COSF(T)
specifies one calculation	B-9.3
specifying many calculations	X*Y-Z**2+3.82
exponentiation is performed first (this is $4.3x^3$)	4.3*X**3

12. *Expression: (contd.)*	*multiplications and divisions are performed before additions and subtractions (this is a+bc)*	A+B*C
	examples:	
	additions and subtractions are done last (this is $x+\frac{y}{z}-w^3$*)*	X+Y/Z-W**3
	calculations in parentheses have priority (this is $\frac{(x+y)}{(z-w)^3}$*)*	(X+Y)/(Z-W)**3
	same level calculations (i.e., multiplication and division, or addition and subtraction) are done from left to right (this is $\frac{ac}{b}$*)*	A/B*C
	This is $\frac{a}{bc}$	A/(B*C)
	illegal! The mode of an expression must be either integer* or floating-point**	X-5
	integer quantities in floating point expressions are permitted as arguments of functions, subscripts, and exponents*	ANYF(Z,2)*X(3)-Y**4

12. *Expression: (contd.)*

FORTRAN IV: The FORTRAN IV language contains 2 kinds of expressions: arithmetic and logical. Arithmetic expressions are the same as they were in FORTRAN II with some extensions. Logical expressions will be described in the glossary under Logical-expression.*

All rules for FORTRAN II apply except the one concerning the mode of the expression.

an expression may consist entirely of integer quantities	I-3*IABS(K)
no other types of quantities may appear in an integer expression except as function arguments	L+IFCN(T)
real quantities may be combined (for +,-,,/) with either double precision or complex quantities (Note: The result will be double precision or complex respectively.)*	4.2*(5.3,7.9) A+5.3D7
illegal! double precision may never appear with complex	.72D1-(5.3,7.9)
logical quantities may never appear in arithmetic expressions (see logical-expressions)*	A.GT.B
integers may appear as exponents for real, double precision and complex quantities	ABLE**2+ (22.7,5.3)**3
illegal! complex quantities may not appear as exponents	Y**(2.1,3.5)

13. *Field:*

A single character or group of characters which logically constitute one item of information*
examples:

numeric field (all numbers)

alphanumeric field (combination of alphabetic and numeric characters)

14. *Fixed Point Number:*

*an integer**
See examples of Integers

15. *Floating Point Number:*

A number written with a decimal point which is expressed internally as a decimal fraction times a power of 10.*
examples:

whole number with a decimal point	1.
decimal fraction	.99
any of the above may have E notation (decimal exponent) which means 3.5 x 10^2, or 350	3.5E2
same as .95 x 10^{-3}, or .00095	.95E-3

FORTRAN IV: See Real-Number.*

16. *Function:*

A set of procedures to be followed for which there is a single answer
examples:

subprogram functions
arithmetic statement functions
built-in functions
library functions

16. *Function: (contd.)*

FORTRAN IV: Library functions are classified as subprogram functions.

17. *Hollerith String:*

A series of alphabetic, digit, and special characters examples:*

ABC1234

ANSWERS TO THE PROBLEM

18. *Integer:*

A whole number examples:*

*can be one digit**	3
valid integer	0
positive integers may have plus sign	+794
negative integers must have minus sign	-93

19. *List:*

A sequence of variables, separated by commas, usually appearing in an input or an output statement. examples:*

simple variables	A,B,K
single values of subscripted-variables (Note: I must have been given a value previously)*	C(3),D(I), T(4,7)
a group of values of a subscripted variable (Note: This is called an implied DO*)*	(D(I),I=1,5), ((E(J,K),J=1,10), K=1,20)
*an entire array**	MATRIX

19. *List: (contd.)*

any combination of the above examples	C,TABLE,(JOB(I), I=1,3)
The list may not include constants except if used as an argument* list*	R(2),S,3.721

FORTRAN IV: A list may also appear in a DATA *statement.*

20. *Logical Expression:*

A sequence of logical constants, logical variables*, logical functions*, logical operators*, arithmetic expressions* separated by relational operators*, and parentheses, indicating that a calculation is to be performed; also a single logical constant, variable or function.*

Rules of Calculation:
The hierarchy of operations is as follows:

exponentiation	**
multiplication, division	*,/
addition, subtraction	+,-
*relational operators**	.LT.,.LE.,.EQ., .NE.,.GT.,.GE.
negation	.NOT.
intersection	.AND.
union	.OR.

