Mark I
Time-Sharing Service

Reference Manual

Information Systems

Time-Sharing Service

FORTRAN Language

GENERAL ELECTRIC

INFORMATION SERVICE DEPARTMENT

Mark I
Time-Sharing Service

FORTRAN LANGUAGE

Reference Manual

August 1966
Revised 4-67
Revised 9-68

INFORMATION SERVICE DEPARTMENT

PREFACE

This manual explains and tells how to use Mark I Time-Sharing FORTRAN language available to customers of General Electric Information Service Department.

The Introduction lists the many features available when using FORTRAN in a time-sharing environment. Users familiar with FORTRAN should consult the lists in the Introduction for differences between the Mark I Time-Sharing FORTRAN language and FORTRAN II and FORTRAN IV, respectively.

Users unfamiliar with FORTRAN should read the General Electric manual entitled "Introduction to FORTRAN", 227106, available from any General Electric Time-Sharing Service representative. Writing and running trial programs on the time-sharing system is also encouraged as a means of learning the language.

Appendix A is an easy-to-use reference and summary. It lists the rules for writing each kind of statement and supplies a page reference to the detailed discussion of the statement in the body of the manual.

An additional convenience, an alphabetical index of statements is provided. It is located immediately following the table of contents. This index shows the statement form and lists the page numbers for the statement rules and the statement discussions.

Users should also become familiar with the powerful Mark I FORTRAN library programs available with General Electric's Computer Time-Sharing Service. They are described in the Time-Sharing Library Catalog, 228116.

This edition supercedes the previous editions.

CONTENTS

CONTENTS (CONT'D)

CONTENTS (CONT'D)

ALPHABETIC SUMMARY OF FORTRAN STATEMENTS

General Statement Form	Statement Category	Page References	
		Rules (App. A)	Detailed Discussion
variable or array element = expression	Arithmetic	104	21
mode qualifier, arithmetic statement function (dummy argument list) = expression	Subprogram Definition	105	46
mode qualifier, internal function (dummy argument list) : first statement	Subprogram Definition	105	46
. . . . .	. . . . .	. . . . .	. . . . .
ASSIGN label or label variable, TO label variable	Control	105	30
BACKSPACE file reference	I/O File	105	67
CALL subprogram or entry (actual argument list)	Control	105	37
mode qualifier COMMON array, array (dimension specification), variable, . . .	Declaration	106	23
CONTINUE	Control	106	34
mode qualifier DIMENSION array (dimension specification), variable . . .	Declaration	106	25
DO label control variable = initial value, final value, increment	Control	106	32
END	Declaration	106	38
END internal function	Declaration	107	46
ENTRY entry (dummy argument list)	Subprogram Definition	107	49
mode qualifier EQUIVALENCE (array, array element, variable, . . .), (equivalence), . . .	Declaration	107	26
EXTERNAL subprogram entry, . . .	Declaration	108	28
FORMAT (format specification sequence)	Input/Output	108	71

General Statement Form	Statement Category	Page References	
		Rules (App. A)	Detailed Discussion
mode qualifier FUNCTION external function (dummy argument list)	Subprogram Definition	108	41
GOTO label or label variable	Control	108	29
GOTO (label, label variable, . . .), control variable	Control	108	29
IF (expression) label reference, . . .[negative, zero, positive]	Control	108	30
IF (END FILE file reference) label reference, . . . [no end of file, end of file]	I/O File	108	31
INPUT format reference, input list	I/O Terminal	109	59
INTEGER array, array (dimension specification), external subprogram, variable, . . .	Declaration	109	41
PAUSE constant or variable	Control	109	38
PRINT format reference, output list	I/O Terminal	109	57
READ format reference, input list	I/O File	110	69
READ (file reference, format reference) input list	I/O File	110	60
REAL array, array (dimension specification), external subprogram, variable, . . .	Declaration	110	41
RETURN	Control	111	37
REWIND file reference	I/O File	111	67
STOP constant or variable	Control	111	38
SUBROUTINE subroutine (dummy argument list)	Subprogram Definition	111	41
WRITE format reference, output list	I/O File	111	69
WRITE (file reference, format reference) output list	I/O File	111	60

1. INTRODUCTION

FEATURES AND BENEFITS

General Electric's versatile Mark I FORTRAN is an enrichment of FORTRAN II, which has been specially adapted for remote terminal and time-shared use.

The following features and capabilities, not normally available in the more traditional FORTRAN languages, have been added in Mark I FORTRAN.

- Mixed real and integer expressions, parameters, library function calls, and input/output
- Unformatted input and standard format output as well as extended formatting facilities
- Unrestricted subscripting with or without subscript checking
- Extended subprogram facilities provided by internal subprograms (a generalization of the statement function) and entry statements
- Alphabetic capabilities based on short quoted literals in expressions and quoted literals of unlimited length for input or output
- Temporary and permanent file, and terminal input/output using FORTRAN IV style statements
- Relaxed naming rules including--

  acceptance of longer names
  specific and general mode declarations
  both FORTRAN II and FORTRAN IV style library and statement function names
  statement label names as well as numbers

- Simplified statement form that does not require starting in a particular column and allows--

  multiple statements per line
  unlimited continuation of a statement from line to line
  embedded comments

- Monitor system facilities that allow--

  paging and conditional compilation of source programs
  naming of data and permanent files
  chaining of program execution

FORTRAN II Incompatibilities

Statements that must be omitted or altered:

READ TAPE	use READ (file)
WRITE TAPE	use WRITE (file)
READ INPUT TAPE	use READ (file, format)
WRITE OUTPUT TAPE	use WRITE (file, format)
READ DRUM	omit
IF ACCUMULATOR OVERFLOW	omit
IF QUOTIENT OVERFLOW	omit
IF DIVIDE CHECK	omit
IF (SENSESWITCH)	use variable called SENSESWITCH
IF (SENSELIGHT)	use variable called SENSELIGHT
FREQUENCY	omit
PUNCH	omit
F-card	use EXTERNAL

Facilities not available:

Boolean statement
Double precision statement
Complex statement
Slew control by first character of the output record
Intrinsic or library functions: DIMF, XDIMF, TANHF , EXIT

Changed facilities:

Largest common declaration must precede other common and all dimension declarations

Arithmetic Statement Function (ASF) name beginning with X does not imply integer function

Equivalence does not reorder common storage

Formatted output is subject to field widening rather than value truncation

FORTRAN IV Incompatibilities

Statements that must be omitted or altered:

COMPLEX	omit
DOUBLE PRECISION	omit
LOGICAL	omit
IF (logical)	omit
NAMELIST	omit
DATA	use $DATA and READ temporary file
PUNCH	omit
BLOCKDATA	omit
DEBUG	omit

Facilities not available:

Block common
Intrinsic and library subprograms: all complex, double precision, machine indicator tests, DIM, IDIM, TANH, ALOG10, EXIT
Adjustable dimensions
Slew control by first character of the output record

Changed facilities:

Largest common declaration must precede other common and all dimension declarations

Mode of ASF dummy arguments cannot be declared

Formatted output subject to field widening rather than value truncation

2. FORM FOR TIME-SHARING FORTRAN STATEMENTS

In Mark I FORTRAN, a source program is entered in a sequence of numbered lines. In addition to the required line numbers, each line may contain:

- One or more FORTRAN statements
- A continuation of the statement on the previous line
- One or more labels (numbers or names) for each statement (or, there may be none.)
- Comments, either embedded within each statement or comprising the entire line
- Control information

The rules for writing each of these entries follow.

Line Number

Each line must begin with a 1 to 5 digit sequence number. The operating system uses this number to sequence the lines. The FORTRAN system uses the number to refer to parts of the program written on the line. The line number cannot, however, be referred to from within the program, as it can in a BASIC program, and should not be confused with the FORTRAN statement number.

Unless the entire line is a comment, the first character after the line number must be a blank.

Statement

A line may contain one or more statements. If more than one statement is entered on the same line, each statement except the last must be terminated by a semicolon. A semicolon may, but need not, terminate the only or last statement on the line.

Example

```
140 I=12; J=15; K=0
150 5 A(I)=B(I)=F(I);
160 B(I)=B(I)/C(I); 6 T=T-B(I)
```

Semicolons separate the three statements in line 140. The semicolon after the statement in line 150 is optional. Notice that a blank between the line number and statement number in line 150 is required, but that none is required between the semicolon and statement number in line 160.

Continuation

A statement may be continued on as many lines as desired. Following the line number, the first nonblank character of each continuation line must be a plus sign. The plus sign only indicates continuation and is not a part of the statement.

Example

```
200 A(K)=A(I)+ROUTE(
210 + ARSIN'FUNCTION NAME
220 +, F(16), T)
```

These lines show how a statement begun on line 200 is continued on lines 210 and 220. The plus signs, as the first nonblank characters after the blank following the line numbers 210 and 220, indicate the continuation.

Statement Labels

A statement may or may not be labelled. The label may be a 1 to 5 digit number or a 1 to 30 character name. A colon must separate a label name and the statement it labels. A colon may but need not separate a label number and the statement. A statement may be given more than one label, and in this case, a colon must be written after all but the last statement number as well as after statement names. An empty statement may be labelled.

Example

```
280 ZERO:XB=XC=0 ; 3: XD=XE;4:TICK: XA=XA+1
290 LOC: 5 : B(XA)=0
300 FILL: REC: B(XA+J)=C(XA); 7:8 C(XA)=0
310 END: ; 6; END
```

These lines show various ways in which statements can be labelled. Line 280 shows that a statement may have a statement name, a statement number, or both. Line 290 shows statements multiply labelled with names and with numbers. Notice that the last or only statement number may have a colon after it, as in line 290, but need not, as in line 300. Line 310 shows empty statements labelled with a name and with a number.

Comments

An entire line may be devoted to a comment. Comment lines are indicated by entering any nonblank character as the first character after the line number. By this means, traditional FORTRAN comment lines, as well as source-included non-FORTRAN lines may be entered.