For operations on the same level, the order of calculation is from left to right.
examples:

the result of a logical expression is either .TRUE. or .FALSE.	A.EQ.B
single variable	LOGIC
single constant	.TRUE.
single function	LFCN(SWTCH)

20. *Logical Expression: (contd.)*	*integer quantities may be combined by relational operators only with other integer quantities*	I.NE.J
	real and double precision quantities may be combined by relational operators	BOB.GE.31.3D5
	illegal! logical quantities may not be combined by relational operators	LOGIC.EQ..TRUE.
	complex quantities may appear in logical expressions only as function arguments	A.GT.REAL(COMP)
	only logical quantities may be connected by the logical operators (this is .TRUE. if SW1 is .TRUE. or if both SW2 and SW3 are .TRUE. and A≠B)	SW1.OR.SW2.AND. SW3
	calculations in parentheses have priority (this is .TRUE. only if D is .TRUE. and either C is .TRUE. or A=B)	(A.EQ.B.OR.C) .AND.D

21. *Logical record:* *See logical-tape-record*. The record so defined need not appear on magnetic tape. It may be on any I/O device capable of handling it.*

22. *Logical tape record:* *That amount of information which is defined by a single input or output list*. It may be made up of several physical records, each containing a signal word which determines the boundaries of the logical record.*

23. *Logical tape units:* *Numbers used to refer to each tape unit in writing a FORTRAN program.*

24. *Logical Value:* *A value of true or false.*
examples:

true	.TRUE.
false	.FALSE.

25. *Loop, or Looping:* *The process of repeating a sequence of instructions a number of times until a terminal condition is reached.*

26. *Main Program:* *A program which can be executed without being called upon by another program. However, it may reference any number of associated programs called subprograms*.*

27. *Monitor:* *A master control program which governs the flow of jobs through a computer.*

28. *Mode:* *Form of a number*; form of a name*; form of an expression*. Modes may be integer* or floating-point*. Mode of a number is determined by the presence or absence of a decimal point. Mode of a variable or function is determined by its name. Mode of an expression is determined by the modes of its operands*.*
examples:

a constant in the integer mode*	6
a constant in the floating point mode	6.
a variable in the floating point mode*	ALPHA
a variable in the integer mode	MONEY
a function name in the floating point mode	SINF(X)
a function name in the integer mode	XANYF(Y,P,3)

28. *Mode: (contd.)*

a function name in the floating point mode	FIRSTF(A,T,4.7)
an expression in the floating point mode	X-Y*5.27
an expression in the integer mode	I*J-K
an expression in the floating point mode	A-B**2
an expression in the floating point mode	V-SQRTF(7.51)-W(3,4)

FORTRAN IV: The term mode is usually referred to as 'type' in FORTRAN IV (see Type in glossary).

29. *Name:*

A label given to an integer, to a floating-point-number*, to a Hollerith-String*, to a function*, to a subprogram*. Data with a name is considered a variable*; hence, distinction between variable and name is conceptual.*

examples:

	Name	*Data*
integer name (1-6 alphabetic or numeric characters; first character must be I, J, K, L, M, or N)*	JOB	425
floating point name (1-6 alphabetic or numeric characters; first character must be A-H or O-Z)	ALPHA	22.715
Hollerith string name (1-6 alphabetic or numeric characters; first character must be alphabetic). Note: This is included to emphasize that alphabetic data may be manipulated as numeric when assigned a name.	DATE	1/30/5

29. *Name:*
(contd.)

	Name	Data
function name (3-6 alphabetic or numeric characters; first character must be alphabetic; first character must be X if, and only if, answer is to be an integer.)	ABS	33.7
function subprogram name (1-6 alphabetic or numeric characters; first character must be I, J, K, L, M, or N if answer is to be an integer; first character must be A-H or O-Z if answer is to be a floating point number)	CALC	9418.2
subroutine subprogram name (1-6 alphabetic or numeric characters; first character must be alphabetic)	MATMPY	*No data is associated with a subroutine sub program nam*

FORTRAN IV: A label given to an integer, to a real-number*, to a double-precision-number*, to a complex-number*, to a logical* value, to a Hollerith string, to a function, to a subprogram, to a common block; also a label which is part of a* NAMELIST *statement.*
examples:

integer name (1-6 alphabetic or numeric characters; may begin with any letter if the name appears in an INTEGER *type statement)*	TAB	370

29. *Name:* *(contd.)*

	Name	*Data*
real name (1-6 alphabetic or numberic characters; may begin with any letter if the name appears in a REAL *type statement)*	LENGTH	36.37
a double precision name (1-6 alphabetic or numberic characters; may begin with any letter and the name must appear in a DOUBLE PRECISION *type statement)*	G24	.379841D6
a complex name (1-6 alphabetic or numberic characters; may begin with any letter and the name must appear in a COMPLEX *type statement)*	SOS	24.3+7i
a logical name (1-6 alphabetic or numberic characters; may begin with any letter and the name must appear in a LOGICAL *type statement)*	LOGIC	.TRUE.
Hollerith string name (1-6 alphabetic or numeric characters; first character must be alphabetic)	DATE	1/30/5
function name (1-6 alphabetic or numberic characters; may begin with any letter if the name appears in a type statement)	ANY	225

29. *Name:* *(contd.)*

	Name	Data
function subprogram name (1-6 alphabetic or numeric characters; may begin with any letter if the name appears in a FUNCTION *statement with its type)*	POLY	48.3
subroutine subprogram name (1-6 alphabetic or numeric characters; may begin with any letter)	MATRIX	*No data is associated with a subroutine subprogram name*
common block name (1-6 alphabetic or numeric characters; may begin with any letter)	BLK1	*No data is associated with a common block name*
a NAMELIST *name (1-6 alphabetic or numeric characters; may begin with any letter)*	LIST1	*No data is associated with a* NAMELIST *name*

30. *Number:* *a string of digits, either signed or unsigned, in admissible FORTRAN form; an integer, a decimal number*
examples:
86
-777
+9357
0
1.768
3.7E+3

31. *Object Program:* *a machine language program produced from a source-program**

32. *Open Subroutine:* *a subroutine designed only for sequential execution, not referenceable from other parts of a program; usually very short.*

33. *Operands:* *constants*, variables*, and functions* connected by operators* in an arithmetic expression**

examples:

a variable and a constant	A+5.31
a subscripted-variable and a simple variable*	B(3)*C
a function and a subscripted variable	SQRTF(D)-X(I)
a constant and a subscripted variable	5-JOB(3)

FORTRAN IV: Also pertains to constants, variables and functions in a logical expression.

34. *Operators:* *symbols designating arithmetic*

examples:

addition	+
subtraction	-
multiplication	*
division	/
exponentiation	**

FORTRAN IV: Also symbols designating logical and relational operations

examples:

arithmetic operators (same as FORTRAN II)

logical operators

negation	.NOT.
intersection	.AND.
union	.OR.

relational operators

greater than	.GT.
greater than or equal to	.GE.
less than	.LT.
less than or equal to	.LE.
equal to	.EQ.
not equal to	.NE.

35. *Real Number:* A number written with a decimal point which is expressed internally as a decimal fraction of 8 or fewer significant digits times a power of 10 (same as floating-point-number* in FORTRAN II).
examples:

whole number with a decimal point	1.
decimal fraction	.99
any of the above may have E notation (decimal exponent) which means 75.1×10^2 or 7510.	75.1E2
same as $.22 \times 10^{-3}$ or .00022	.22E-3

36. *Sense Light:* A light on a computer console which may be turned on, turned off, and tested by a computer instruction.

37. *Sense Switch:* An external switch on a computer console which may be turned on or off by a machine operator andmay be tested by a computer instruction.