In addition to a comment line, a comment may be embedded anywhere within a line (even in the middle of a name if it suits your fancy). An embedded comment is introduced by an apostrophe and terminated by the next apostrophe, a semicolon, or the end of the line. An embedded comment may contain any characters, except a semicolon or apostrophe which would terminate the comment.

Examples

```
250+++++++
260$COMDEK, NOLIST
270'PROGRAM ACCEPTS THREE KINDS OF INPUT
```

These three lines illustrate lines taken to be comment lines because the first character after the line number is not a blank.

```
170 'FOR THE LAST THREE' J=J+1
180 IF (J-12)6 'ELSE FOR ALL; P=J*2
190 IF (I=P+1) 7, 7, 5 'END OF INITIALIZATION
```

These three lines illustrate the three ways in which an embedded comment may be terminated:

- By an apostrophe, as in line 170
- By a semicolon, as in line 180
- By an end of line, as in line 190

Note: If an apostrophe is the first character after the line number, the entire line is taken to be a comment.

```
120COMMON ARRAYS
130 COMMON A(R), B(16), C(6)
140 COMMON 'INDEPENDENT' L(R), 'DEPENDENT' M(16), 'CONSTANTS' N(6)
```

Because the first character after the line number in line 120 is not a blank, the entire line is taken to be a comment. The blank after the line number, in line 130, indicates that the line is not a comment. However, a comment may be embedded in the line by using apostrophe, as in line 140.

Control Information

Information which directs the compilation or execution of a program is entered in control lines. In Mark I FORTRAN, a control line is indicated by writing a dollar-sign preceding the control word as the first nonblank character after the line number.

Example

```
230 $FILE STA, ROUTE, SALES
240  $OPT SIZE
```

These lines illustrate how control information is entered. The $-preceded words in lines 230 and 240 are interpreted as control words.

Note: If the first character after the line number is a dollar sign, the entire line is taken to be a comment.

Sample Program

An example of a program using some of the unconventional features of Mark I FORTRAN is shown below. Following the example, notes describe the features illustrated.

Example

```
100'PROGRAM PROVIDES ROOTS OF QUADRATIC IF REAL
110 PRINT"PROVIDE COEFFICIENTS"
120 INPUT,A,B,C
130 PRINT"FOR COEFFICIENTS:",A,B,C;PRINT"ROOTS ARE:",
140 IF(DISCRIMINANT=B↑2-4*A*C)IMAGINARY
150 PRINT,(-B+(X=SQRT(DISCRIMINANT)))/(Y=A+A),
160 + -(B+X)/Y'SKIP AROUND; GOTO DONE
170 IMAGINARY:PRINT"IMAGINARY"
180 DONE'EMPTY STMT.':
```

Notes

1. Line 100 shows a comment line. The character after the line number is not a blank, so that the entire line is taken to be a comment.

2. Line 110 shows how a quotation is transmitted to the terminal.

3. Line 120 shows how terminal input is done. Unformatted input is indicated by omitting a format reference. Notice that the comma normally written after the format reference is retained.

4. Line 130 shows two statements separated by a semicolon on the same line. The extra comma at the end of the second output list suppresses slewing.

5. Line 140 shows an expression with an equal sign. Notice that it is a mixed mode expression. The integer 4 is automatically changed to a real 4.0. Notice also that long names may be used (as long as 30 characters). Only one label reference needs to be given, as it is in the example; or two or three may be given. Omitted references are assumed to be to the next statement. Labels may be either names or numbers.

6. Line 150 shows another terminal output statement. Since the format reference is omitted, standard output format is used. Expressions may be included in the output list, and expressions may use embedded equal signs. In this example X and Y name partial expressions so that the second item in the list can be computed and written without repeating the explicit definitions of X and Y.

7. Line 160 shows how a plus sign at the beginning of the line is used to indicate a continuation of the previous line.

8. Line 170 shows how a statement label name is separated by a colon from the statement.

9. Line 180 shows a labeled empty statement. Notice that neither an END nor a STOP statement is required. An apostrophe enclosed comment is shown in this line. An apostrophe introduced comment closed by a semicolon is shown in line 160.

3. ELEMENTS OF THE FORTRAN LANGUAGE

GENERAL DESCRIPTION

A program written in FORTRAN language consists of statements. These may either be arithmetic statements which resemble mathematical formulas or control, declaration, and input/output statements. All of these statements may be composed of the following kinds of elements:

- Constants
- Variable, array, or subprogram names
- Arithmetic operators
- Statement labels
- Punctuation

This chapter describes these elements and provides the rules to be observed in using each of them. Subsequent chapters describe the different kinds of statements in which these elements are used.

Use of Blanks

In FORTRAN statements, blanks are not used for punctuation. Words need not be separated by blanks, and the characters in a word may be written with blanks between them. In Mark I FORTRAN, aside from the blank required after the line number, statements may be written with or without blanks and any blanks are simply ignored.

Reals and Integers

Reals are distinguished from integers in FORTRAN. An integer is a real, or course, but a variable that has only integer values is distinguished from one that may have fractional (or large) values so that the integer can be handled specially. The computer can calculate faster with integers and in Mark I FORTRAN an integer requires half the space to store as a real. Another term for "real" is "floating-point variable."

A feature of Mark I FORTRAN is that a user unconcerned with the space required for his program or the time required to execute it may use only reals, without distinguishing those variables that can logically have integer values. Ordinarily, a user will be concerned with how much space his program requires and how much time it takes; he will want to distinguish integers from reals.

In Time-Sharing FORTRAN, a signed integer must remain within the range:

$-524287 \leq \text{integer} \leq 524287$

If the integer does not remain within this range, the variable should be handled as a real. Reals can be handled which have a magnitude within the range:

$.863616852 \times 10^{-77} \leq |\text{real}| \leq .578960444 \times 10$

CONSTANTS

Three kinds of constants can be represented:

- Decimals (both integer and real)
- Octal integers
- Quoted characters

Decimal Numbers

A decimal number may be written with (in this order):

- A sign (plus (+) or minus (-))
- An integral part
- A fractional part
- A ten's exponent

If no sign is written, a plus sign is assumed. Either an integral or fractional part, or both, must be written. The fractional part is introduced by a period (.) to indicate the decimal point.

The ten's exponent, if any, is introduced by the letter E or a dollar sign ($). A signed integer is written after the introductory character; if no sign is given a plus sign is assumed. The signed integer is the power of ten by which the preceding basic value is to be multiplied. An integral or fractional part, or both, must precede the multiplicative factor.

The decimal representation is interpreted as a real if any of the following exist:

- There is a fractional part
- There is a multiplicative factor
- The integral part is > 524287 in magnitude

The entire representation is restricted to 30 characters, but the integral part, fractional part, and multiplicative factor may each contain any number of digits subject to this restriction. If a number is represented that is larger in magnitude than can be handled, the largest real that can be handled is used. If the fractional part has more than 10 digits, the maximum that can be handled, the low order digits are ignored. (Leading zeros are not counted.)

Examples

```
      47          (integer)
      +1.         (real)
        .5        (real)
      -0.500      (real)
750000            (real, too large for integer)
      5$17        (real)
        .3E-15    (real)
       1.537E3    (real)
      -2          (integer)
        .01E+10   (real)
```

Octal Integers

From one to seven octal digits (0,1, ..., 7) may be used to represent an unsigned integer. The integer must be within the range:

$0 \leq \text{octal integer} \leq 3777777_8$

A slash (/) precedes the representation. If the number represented has more than seven digits or contains either an 8 or 9, the constant is marked as erroneous.

Quoted Characters

From one to three characters may be enclosed in quotation marks and used in expressions. If more than three characters are quoted in an expression, the constant is marked as erroneous. Use of longer quotations is restricted to output statements and formats.

When fewer than three characters are enclosed in quotation marks, the characters are left-justified in a machine word and remaining space is filled with code bits to represent blanks. (This is the same way that values transmitted using the A3 format are stored. Refer to page 83.)

Examples (Refer to "Character Set," page 112.)

```
"ABC"          is stored as 0212223₈
"AB"           is stored as 0212260₈
"A"            is stored as 0216060₈
```

Character constants are treated as integers in calculations. If the calculation involves a real, the character constant will be converted to a real which will usually not be useful.

NAMES

The user assigns names to arrays, subprograms, and variables. The following rules must be observed in constructing these names.

Spelling Rules

1. Names may be composed from letters, digits, and the dollar sign.

2. The first character must be a letter.

3. As few as one character and as many as 30 characters may be used. Note: If blanks are included in the name they are ignored and the name is considered identical to the same string of characters without blanks.

4. When the name follows a statement word (for example, DIMENSION or SUBROUTINE) without intervening punctuation, the characters in the statement word plus those in the name must not exceed 30 characters.

5. To ensure that a subscripted name is not taken for a statement word, array names whose first letters spell the following statement words should be avoided:

CALL	EXTERNAL
COMMON	FUNCTION
DIMENSION	INTEGER
ENTRY	REAL
EQUIVALENCE	SUBROUTINE

   For the same reason array names identical with the following statement words should be avoided:

FORMAT	READ
GOTO	WRITE
IF	

   Moreover, a variable name should be avoided of the form:

   DO nn...naa...a,

   where n's represent digits and a's represent letters.

6. A misspelled array name may result in the diagnostic UNDEF (see page 100.)

Indication of Integer or Real Mode

The mode of an element can be indicated in one of three ways:

1. The mode can be implied real or integer by the first letter of the name. If the first letter is I, J, K, L, M, or N, the name implies that the element is integer. Otherwise the name implies that the element is real.

2. The implied mode may be overridden by a declared mode. This is done by mentioning the element in an INTEGER or REAL statement before any other use of the element in the program. The declared mode is then used instead of the implied mode.

3. The implied mode may also be overridden by a control option which specifies that all elements for which no mode has been declared are to be interpreted as of the same mode--all integer or all real.

For any name, the declared mode always takes precedence over the implied or assumed modes. (Naturally, a declared mode may confirm as well as override an implied or assumed mode.)

Examples:

```
REAL  IMAGE, MANDATORY, OFFICE
INTEGER  FIRST, SECOND, LAST
```

The names in these mode declarations may be variable, array, or external subprogram names. If array names, they may have dimensioning information appended.

Array Names

A name is indicated as referring to an array by mentioning it first in one of the following kinds of statements:

- DIMENSION
- COMMON
- INTEGER
- REAL

Appended to the mention is a parenthesized list of positive integers which imply the dimension of the array and specify the size of each dimension.

If no parenthesized list follows the name in REAL or INTEGER statements, the name is assumed to refer to a variable or subprogram, not an array.

The number of dimensions is indicated by the number of positive integers in the parenthesized list:

- One number in the list indicates one dimension
- Two numbers in the list indicate two dimensions and so on

As few as one number and as many as 15 numbers may be given to indicate 1-dimension, 2-dimension, ..., 15-dimension arrays. If there is more than one number, each is separated from the next by a comma.

The size of each dimension is given by a number. The first number indicates the number of rows, the second indicates the number of columns, and so on. The product of all dimension sizes for one array name must not exceed 8191.

Note: Because an integer array of 6000 elements takes more space than is available for the entire program, smaller arrays (much smaller if there are many arrays) have to be used if the program is to be executed. The restriction of a single array to 8191 elements is required even if the program is only to be trial-compiled for use on another computer.

Examples:

```
DIMENSION  A(3), B(100), C35(12,12), ANTHRAX (3,5,1)
COMMON SPECIAL (10,3,2,1)
```

When the dimension specifications of one array are the same as those of another, the second array may be written with a quote mark in place of its dimension specification, to indicate ditto.

Example:

```
COMMON  A(20,20), B", C"    is the same as
COMMON A(20,20), B(20,20), C(20,20)
```

Every element of an array has a value of zero before execution of the program begins.

Subprogram Names

A name is indicated as referring to a subprogram by any of the following:

- Its use in a function call
- Its use after the word CALL in a CALL statement
- Its mention in an EXTERNAL statement
- Its use in a subprogram definition statement

A function call is an occurrence in an expression of a name which is suffixed by a parenthesized list of arguments and was not previously declared to be an array. (Parentheses may possibly be empty.)

Examples

```
J=MAGI (3): S=SUMP()                 function calls
CALL TOST; CALL ROSTER(3)            call statements
EXTERNAL ANTON, BUXT                 external statement
SUBROUTINE KROG                      external subprogram definition
PUN(X,Y) = X*SIN(Y) + Y*SIN(X)       internal function definition
```

Note that, in the first example, failure to first mention an array name in an array declaration may cause a subscripted reference to the array to be interpreted as a function call. In the last example, such failure to mention may cause the array name to appear as an internal function definition.

Variable Names

A name is indicated as referring to a variable if it is neither declared as an array nor used as a subprogram name or statement label.

Like array elements, the initial value of every variable is zero.

ARITHMETIC OPERATORS

Characters Used

The table below lists the expressions used to indicate the specified operation.

Operator	Operation Specified
=	Assignment
+	Addition
-	Subtraction or negation
*	Multiplication
/	Division
** or ↑	Exponentiation

Examples

```
-x                  negation of x
x+y                 addition of x and y
x-y                 subtraction of y from x
x*y                 multiplication of x and y
x/y                 division of x by y
x↑y  x**y           x raised to the y power

x=y                 assign the value of y to x
```

In this list, the operands x and y may be constants, variable names, array elements, function calls, or expressions involving these operands and the operators in the list. In the last entry in the list, however, the x in x=y may designate only a variable, an array element, or an internal defined function.

The -x indicates a unary minus. This is treated by the computer as though the expression 0.0 -x had been used.

Priority of Application

When multi-operator expressions are written, parentheses are used, as they are in mathematical formulas, to indicate to which operands the operators apply. For example:

x*(y+z) means multiplication of x and the sum of y and z
(x*y)+z means addition of z and the product of x and y

In the absence of parentheses, operators are applied according to the priority of application each has. These priorities are:

Highest:	x↑y, x**y	exponentiation
	x*y, x/y	multiplication or division
	x+y, x-y	addition or subtraction
	-x	negation on same level as addition and subtraction
Lowest:	x=y	assignment

The rule is: If the priority of the present operator is higher than that of the previous operator, the present operator is applied first, but if the priority of the present operator is the same as or lower than that of the previous operator, then the previous operator is applied first. An exception is that multiple assignments are done from right to left.

Examples:

A=B/C+D	means	A=((B/C) + D)
A=-B*C	means	A=-(B*C)
A=B=C+D	means	A=(B=(C+D))
A=B+C-D	means	A=((B+C)-D)
A=B↑C↑D	means	A=((B↑C)↑D)
A=-B↑C	means	A=(-(B↑C))

Expressions

An expression consists of a sequence of operands and operators. The expression is evaluated by applying the operators to the operands according to the rules discussed under "Priority of Application." If the operands are all real, only real operations are performed. If all operands are integer, only integer operations are performed. When both integer and real operands occur within an expression, not only are both integer and real operations performed, but also a conversion operation from one mode to the other is required. In Mark I FORTRAN, the conversion is automatically provided.

If a real operand is assigned to an integer operand, the real is truncated to an integer, and any fractional part is ignored. For example:

3.5	is truncated to 3,
0.57	is truncated to 0,
-8.7	is truncated to -8,
-0.61	is truncated to 0.

If the integer part of the real is too large to express as an integer (greater than 524287 in magnitude), a substitution notice is printed, and 524287 is then used as the magnitude.

When there are multiple assignments, the mode of the operand to be assigned is adjusted to the mode of the operand to which it is assigned.

Example:

```
A = I = B = 3.5 is the same as
B = 3.5; I = 3; A = 3.0
```

Except for assignment of a real to an integer, for all operations involving an integer and real operand, the integer is first expressed in a form suitable for real calculation. In Mark I FORTRAN, this conversion to real form is performed automatically, without user specification.

Exponentiation

Exponentiation is performed in three different ways, depending on the modes of the operands:

integer ** integer	successive multiplication
real ** integer	successive multiplication
real ** real	base ** exponent = EXP (exponent * LOG(base))

When the exponent is a large integer, and the base is a real, expressing exponentiation as "real**real" provides greater accuracy at the expense of speed.

STATEMENT LABELS

Definition

A statement must be labeled if it is to be referenced. The label is written at the beginning of the statement. In FORTRAN, the statement label may be a non-negative integer. In Mark I FORTRAN, the following range must be used:

$0 \leq$ statement number $< 100{,}000$

A statement number should not be confused with the number of the line on which the statement is written.

In addition to statement label numbers, Mark I FORTRAN allows the use of statement label names. The following rules apply to statement label names:

1. The name is suffixed with a colon to indicate that it is a statement label.
2. Like a statement number, the name is written at the beginning of the statement.
3. The name is subject to the same spelling rules as a variable or array name.
4. A statement label name need not differ from names used elsewhere in the program except that a label variable or format array should not be given the same name as a statement label.

In Mark I FORTRAN a statement may be labeled with more than one label. Empty statements may be labeled.

A statement number may be suffixed with a colon and must be colon suffixed if another label follows it.

Examples:

```
47   A=B+1
47:  A=B+1
47:  TICK:  A=B+1
TICK: 47  A=B+1
TICK: 47:  A=B+1
SIX: ; 6 ;   (these are labelled empty statements)
```

Statement Label Reference

Reference may be made to statements only in control and input/output statements. The reference is made by mentioning the name or number by which the statement is labeled. Naturally, if a statement is referred to, there must be one (and only one) statement with that label in the same subprogram as the reference.

In most statements, the statement reference is separated from other parts of the statement by a comma. In two statements (DO and ASSIGN... TO...) which do not require a comma after a statement label number, Mark I FORTRAN requires a comma after a statement label name. For uniformity, a comma is allowed after a statement label number. The list below illustrates all possible contexts.

```
DO 6 I = 1, 10
DO 6, I = 1, 10
DO SIX ,I=1, 10
ASSIGN 25 TO J
ASSIGN 25, TO J
ASSIGN QUARTER , TO J
IF (A-B) 3, 4, 5
IF (A-B) LESS, EQUAL, MORE
GO TO 15
GO TO QUINCE
GO TO (1, TWO, 3, FOUR) , N
READ 8, A
READ EIGHT , A
```

Note that a comma is required after a statement name whenever more of the statement follows the statement name.

In an input/output statement, as illustrated in the last example, the name given as a statement label could be interpreted in one of two ways. It could be taken as the name of an array which holds the format to be used, or it could be taken as the name of the statement in which the format is given. A name is interpreted as an array name only if it has previously been declared to be an array.

Label Variables

A label variable is declared by means of an ASSIGN...TO...statement. The variable named after the word "TO" is a label variable.

The traditional use of a label variable is in a GO TO statement. It is used to direct control to the statement whose label was most recently assigned to the label variable. For example,

```
GO TO QUINCE
```

If the most recently executed ASSIGN...TO QUINCE statement was ASSIGN 6 TO QUINCE, the above example would direct control to the statement labeled 6. However, if the statement ASSIGN ELSE, TO QUINCE was more recently executed, the above example would direct control to the statement labeled ELSE.

In addition, a label variable may be referred to anywhere a statement label may be, with two exceptions. One exception is the DO statement in which the label reference obviously must be to a statement. The format statement referenced by the ASSIGN statement must be located prior to the ASSIGN statement.

A name used as a label variable must not also be used as a statement label. Such use would provide multiple definitions of the same name. However, label variable names need not differ from names used for arrays, variables, and subprograms.