38. *Source Program:* A program written in a language other than machine language, such as FORTRAN.

39. *Statement:* An instruction in the FORTRAN language
examples:

arithmetic statement	A=B+C*D
control statement	GO TO 45
specification statement	DIMENSION X(15)

40. *Statement Number:*

An integer associated with a single FORTRAN statement* which may be used to refer to that statement within a program.*
examples

smallest permissible statement number	1
largest permissible statement number	32767
must be an unsigned integer	394

41. *Subprogram:*

A program which cannot exist alone but must be executed in conjunction with at least one other program called a main-program; there are two types of FORTRAN subprograms - function subprogram and subroutine subprogram.*

FORTRAN IV: There is an additonal type of subprogram called a block data subprogram. It is used when data is to be entered into a labeled common block during compilation.
example:

```
BLOCK DATA
```

42. *Subroutine:*

A set of instructions to perform some well-defined procedure which is usually of a recurring nature.
examples:

Sine, cosine, square root

43. *Subscript*

*A designation of one element within an array**
examples:

integer constant designating 2nd number in an array*	A(2)

43. *Subscript (contd.)*

integer variable designating the ith number in an array*	B(I)
integer variable plus (or minus) an integer constant, designating the j plus 3rd number in an array	C(J+3)
integer constant times an integer variable, designating the 7mth number in an array	D(7*M)
integer constant times an integer variable, plus (or minus) an integer constant, designating the 4k-3rd number in an array	E(4*K-3)
combinations of the above	F(I-4,3*K,L)

44. *Subscripted Variable:* *A variable* which is an array* element having its subscripts* enclosed in parentheses, and separated by commas.*

examples:

the 5th number in a floating-point array whose name is A*	A(5)
the i-2nd number in the floating point array named TABLE	TABLE(I-2)
2-dimensional array element	JOB(3,L-4)
illegal! Array names may not end in 'F' if they are 4, 5, or 6 characters in length*	NAIF(3)

45. *Symbol:* *A character* which is part of a particular language.*
examples:

alphabetic characters	A,B,C,.....,Z
digit characters	0,1,2,.....,9
special characters	+, -, /, *, (,)

46. *Symbolic device (input or output):* *Technique for referencing an I/O unit without specifically designating it. This permits flexibility in the use of the I/O units. The actual correspondence is handled by the FORTRAN processor.*

47. *Type:* *Form of number*, form of a name*, form of an expression*. Types may be integer*, real*, double_precision*, complex*, and logical*. The type of a number is determined by a type statement or by its name. The type of an expression is determined by the types of its operands*.*
examples:

an integer constant	273
a real constant	3.75E2
a double precision constant	14.5D-3
a complex constant	(22.1,3.3E5)
a logical constant	.FALSE.
integer variable	INTEGER CAR
complex variable	COMPLEX POLY
real function	SIN (A*B)
double precision function	DSIN (A*B)
logical expression	P.EQ.Q.AND.CHK

48. *Variable:* *Integer* with a name*; a floating-point-number* with a name; a Hollerith-String* with a name; a variable is referred to in a program by its name.*
examples:

48. *Variable: (contd.)*

	Value of Variable	*Name*
integer variable (name may be from 1-6 alphabetic or numeric characters: first character is I, J, K, L, M, or N)	347	NET
floating point variable (name may be from 1-6 alphabetic or numeric characters; first character is A-H, O-Z)	93.2	X15
Hollerith-string (name may be 1-6 alphabetic or numeric characters; first character must be alphabetic)	C12C3=	STRING

FORTRAN IV: A double precision number with a name, a complex number with a name, a logical value with a name.
examples:

double precision variable (name may be from 1-6 alphabetic or numeric characters and must appear in a DOUBLE PRECISION *type statement)*	327.4217892	TOT
complex variable (name may be from 1-6 alphabetic or numeric characters and must appear in a COMPLEX *type statement)*	32+12i	IMAG
logical variable (name may be from 1-6 alphabetic or numeric characters and must appear in a LOGICAL *type statement)*	.FALSE.	SWTCH