Examples:

```
6 FORMAT (10.4)
ASSIGN 6 TO FORM
READ FORM, A

ASSIGN ANTI, TO MINUS
IF (A) MINUS

ASSIGN 6 TO LVA
ASSIGN LVA, TO MINUS
```

4. SUBSCRIPTING

ARRAY ELEMENT REFERENCE

A particular element of an array can be referenced by means of a subscript. The name of the array is written followed by a parenthesized list of subscript expressions. In this list there is one expression for each dimension. The integer value of the first expression gives the ordinal of the row in which the referenced element is located, the second, the column, and so on. Counting of rows, columns, and so on always starts with one.

Any expression may be used as a subscript expression. In Mark I FORTRAN, expressions involving subscripting or requiring real calculations are acceptable. (Traditional FORTRAN restricts the expressions to linear combinations of integers.)

MISSING SUBSCRIPT

When a subscript expression is not given for a dimension, a value of one is assumed. Thus, with a dimension declaration--

```
DIMENSION A(3,4,2,2), B(4,4,4)
A(J,K)+B means A(J,K,1,1) + B(1,1,1)
A(L+2/N, R(P))*B(3,3) means A(L+2/N, R(P),1,1)*B(3,3,1)
B/A means B(1,1,1) / A(1,1,1,1)
```

In some contexts, missing subscript expressions are not assumed to be 1. When only the name of the array appears as an item in an input/output or argument list, or as a format specification, the entire array (not just its first element) is assumed to be referenced.

SUBSCRIPT RESTRICTIONS

In Mark I FORTRAN it is considered an error:

1. To give more subscript expressions than there are dimensions declared for an array

2. To give any subscript expressions for a variable

3. For an array of three or more dimensions, to give a subscript expression whose value is:

   nonpositive, or
   greater than the size declared maximum for the corresponding dimension

4. To give subscript expressions whose values, taken together with the dimensioning information provided for the array, would cause an assignment to be made outside all arrays.

SUBSCRIPT CHECKING

The third subscript restriction together with the assumption of one for missing subscript expressions provides the basis for two degrees of subscript validation. A single or double dimensioned array can be dimensioned to be three, with the extra dimension having the size of one. Then, every subscripted reference to this array is subjected to a subscript validation. This validation confirms two things. First, it confirms that the expressions are always positive, and second, that they are not greater than the maximum declared for the corresponding dimension.

An optional control statement, $OPT SS, is also available that automatically treats all single and double dimensioned arrays as though they had been extended to three dimensions. By means of this treatment, all subscripting can be subjected to validation. The $OPT SS must precede the DIMENSION statement containing the variable for which subscript checking is desired.

The fourth subscript restriction is detected principally to provide protection to other programs sharing the computer at the same time. In addition, it can be useful for program checkout, by detecting erroneous subscripted assignment.

SUBSCRIPT TRUNCATION

A subscript value for an integer array is modulo 8192; for a real array 4096 (since the subscript is doubled to select a two-word array) element. Because of this truncation, negative subscripting does not work. For example, a subscript of -2, when used modulo 8192, is changed to 8190 (because negative values are held in complemented form).

5. ARITHMETIC STATEMENT

An arithmetic statement is written like a formula: a name is written first, followed by an equal sign and an expression.

name = expression

In FORTRAN, the arithmetic statement indicates that the expression is to be evaluated and this value assigned to the name.

The name may be that of a variable or an array element (subscripted array name). If an unsubscripted array name is given on the left of the equal sign, it is assumed to refer to the first element of the array.

In Mark I FORTRAN, more than one name may be written to the left of the equal sign. Each name is separated from the other by an equal sign, to indicate that the value of the expression is to be assigned to each of the names.

name = name = ... name = expression

If the name and expression are of different modes, the value of the expression is converted to the mode of the name upon assignment.

Examples

```
A = 2.
A = A+2
L(I) = (A+2.)/SIN(A-2.)
C(2*L(I)+K) = (T=A+2.)/SIN(T)
F(I) = C = L = -1/S
CIRCUMFERENCE = PI*DIAMETER
```

6. DECLARATIONS

A declaration provides information needed to allocate storage for an array or variable, to indicate what names are used for arrays or external subprograms, and to indicate mode.

The declaration must contain the first mention of the name in the subprogram.

Sometimes the information about a name is given in more than one declaration statement. In this case, no nondeclarative statements must intervene.

Examples

```
DIMENSION A(5)
INTEGER A
COMMON A

        is a permissible, if round-about, way of making
        declarations about the array A, but


DIMENSION A(5)
J=J+1
INTEGER A
COMMON A

        is not permissible since a nondeclarative statement
        intervenes.
```

If a declaration provides dimension information for an array, no other declaration in that external subprogram may also provide dimension information for the same array. If a declaration provides mode information for a variable or array, no other declaration may provide different mode information.

MODE DECLARATION

The mode qualifiers, INTEGER or REAL, may be used to declare the following:

- Modes of program elements named in any of the declaration statements
- Modes of the statements introducing function definitions to declare the mode of the result returned by the function.

The list below shows all possible contexts in which they may be used, and to what the declared mode applies.

Declarations (declared mode applies to program elements named)

REAL INTEGER	array, variable*
REAL INTEGER	DIMENSION array, variable
REAL INTEGER	COMMON array, variable
REAL INTEGER	EXTERNAL external subprogram or entry, intrinsic function
REAL INTEGER	EQUIVALENCE array, variable

Subprogram Definition (declared mode applies to subprogram being defined)

REAL INTEGER	FUNCTION external*
REAL INTEGER	, arithmetic statement function, internal function

A declared mode confirms or overrides a mode implied by the first letter in the name, or assumed because of a single mode option. The mode declaration is required only to override an implied or assumed mode.

Mode declarations are not required for intrinsic functions.

If an external function mode is different than as assumed or implied mode, a mode qualifier is required in the FUNCTION statement that begins the definition of the function. This definition must precede the reference to the function.

COMMON DECLARATION

An array or variable is declared to use common storage to enable it to share storage with arrays or variables declared in common storage in another external subprogram.

---

* In many FORTRAN systems these are the only declarations allowed.

The sharing is done for one of two reasons:

1. To conserve storage. If an array declared in one subprogram is required only when an array declared in another is not required, space for only one is needed when they share storage.

2. To share the values stored in shared storage. By declaring arrays used by more than one external subprogram to share storage, the values of these arrays can be referenced in each subprogram.

Reference in each subprogram is made to the name given the shared storage in that subprogram. The names may be the same in each subprogram but it is the sharing of storage, not the use of the same name, that provides for sharing of values.

The basis for this sharing is provided by starting allocation of common storage at the same space in each external subprogram. Common storage is started at the end of the available space and is allocated backward. For example:

An array A in one subprogram is allocated the last 100 spaces.
An array B in a second subprogram is allocated the last 100 spaces.
Therefore, the arrays A and B are allocated the same space.

To declare that common storage is to be allocated for a variable or array, it is named in a COMMON statement. The first named is allocated the last space, or spaces; the second is allocated the next to last space, or spaces, and so on.

IN MARK I FORTRAN, ALL COMMON STORAGE REQUIRED FOR THE ENTIRE PROGRAM MUST BE DECLARED IN THE FIRST SUBPROGRAM or in the first in which any common or dimensioned storage is declared, to ensure that adequate space for all common storage is reserved.

When two arrays are to be allocated the same space as used by a single array in another subprogram, the array to share storage with the last elements is named first and the one to share storage with the first elements is named second.

Example

```
COMMON A(4)  in one external subprogram, and
COMMON C(2), B(2) in another subprogram causes
      A(1) and B(1), A(2) and B(2)
      A(3) and C(1), A(4) and C(2) to share storage.
```

The number of spaces required depends on the mode and dimension of the variable or array. An integer variable requires one space and a real variable, two. An integer array requires as many spaces as the product of its dimension sizes, and a real array, twice the product of its dimension sizes.

Example:

COMMON A(12), B, C(3,2), K, J(4,3,2,2)

allocates	first 24 spaces for A	(twice 12)
	next 2 spaces for B	(twice 1)
	next 12 spaces for C	(twice 3*2)
	next 1 space for K	(once 1)
	next 48 spaces for J	(once 4*3*2*2)

If in another external subprogram, the statement below is given:

COMMON I (10,4),N(12)
it allocates the
first 40 spaces for I (once 10*4)
next 12 spaces for N (once 12)

Together these common declarations cause:

I to share storage with A, B, C, K, J(4,3,2,2)
N to share storage with J(4,3,1,2) through J(3,3,2,2)

As this example shows, causing an array to share storage with another with different dimensions or dimension sizes, requires knowledge of how elements of an array are stored.

ARRAY STORAGE

Storage of arrays is "column-wise." That is, the first column is stored first, then the second if there is one, and so on. After all columns of the first plane are stored, any columns of the second plane are stored in the same order, and so on.

Example

J(4, 3, 2, 2) is stored

J(1, 1, 1, 1)	first column, first plane, first hyperplane
J(2, 1, 1, 1)	
J(3, 1, 1, 1)	
J(4, 1, 1, 1)	
J(1, 2, 1, 1)	second column, first plane, first hyperplane
-	
J(1, 3, 1, 1)	third column, first plane, first hyperplane
-	
J(1, 1, 2, 1)	first column, second plane, first hyperplane
-	
J(1, 2, 2, 1)	second column, second plane, first hyperplane
-	
J(1, 3, 2, 1)	third column, second plane, first hyperplane
-	

repeated for the second hyperplane

DIMENSION DECLARATION

An array is declared in dimensioned storage to provide it unique storage. This storage is not shared with arrays or variables declared in other external subprograms.

The basis for unique storage is sequential allocation of dimensioned storage. The first space, after the last one used in the previous subprogram for dimensioned storage, is used for dimensioned storage in the present subprogram. Thus, if an array A is declared in dimensioned storage in one subprogram, and an array B is declared in the next, B is allocated space immediately after the spaces allocated for A.

IN MARK I FORTRAN, ALL COMMON STORAGE REQUIRED FOR THE ENTIRE PROGRAM MUST BE DECLARED BEFORE DIMENSIONED STORAGE CAN BE ALLOCATED.

An array is declared in dimensioned storage by naming it in a DIMENSION, INTEGER or REAL statement. The number of spaces required and the way an array is stored are the same for dimensioned as for common storage.

EQUIVALENCE DECLARATION

The equivalence declaration is used to cause variables or arrays to share storage with other arrays or variables in the same external subprogram. Unlike common declarations which are used to cause variables or arrays in different subprograms to share storage, the equivalence declaration is used to cause those in the same subprogram to share storage.

There are two reasons for such sharing of storage:

1. To conserve storage. If the values of an array are not required at the same time as those of another array, in the same subprogram, declaring them as equivalent causes them to use the same storage.

2. To cause two or more names to refer to the same variable or array. If two or more names have, by mistake, been used to refer to the same array or variable, declaring them as equivalent causes them to use the same storage. In this way, reference to any of the names refers to the storage referred to by any of the other names.

The variables or arrays to be declared equivalent are named within a parenthesized list in an EQUIVALENCE statement. Names in the list are separated by commas.

Example

```
EQUIVALENCE (BEGIN, START, INITIATE), (FINALIZE, TERMINATE)
```

If more than one equivalence is to be declared in the same statement, commas are used to separate the parenthesized lists which declare each equivalence.

Subscripted Equivalence

To indicate that only certain array elements are to share storage with other arrays, part-arrays, or variables named in the same equivalence declaration, these array names are subscripted with a single integer. The array elements that are included in the equivalence are those stored from the array element indicated by the subscript through the end of the array. (Refer to "Array Storage" on page 25.)

Example

```
EQUIVALENCE (A(10), B)
DIMENSION A(40),B(40)
```

The last 31 elements of the A array (A(10), A(11), ... A(40)) share storage with the first 31 elements of the B array (B(1), B(2), ..., B(31))

When negative or zero subscripts are given, reference is made to the array allocated storage before the array subscripted.

Example

```
EQUIVALENCE (E(15), G(1))
DIMENSION E(6,3), F(20), G(20)
```

The last 4 elements of the E array share storage with the first 4 elements of the G array.

```
F(1)
.
.
.
F(20)
E(1)
.
.
.
E(10) = E(4,2)
E(11) = E(5,2)
E(12) = E(1,3)
.
.
.
E(15) = E(3,3) = G(1)
E(16) = E(4,3) = G(2)
E(17) = E(5,3) = G(3)
E(18) = E(6,3) = G(4)
                 G(5)
                 .
                 .
                 .
                 G(20)
```

Notice that since the equivalence declaration in Mark I FORTRAN accepts only a single subscript, the two-dimension E array has to be subscripted as though it were a single dimension array.

Equivalence Errors

An array or variable declared to use common storage may also be declared to be equivalent to other arrays or variables. In Mark I FORTRAN, however, only one of the arrays or variable names in an equivalence can be declared to use common storage.

Examples

1.
```
COMMON A(20)
EQUIVALENCE (A,B)
```

2.
```
COMMON A(20), B(20)
EQUIVALENCE (A, B)
```

The second example is erroneous because in Mark I FORTRAN, as in FORTRAN IV, common storage is allocated without regard for equivalence declarations that affect it.

Similarly, an equivalence declaration may include only one name for which a previous equivalence declaration has caused storage to be allocated.

Example

```
DIMENSION R(10), S(10)
EQUIVALENCE (R,S), (R(2), S(4))
```

Obviously both equivalence declarations cannot be satisfied, so the second is marked erroneous.

A third kind of equivalence error occurs when both integer and real arrays or variables are declared in the same equivalence and one of them is also declared in common storage. If a real is made equivalent to an odd-numbered space, the equivalence is marked erroneous. (The hardware requires that the first word of real variables occupy even-numbered spaces.)

Examples

1.
```
INTEGER COMMON J(4), I(10)
REAL DIMENSION A(10)
EQUIVALENCE ( A(2), I(4) )
```

2.
```
INTEGER J(4), I(10)
REAL A(10)
EQUIVALENCE (A(2), I(4))
```

In the first example, a space would have to be left between the J and I arrays if I(4) were to be allocated an even-numbered space (common storage starts at an odd-numbered space.) Rather than do this, the equivalence is marked erroneous. In the second example, since J and I arrays are not declared in common, a storage space is left between them. In this way the element I(4) is allocated an even-numbered space if necessary.

EXTERNAL DECLARATION

If an external or intrinsic subprogram is used as an argument, it must be named in an external statement before it is used. This is required to identify the call argument as a subprogram name.

Examples

1.
```
EXTERNAL HUNCH
CALL DRAG (HUNCH)
```

2.
```
EXTERNAL HUNCH
CALL DRAG (HUNCH (3.) )
```

In the first example, the external declaration is required. In the second example, external declaration is not required because the name of the subprogram occurs in an expression used as a call argument and not as the call argument itself.

A name declared external:

- Must not also be declared common
- Must not be dimensioned
- Must not appear in an equivalence declaration.
- Must be either an external subprogram or entry name or an intrinsic function name (and not an internal function or entry name).

An internal function or entry name may be passed as an argument to another subprogram, but it must not be named in an external statement to identify it as a subprogram.

7. CONTROL STATEMENTS

Execution of a program begins with the first executable statement in the main program. Execution continues with each succeeding statement unless control is directed elsewhere by means of a control statement. The control statement may direct control to another statement in the same subprogram, to a statement in another subprogram, or to the operating system.

DIRECTING CONTROL TO A STATEMENT IN THE SAME SUBPROGRAM

To direct control to a statement in the same subprogram, the control statement refers to the label of the statement to which control is to be directed. A label variable may be referenced in place of the statement label, except in the DO statement.

GO TO Statements

There are two forms of the GO TO statement. In one, control is directed to the statement whose label is referenced, or whose label was last assigned to the label variable referenced in the GO TO statement.

Examples

```
GOTO 13
GO TO 0
GO TO EXTRA
GO TO WHEREVER, (2, EXTRA, 13)
```

In the last example, the parenthesized expression may contain labels that may be assigned to the label variable WHEREVER. If given, the list must be separated by a comma from the label variable name. In Mark I FORTRAN, the list is ignored and, in fact, WHEREVER may be the label of a statement. Statement label names and label variables are indistinguishable in a GO TO statement. In Mark I FORTRAN, EXTRA and WHEREVER may be either statement labels or label variables.

In the other form of the GO TO statement, control is directed to one of the statement labels (or label variables) listed inside parentheses in a GO TO statement. The statement label to which control is directed is determined by the value, when the statement is executed, of the variable named after the list. If the integer value of this variable is 1, control is directed to the first listed label reference. If the value is 2, control is directed to the second listed, and so on.

Examples

```
GOTO ( 12, LAST, KONLY, 15 ) AFTER
GOTO (M1, M2, MT3 ), M
GOTO (7, 3, 6, 9 ), RAST
GOTO ( 7, M1, LAST, 9, 3 ) TAG
```

The parenthesized list is written after the words "go to." The name of the variable whose value determines which label reference is used, may or may not be separated from the parenthesized list by a comma. The variable may be either an integer or real variable. If the value is real, it is truncated to an integer. If the value is negative, zero, or more than the number of label references in the list, execution stops.

ASSIGN Statement

The ASSIGN statement, while not in itself causing control to be directed to another statement, does establish the value of a label variable. A label is assigned to a label variable as follows. The label is written first, after the word "assign." Then the word "to" is written, followed by the name of the label variable.

Examples

```
ASSIGN 6 TO J
ASSIGN FORMAL, TO R
ASSIGN A5, TO A6
ASSIGN 1273, TO LIN6
```

In the second and third examples, a comma is required after the statement label name to separate it from the word TO. The comma may be used after a statement label number as in the last example, but this is not required.

The label variable may have either an implied real or integer name since it has no mode. Reference to a variable with the same name as a label variable in any context except a label reference context is not a reference to a label variable. In particular, the label variable cannot be used in an expression as a call argument or transmitted as output because none of these are label reference contexts.

IF STATEMENTS

The IF statement includes a parenthesized expression or arithmetic statement, the value of which at the time of execution determines which statement is next executed.

The IF statement includes a parenthesized expression or arithmetic statement, the value of which at the time of execution determines which statement is next executed.

- Negative
- Zero
- Positive

Examples

```
IF(A) 25,26,27
IF(A*SIN(B) ) AGAIN, EXCEPT, AGAIN
IF(J=K/3)3, NOW, 5
IF(L-"END") NEXT, DONE, NEXT
IF(B*B-4.*A*C) IMAG
IF(F+R/P) LOOP, LOOP
```

In the first example, control is directed to the statement labeled 25 if A is negative, to 26 if A is zero, or to 27 if A is positive. In the last two examples, fewer than three label references are listed after the expression. When only one is listed, execution continues with the next statement after the IF statement, if the expression value is either zero or positive. When two are listed, execution continues with the next statement if the expression value is positive.

IF(ENDFILE)STATEMENT. The IF(ENDFILE) statement in Mark I FORTRAN provides a way of testing whether an end-of-file condition was encountered in reading from or writing on a designated file. The end-of-file condition is described with other file statements. (Refer to "File Statements," page 60.)

One or two label references are given after the parentheses to indicate what statement is to be executed next if:

- There is not an end-of-file condition (Transfer is made to the first statement label.)
- There is an end-of-file condition (Transfer is made to the second statement label or to the next statement if the second statement label is missing.)

Examples

```
IF(ENDFILE 3) MORE
IF(ENDFILE J) LOOP, EOF
IF(ENDFILE) DATA, ALL
```

The file to be tested for an end-of-file condition is designated by writing after the word "endfile" an unsigned constant or variable whose integer value designates a file. When no file designation is given, as in the last example, file 0 (the temporary file) is designated. (Refer to "File Statements," page 60.)