Appendix 2

FORTRAN Built-In Functions and Library Functions

FORTRAN Built-In Functions and Library Functions

Built-In

*FUNCTION NAME**		*MODE*		*NO. OF*	*ACTIVITY*
FORTRAN II	*FORTRAN IV*	*ARG.*	*ANSWER*	*ARGS.*	*PERFORMED*
ABSF	ABS	*Real*	*Real*	1	*Absolute Value*
XABSF	IABS	*Integer*	*Integer*	1	
DIMF	DIM	*Real*	*Real*	2	*arg1 - min(arg*$_1$*, arg*$_2$*).*
XDIMF	IDIM	*Integer*	*Integer*	2	
FLOATF	FLOAT	*Integer*	*Real*	1	*Convert Integer to Real*
XFIXF	IFIX	*Real*	*Integer*	1	*Convert Real to Integer*
INTF	AINT	*Real*	*Real*	1	*Extract Largest Integer*
XINTF	INT	*Real*	*Integer*	1	
MAXOF	AMAX0	*Integer*	*Real*	$\geq$ 2	
MAX1F	AMAX1	*Real*	*Real*	$\geq$ 2	*Select*
XMAXOF	MAX0	*Integer*	*Integer*	$\geq$ 2	*Largest Value*
XMAX1F	MAX1	*Real*	*Integer*	$\geq$ 2	
MINOF	AMIN0	*Integer*	*Real*	$\geq$ 2	
MIN1F	AMIN1	*Real*	*Real*	$\geq$ 2	*Select*
XMINOF	MIN0	*Integer*	*Integer*	$\geq$ 2	*Smallest Value*
XMIN1F	MIN1	*Real*	*Integer*	$\geq$ 2	
MODF	AMOD	*Real*	*Real*	2	$arg_1 - \left[arg_1/arg_2\right] * arg_2$
XMODF	MOD	*Integer*	*Integer*	2	
SIGNF	SIGN	*Real*	*Real*	2	$Sign(arg_2) * \lvert arg_1 \rvert$
XSIGNF	ISIGN	*Integer*	*Integer*	2	

*FUNCTION NAME**		*MODE*		*NO. OF ARGS.*	*ACTIVITY PERFORMED*
FORTRAN II	*FORTRAN IV*	*ARG.*	*ANSWER*		
ATANF	ATAN	*Real*	*Real*	*1*	*Arctangent*
COSF	COS	*Real*	*Real*	*1*	*Trigonometric Cosine*
EXPF	EXP	*Real*	*Real*	*1*	*Exponential*
LOGF	ALOG	*Real*	*Real*	*1*	*Natural Logarithm*
SINF	SIN	*Real*	*Real*	*1*	*Trigonometric Sine*
SQRTF	SQRT	*Real*	*Real*	*1*	*Square Root*
TANHF	TANH	*Real*	*Real*	*1*	*Hyperbolic Tangent*
	ALOG10	*Real*	*Real*	*1*	*Common Logarithm*
	ATAN2	*Real*	*Real*	*2*	*Arctangent*(arg_1/arg_2)

Appendix 3

FORTRAN Symbols with Equivalent Codes

FORTRAN Symbols with Equivalent Codes:

SYMBOL	CODE	SYMBOL	CODE
A	12- 1	X	0- 7
B	12- 2	Y	0- 8
C	12- 3	Z	0- 9
D	12- 4	0	0
E	12- 5	1	1
F	12- 6	2	2
G	12- 7	3	3
H	12- 8	4	4
I	12- 9	5	5
J	11- 1	6	6
K	11- 2	7	7
L	11- 3	8	8
M	11- 4	9	9
N	11- 5	/	0- 1
O	11- 6	+	12
P	11- 7	-	11
Q	11- 8	*	11, 8- 4
R	11- 9	.	12, 8- 3
S	0- 2	,	0, 8- 3
T	0- 3	=	8- 3
U	0- 4	blank	
V	0- 5	(	0, 8- 4
W	0- 6	)	12, 8- 4