IF (SENSESWITCH switch) and IF(SENSELIGHT light) Statements. These two statements, often provided in FORTRAN II systems, are not provided in Mark I FORTRAN. In many cases, naming a variable SENSESWITCH 3, for example, or SENSELIGHT 1, will provide both the ability to set a switch and test for its setting.

The variable should be assigned a negative value to indicate ON, or a zero or positive value to indicate OFF. For compatibility, a test of a senselight should, if the ON condition is met, set it OFF, but a test of a senseswitch should not reset the condition.

IF ACCUMULATOR OVERFLOW AND IF DIVIDE CHECK STATEMENTS. These statements are not provided in Mark I FORTRAN. In real calculations, overflow and divide check conditions are automatically sensed. A substitution value is used, and a notice of the substitution is transmitted to the terminal.

In integer calculations, overflows have the following result:

```
 524287 + 1  = -524287
-524287 - 1  = 0
```

Integer division by zero produces a quotient equal to zero.

DO Statement

SUMMARY. The DO statement is used to repeat one or more subsequent statements. Indicate the statements to be repeated by giving the label of the last statement: this statement together with all between it and the DO statement are to be repeated. The number of times to repeat the statements is indicated indirectly by giving an initial, final, and increment value for a control variable. The control variable is assigned the initial value. The first time the statements are executed, the initial value is used. After execution of the last statement, the control variable value is increased by the increment, and the increased value is tested against the final value. As long as it remains less than or equal to the final value, the statements are repeated. Repetition is achieved by directing control back to the first statement after the DO statement.

The five DO parameters are written after the word "do" in the order:

1. Label of last statement
2. Name of control variable
3. Initial value
4. Final value
5. Increment

These parameters are written with the punctuation shown in the model below. Underneath the model of the DO statement is a model composed of FORTRAN statement models which shows how the control variable is used to count repetitions.

```
      DO last, control variable = initial value, final value, increment
```

```
      control variable = initial value

first: . . .
       . . .
last:  . . .

      control variable = control variable + increment
      IF (control variable - final value) first, first
```

Examples

```
DO 16 I = 1, 10
DO ALL, L=K, J, 2
DO 25, X = 3.5, 17., .5
DO 14 R = 25, 100, .2
DO TAB, N = T,S
```

Notes

1. A comma must be given after a label name. A comma may be given after a label number.

2. A control variable may be either real or integer. It may be a dummy argument, but not an array element.

3. Final, initial, and increment values may be real or integer constants or variables. They may be dummy arguments, but not array elements or expressions.

4. An increment value may be omitted in which case an integer constant increment of 1 is assumed.

DO-END. The statements to be repeated consist of those between the DO statement and the statement whose label is referred to in the DO statement, including that labeled statement.

The label reference may be either a statement number or a name. If it is a name, it must be followed by a comma. If it is a number, it may be followed by a comma. The reference must be to a statement label, not a label variable. The statement so labeled must:

- Come after the DO statement
- Not precede the ending statement for another DO statement given after this one
- Not follow the ending statement for another DO statement given before this one

Examples

```
      DO 16
       ...
      DO 25
       ...
      DO 35
       ...                  ← 25 must not precede 35
      35'end of DO 35
       ...                  ← 25 must be between 35 and 16
      16'end of DO 16
                            ← 25 must not be after 16
```

If a statement is labeled with several DO-end labels, they may be in any order because they all refer to the same statement.

Example

```
      DO 40
       ...
      DO 41
       ...
      DO 42
       ...
    41: 42: 40
```

Increasing the control variable value and testing against the final value is not done until after execution of the last statement. Therefore, the statement labeled with the DO-end label should not be a control statement that causes control to be diverted from this testing. For this reason, the last statement should usually not be either of the following:

- GO TO
- RETURN

The last statement must be in the same subprogram as the DO statement. A statement like a SUBROUTINE or FUNCTION statement which would begin another subprogram must neither intervene before nor be used as the last statement. However, an END statement that marks the end of the subprogram containing the DO statement may be used as the DO end.

CONTINUE STATEMENT. The CONTINUE statement is often used as the statement labeled with a DO-end label. It performs no other purpose.

The CONTINUE statement is unnecessary in Mark I FORTRAN since labeled empty statements can be used. However, it may be used.

Example

```
DO 101 . . .
 . . .
101 CONTINUE
 . . .
DO 102
 . . .
102
```

DO-NAME. The control variable whose value is used to count repetitions is named after the DO-end label reference. The DO-name:

- May be an integer variable
- May be a real variable
- May be a dummy argument of either mode
- Must not be an array element

The DO-name must not be the same as one given in a prior DO statement which is not yet ended.

Example

```
DO 44 J
 . . .
DO 45 J   must not be used here since J is already in use as DO-name
 . . .
44 : 45
 . . .
DO 51 J
 . . .
51
 . . .
DO 52 J   may be used here since J is no longer in use as a DO-name
```

DO PARAMETER VALUES. In a DO statement, the initial, final, and increment values may be real or integer constants, variables, or dummy arguments. They must not be array elements or expressions.

Quoted constants may be used, but constants may not be written with preceding slash (to indicate octal) and neither constants nor variables may have a preceding minus sign (to indicate negation).

Considering the way the testing for the end of repetition is done, it can be seen that "backward" counting does not work. The final value must be greater than the initial value, otherwise the first test will show the control variable value to be greater than the final value and no repetition will occur.

When the final or increment values are variables, their values may be altered by statements being repeated. These new values will be used thereafter to determine whether to repeat.

CONTROL VARIABLE VALUE. When control leaves repeated statements because the increased control variable value is greater than the specified final value, the control variable has that increased value. When a control statement for example, an IF statement, is one of the repeated statements, control may leave the repeated statements by means of the control statement. In this case, the value of the control variable equals the result of the last increase.

Example

```
DO NEXT J, J = 1, 10
READ, A(J)
IF (A(J) ) M1 exit here means J has the value 1, 2..., 10 last assigned
NEXT J :
exit here means J = 11
```

The value of the control variable can be changed by means of statements included in the repeated statements or to which control is directed. For example, the statement whose label "M1" is given in the IF statement above might perform:

```
M1 : J = J - 1 ; GO TO NEXTJ
```

DIRECTING CONTROL TO STATEMENTS IN ANOTHER SUBPROGRAM

Function Call

A function call is used to direct control to an intrinsic, external, or internal function. The result returned by the function is then used in place of the function call.

Example

```
AX = AY * FAN (AZ)
```

The function FAN is called with the argument AZ. The result returned by that function is multiplied by AY and the product is assigned to AX.

A function call is recognized by the occurrence of a parenthesized list of arguments (possibly empty) appended to a name not declared to be an array.

Example

```
T = X(L)
S = J J ()
R = YAW (P, 5, L/V, A(J))
```

Note: Forgetting to declare an array may cause the array name to be mistaken for a function name, its subscript to be mistaken for an argument list, and the array element to be mistaken for a function call.

The name in a function call may be:

- An intrinsic function name
- An internal function name in the same subprogram
- An external function name
- A dummy argument
- An external subroutine name
- An entry name

When the name is not a function name, the result returned is equal to the value of the last expression evaluated before returning. The result is not the value of a result variable in the subprogram. When the name is an entry name, the result returned is that of the subprogram (external or internal function, subroutine, or main program) in which the entry occurs.

REPEATED VS RECURSIVE CALL. A function call may contain in its argument list a call to itself. The call in the argument list is made first to evaluate the expression in which it appears. Then the value of this expression is supplied as an argument in calling the function a second time. This is repeated use.

Example

```
T = LOG (LOG(X) )
The value of LOG (X) is first obtained and used as an argument in calling LOG again.
```

Recursive use of a function occurs when a function is called, either directly or indirectly, in statements used to define the function. In Mark I FORTRAN, recursive calling disables the mechanism by which control is returned so that control may cycle endlessly.

Example

```
FUNCTION CURSE (R)
X = CURSE (1.5)
CURSE = X + 1.
RETURN
```

In this example, the call to CURSE within the definition of CURSE sets the return mechanism to return control to the expression X = CURSE (0.). Now when the RETURN statement is executed, instead of control returning to a call to CURSE from outside the subprogram, control is returned to the expression X = CURSE (1.5).

CALL Statement

A CALL statement may be used to call either a function or a subroutine. Ordinarily, a CALL statement is used to call a subroutine which does not return a result. When a function is called with a CALL statement, the result it returns is not used, since there is no expression for the result value to affect. The call statement is written with the name of the subprogram called after the word "call." If there are arguments, they are written in a parenthesized list after the name.

Example

```
CALL FEN (S, A(5), B(L/2) )
CALL OUT
CALL B(N)
```

The name may be that of:

- An external subroutine
- An external function
- An internal function
- An entry

Recursive use of CALL should be avoided for the same reason that recursive function calls are avoided.

RETURN Statement

Execution of a RETURN statement in a subprogram causes control to return to the last call made to the subprogram. Execution of an END statement or the last statement in a subprogram has the same effect.

The RETURN statement serves no other purpose.

If a subprogram was called using an entry in it, return is to the call to the entry. Also, if the main program was called by using an entry, return is to the call to the entry.

The main program is first entered without any call. If the main program was not subsequently entered by calling an entry in it, execution of a RETURN statement or the END or last statement has the same effect as the execution of a STOP statement.

DIRECTING CONTROL TO THE OPERATING SYSTEM

STOP Statement

Execution of a STOP statement anywhere in the program halts execution of the program. A message is output to the terminal which gives the line number of the STOP statement and the time charged to the execution of the program in 1/6 seconds.

If the STOP statement is written with a variable name or a constant after the word STOP, its character value is transmitted to the terminal also. If nothing is written after the word STOP, blanks are transmitted. If the stop was the result of the execution of a return in a main program, END is transmitted.

Examples

```
STOP                        ("   " is output)

STOP "UGH"                  ("UGH" is output)

STOP 3                      ("3" is output)

STOP 33                     ("J" is output since 33(10) = 41(8) = "J")

STOP /33                    ("." is output since 33(8) = ".")

STOP V                      (the value of V, truncated to an integer, is output as
                             a character)
END or last statement ("END" is output)
   in main program
```

PAUSE Statement

Execution of a PAUSE statement anywhere in the program causes execution of the program to be suspended until the user resumes execution. A line is transmitted to the terminal consisting of the word PAUSE and the line number of the line containing the statement. A second line is transmitted containing a question mark. If the user transmits a carriage return after the second line, execution of the program is resumed with the statement after the PAUSE statement. If the user transmits the word STOP, execution stops.

The first output line also contains a legend derived from what is written after the word PAUSE. This legend is derived as for the STOP statement. The legend is either the character value of the variable name or constant given, or blanks if nothing is given.

If a variable name is written after the word PAUSE, any characters transmitted after the question mark and before the carriage return are assigned to the variable as a character value. If a negative value is transmitted, it is transmitted as a numeric value.

8. SUBPROGRAMS

INTRODUCTION

Subprograms are useful when a specific calculation or set of calculations needs to be repeated during the running of a program. Once defined, a subprogram may be called several times in the same program with different parameters. Repetitious coding and unnecessary demands on core storage are thereby avoided.

DUMMY ARGUMENTS

Both internal and external functions and subroutines may be defined in terms of dummy arguments. The dummy (that is formal) arguments are written in a list after the name of the subprogram in the definition of the subprogram. There may be as few as none or as many as 14 dummy arguments.

The first listed dummy argument is the name by which references are made within the subprogram to the first actual argument with which the subprogram is called. The second is the name by which the second call argument is referred, and so on. For example, when a subprogram named CRUST is defined with a dummy argument list, CRUST (X,Y):

The call	means reference within CRUST to	is a reference to
CRUST(A+B, 3.2)	X Y	A+B 3.2
CRUST(C(I), D)	X Y	C(I) D
CRUST(SIN(L),EXP)	X Y	SIN(L) EXP
CRUST(C, "E")	X Y	C "E"
CRUST(C(F+D/2), 2)	X Y	C(F+D/2) 2

Dummy arguments can represent the following program elements:

- An expression, for example, A+B or SIN(L)
- A constant, for example, 3.2 or "E" or 2
- An array element, for example, C(I) or C(F+D/2)
- A variable, for example, D
- An array, for example, C
- A subprogram, for example, EXP

When a dummy argument represents an expression, it stands for the value of the expression as it has been calculated at the time the subprogram is called. Similarly, when a dummy argument represents an array element, it stands for that element the evaluation of the subscript expression has chosen at the time the subprogram is called. When a dummy argument represents an array, the dummy argument should be declared in the subprogram to be an array with the same number of dimension sizes as contained in the array declaration in the calling program.

A dummy argument shoud be named in a –

- REAL or INTEGER statement: to declare a mode if the mode of the corresponding call argument is different than an assumed mode or one implied by the dummy argument name.

- DIMENSION or mode statement: to declare the number of dimensions and the size of each dimension, if the corresponding call arguments are arrays.

A dummy argument may be named in an –

- EXTERNAL statement: to declare the corresponding call arguments are subprogram names; intinsic, internal, or external. (This information is not required, but if given is used to confirm that the dummy argument name is not used except as a subprogram in the subprogram definition statements.)

- COMMON or EQUIVALENCE statement: to indicate that the corresponding call arguments are in common storage or equivalent to something. (This information is actually ignored, but because it is sometimes given mistakenly, it is allowed in Mark I FORTRAN.)

Any declarations in which dummy arguments are named must be given in statements following the statement that contains the dummy argument list. This list is the one in which the dummy argument is first named. No statements except declaration statements may intervene.

In every call to a subprogram the following rules should be observed:

- The number of arguments in the call should be the same as the number of dummy arguments in the subprogram.

- The mode of each call argument should match the mode of the corresponding dummy argument.

- The type of call argument should conform to the way the corresponding dummy argument is used in the subprogram definition statements.

This conformity is as follows:

When the dummy argument is...	The corresponding call argument should be...
used as a value but not assigned one	constant, expression, array element, or variable
assigned a value, perhaps in addition to being used as a value	array element or variable

assigned a set of values perhaps in addition to being used as a set of values, and declared an array	array
used as the name of a subprogram	subprogram; intrinsic, internal, or external
used as a call argument in calling another subprogram	any of the above

No check, however, is made to see that the number of call arguments matches the number of dummy arguments. When a subprogram is called with more arguments than the dummy arguments that have been defined for it, the extra call arguments are provided by the calling program, but they are ignored by the called subprogram. When called with fewer arguments, the dummy arguments for which no corresponding call arguments have been provided refer to the instructions after the call.

Furthermore, no check is made to see that a call argument has the same mode as the dummy argument, or that it is the kind of program element implied by use of the dummy argument in the subprogram. A real variable can be provided by the calling program for a dummy argument whose mode is integer. Or, an integer constant can be provided for a dummy argument declared to be a real array. The consequences of such mismatching can be foreseen and, thus, perhaps used to some advantage by a sophisticated user, but clearly this is not recommended.

In Mark I FORTRAN, three kinds of checks are made to protect programs which share the machine at the same time. When a value is assigned to a subscripted dummy argument name, by either an input or assignment statement, the assignment is checked to ensure that it is to a space within the dimensioned and common storage for the entire program. An assignment to an unsubscripted dummy name is also checked to ensure that the assignment is to a variable or array element. In addition, a call to a dummy argument is checked to confirm that the corresponding call argument is a subprogram.

INTRINSIC FUNCTIONS

The intrinsic functions available in Time-Sharing are listed in Appendix C. The various names by which they may be called, the number and assumed mode of arguments, and the result mode for each function are shown. The modes of arguments are automatically adjusted to the required mode for each library function. The mode of the result is automatically converted to an integer value, if used in an integer expression (as a subscript expression for example), or to a real number value, if used in a real expression.

EXTERNAL SUBPROGRAMS

External subprograms are called "external" because they do not share names with each other. The only names used in external programs that are shared are the names of the subprograms themselves. One external subprogram may use the same name to mean something entirely different from its meaning in another subprogram. In regard to names, an external subprogram is entirely independent from other subprograms.

External Functions

Since an external function does not share names with other subprograms (except for the names of the subprograms themselves), use of values assigned in another subprogram or assignment of values to be used in another subprogram is restricted to two kinds of correspondence.

The first of these is the correspondence between the dummy argument in the function definition and the actual argument in the calling program, as described earlier under "Dummy Arguments". The other allowable sharing of values between external subprograms is achieved by declaring variables or arrays in one subprogram to use the same storage as declared for variables or arrays in the other. This declaration is done through a COMMON statement. The correspondence is established only by the sharing of the same storage; it is immaterial that a variable has the same name in one subprogram as in another.

Function Definition. The definition of an external function must occur before the main program or at the end of the main program, never within the main program. If a function has been typed real or integer, it must occur before the main program.

General Form

```
type FUNCTION name (arg1, arg2, . . . ,argn)
        o
        o
        o
    name = expression
        o
        o
        o
    RETURN
        o
        o
        o
    END
```

The first statement in the definition of an external function must be as indicated in the general form where:

- The name of the function may determine the mode of the result it returns. If the name begins with any of the letters I, J, K, L, M, or N, its implied mode is integer. If it begins with any other letter, the implied mode is real. The implied mode may be overridden by an optional specification that one mode is assumed for all names (refer to $OPT, Page 91). Either an implied mode or the assumed mode may be overridden by writing the words REAL or INTEGER in front of the word FUNCTION.
- The argument list is optional
- If the function name is longer than three characters, the final character may not be an F.

Examples

```
REAL FUNCTION INDEX (A,B,C,D)
FUNCTION SQROOT (A)
INTEGER FUNCTION SUM (A,B,C)
FUNCTION OUTPUT
```

The statements comprising the external function follow the FUNCTION statement.

The value of an external function is normally returned through a result variable, whose name and mode are the same as that of the function. This is done by placing the result variable in an assignment statement.

Example

```
100 FUNCTION  DISCR (A,B,C)
110 D = B*B
120 DISCR = D - 4*A*C  'RETURN VALUE OF DISCRIMINATE
130 RETURN
140 END
```

Note: the value of the result variable, DISCR, is established at line 120. This value is transmitted as the result of the function.

A RETURN statement causes control to be transferred to the calling program. There may be any number of returns within the function definition. An END statement with no characters following it indicates the last statement in the definition. If the next sequential statement is either a FUNCTION or a SUBROUTINE statement, or the last executable statement in the total program, the END statement may be omitted.

USE OF EXTERNAL FUNCTIONS. There are two ways to use external functions.

- They may be referenced in expressions.
- They may be referenced in a CALL statement.

A function is used in an expression by giving its name followed by the parameter list.

Example

```
FUNCTION HIPOT (A,B)
HIPOT = SQRT (A*A + B*B)
RETURN
END
```

The function in the above example (to calculate the hypotenuse of a right triangle) may be used in expressions such as the following:

```
X = SIN(HIPOT(A,B1))
Y = HIPOT(A,7.1)*Z1
PRINT,HIPOT (5.6,7.)
```

To use the function HIPOT with a CALL statement, the following definition would be used:

```
FUNCTION HIPOT(A,B,C)
C = SQRT(A*A + B*B)
RETURN
END
```

To then use HIPOT with a CALL statement, the following statements would be written (these correspond to the examples showing how to use this function in expressions).

```
CALL HIPOT(A,B1,ANS)
X = SIN(ANS)

CALL HIPOT(A,7.1,ANS)
Y = ANS*Z1

CALL HIPOT(5.6,7.,ANS)
PRINT, ANS
```

For further information, see the sections on FUNCTION call, page 35 and the CALL statement on page 37.

Subroutines

A subroutine does not return a result variable, otherwise it is exactly like an external function.

SUBROUTINE DEFINTION. The definition of a subroutine must occur before the main program or at the end of the main program, never within the main program.

General Form

```
SUBROUTINE name (arg1, arg2,,,,,argn)
    o
    o
    o
RETURN
    o
    o
    o
END
```

This first statement in the definition of a subroutine must be as indicated in the general form where:

- The name of the subroutine does not imply a mode, and if one of the mode qualifiers (REAL or INTEGER) is given, it is ignored.
- The name of a subroutine used within the subroutine refers to a variable to which a value may be assigned and whose value may be used. The value of this variable, however, is not supplied to the calling program when control is returned. The name of the external subroutine may not be used as a variable in any calling program.
- The argument list is optional.

Examples

```
SUBROUTINE GOMO
SUBROUTINE FLUSH (A,B,C,Z)
```

The statements comprising the body of the subroutine follow the SUBROUTINE statements.

The only way a subroutine can transmit values to the calling program is through the argument list, and through COMMON.

The definition of a subroutine is terminated in exactly the same way as that of an external function.

USE OF SUBROUTINES. A subroutine may be referenced only by a CALL statement.

Example

```
SUBROUTINE EXTRAN (A,B,C)      ⎫
COMMON BASE                    ⎪
REAL N                         ⎪
N = (A*B)**3                   ⎬  Subroutine
C = (BASE - A)/(B - SQRT(N))   ⎪
RETURN                         ⎪
END                            ⎭
COMMON BASE                    ⎫
INPUT, BASE                    ⎪
CALL EXTRAN(X,Y,RES)           ⎪
PRINT, RES                     ⎬  Main Program
X = X*RES                      ⎪
BASE = 2.*BASE                 ⎪
CALL EXTRAN(X,Y,RES)           ⎪
PRINT, RES                     ⎪
END                            ⎭
```

Note: Values are transmitted through the calling arguments and COMMON. Also, the variable name N is known only to the subroutine EXTRAN. The name N could have also been used in the main program referring an entirely different variable.

For additional instructions on how to call a subroutine, see the discussion under CALL Statement, page 37.

Main Program

A main program is an external subprogram without a name. It is called only once, upon initial execution of the program in which it is a subprogram. There must be a main program to which control can initially be passed.

Like external functions and subroutines, a main program shares no names with those used in other external subprograms. A main program may share values with other external subprograms only by establishing a shared storage for the values (COMMON), or by call argument, dummy argument correspondence.

MAIN PROGRAM DEFINITION. A main program is defined by writing statements, without preceding them with a FUNCTION or SUBROUTINE statement, either at the beginning of the program or after the END statement terminating the definition of another external subprogram. If a FUNCTION definition of a SUBROUTINE definition precedes the statements of the main program, but fail to have an END statement, the statements of the main program are interpreted as part of the subprogram definition.

Execution of the RETURN statement if executed before an ENTRY statement, the END statement, or the last statement in a main program ordinarily stops execution of the entire program, of which the main program may be only a subprogram.

If the main program has been re-entered by execution of an ENTRY statement (a special statement provided by Mark I FORTRAN) execution of a RETURN, END, or the last statement causes control to return to the point from which it was last called. Refer to "Entry Statement," page 49.

INTERNAL SUBPROGRAMS

An internal subprogram may occur inside any kind of external subprogram, function, subroutine, or main program. It then shares names with the including subprogram but not with any other external subprogram. Its own name is unknown outside of its including subprogram.

The definition of an internal subprogram must precede any statements containing references to it.

Arithmetic Statement Function

An arithmetic statement function definition consists of a single arithmetic statement. It is recognized by the occurrence of a parenthesized list of dummy arguments (possibly empty) appended to a name, not previously declared to be an array. This list appears on the left of the equal sign in an arithmetic statement. The evaluation of the expression to the right of the equal sign yields the function result.

General Form

type, name (arg_1,arg_2,...,arg_n) = expression

As with other kinds of subprogram, the definition of an arithmetic statement function (ASF) consists of a function name, a list of dummy arguments, and the instructions comprising the function. It also may be qualified with the words REAL or INTEGER. There are, however, these differences:

- If dummy arguments are omitted, an empty set of parentheses is used.
- The definition is confined to a single arithmetic assignment statement.
- If the qualifiers REAL or INTEGER are used, they are written before the function name and must be separated from it by a comma.
- The word FUNCTION is not used.

The function name may imply the mode of the result returned by the function. An assumed mode may be applied by the control option. Either the implied or assumed mode may be overridden by means of the qualifiers REAL or INTEGER.

Examples

```
HAV(X) = .5* (1-COS(X) )
REAL, LOG10 (X) = LC G(X)*.43429448
INTEGER, CATENATE (I,J,K) = 4096*I + 64*J=K
DET3(A,B)=(S=.5*B )*S+(C=.333333333*A)*C*C
```

USE OF THE ARITHMETIC STATEMENT FUNCTION. An arithmetic statement function definition may occur anywhere in the subprogram prior to any references to the function except within the range of a DO.

An arithmetic statement function is "called" simply by being referenced within the including subprogram. An ASF cannot respond to an array because there is no way of declaring an array dimension in the single statement that comprises the ASF.

Two conventions for establishing modes in ASF's are specifically not followed in Mark I FORTRAN. In FORTRAN II, peculiar rules are used to infer from the name of the ASF, the mode of the result returned by the ASF:

- The name has to have four or more characters.
- The name must end with the letter "F".
- The name is implied to have real mode unless the name begins with the letter "X".

In FORTRAN IV, regular naming rules are used for the name of an ASF. But, the modes of ASF dummy arguments may be declared in mode declarations preceding the ASF definition.

Neither the peculiar ASF naming rules of FORTRAN II nor the mode declarations of ASF dummy arguments given prior to the ASF definition, permitted in FORTRAN IV, are used in Mark I FORTRAN.

Examples

```
10 COSH(X) = (EXP(X) + EXP(-X))/2
20 INPUT, TO, WO
30 DO 1 X = 0.,50.,.1
40 Y = TO/WO*COSH(WO*X/TO)
50 1 PRINT, X, Y
```

General Internal Functions

In Mark I FORTRAN, the concept of the ASF is generalized to an internal function whose definition may consist of more than one statement.

Like an ASF, a name used in a general internal function refers to the same element referred to by other uses of the same name throughout the including external subprogram. This holds true both within other internal functions included in the same external subprogram and in the "main" part of the external subprogram.

Dummy arguments and statement labels are not so shared, however. Like an ASF, an internal function definition may have a dummy argument list. The names in this list are dummy argument names and do not refer to the same thing referred to by the same name outside the internal function.

GENERAL INTERNAL FUNCTION DEFINITION. The definition of a general internal function must precede any references to the function.

General Form

type, name(arg_1,arg_2,...,arg_n):
o
o
o
name = expression
o
o
o
RETURN
o
o
o
END name

The first statement in the definition of a general ASF must be of the form specified above where:

- A mode qualifier (REAL or INTEGER) may precede the name; it must be separated from the general internal function by a comma.
- The name is followed by a parenthesized list of arguments. The parenthese are required if no arguments are given. This is required to distinguish the name from a statement label.
- The argument list is followed by a colon

Examples

RUST (X):
KANS (A,B):
REAL,JUST( ):
INTEGER, FIRST(L):

The statements comprising the general internal function follow the lead statement described in the foregoing.

Statement labels are internal to the general internal function. References to a statement name or number within a general internal function are taken to refer to statements within the internal function labeled with the name or number. If no statement so labeled occurs within the internal function, the label references are marked as erroneous. Label variables are similarly interpreted as internal only.

An internal function returns a result in the same way that an external function does, through the result variable or through calling arguments. The RETURN statement is executed as in the external function.

After the first statement in the definition of the general internal function, the statement or statements comprising the internal function are written. The last statement should be an END statement in which one or more characters are written after the word "end." For example: ENDA or END INTERNAL

The END statement may be omitted if the next statement is the first one of another internal function definition. This convention prevents nesting of internal functions. The attempt to include an internal function within another internal function merely concludes the definition of one and initiates the definition of the other.

USE OF THE GENERAL INTERNAL FUNCTION. General internal functions are referenced in the same way as external functions except for the reference to a general internal function that has an empty argument list. In this instance, the empty parentheses must be provided.

Examples

```
AREA(R):
A = PI*R*R
RETURN
ENDA
PI = 3.14159
RADIUS = 4.5
CALL AREA(RADIUS)
PRINT, "THE AREA OF THE CIRCLE IS",A

AREA( ):
AREA = PI*R*R
RETURN
ENDA
PI = 3.14159
R = 4.5
PRINT "THE AREA OF THE CIRCLE IS", AREA()
```

Note: in the first example, the variable names PI and A are known to the main program as well as the function. A CALL statement was used because the result variable was not established. In the second example, the variable names PI and R are known to the main program and the function. The function call uses the empty set of parentheses for the proper reference. Both function definitions are terminated with the statement ENDA as opposed to END.

ENTRY

The ENTRY statement provides a means of directing control to a subprogram at some point other than the first executable statement of the subprogram. The statement consists of the word ENTRY, a name, and a parenthesized list of dummy arguments that serve the same purpose as the dummy arguments in a FUNCTION or SUBROUTINE statement. The dummy arguments may be omitted, in which case the parenthese are also omitted. The ENTRY statement is written just prior to the statements that are to be executed when the entry is called.

The entry name must not be the same as any external or intrinsic subprogram name or any other entry name in the entire program. Furthermore, the entry name must not be the same as any function name internal to the subprogram containing the ENTRY statement.

If a dummy argument name in an ENTRY statement is the same as one in the subprogram definition introduction statement, or in a previously given ENTRY statement, no declaration providing information about the dummy argument can be given. If the name is used for the first time in the dummy argument list of an ENTRY statement, however, declarations should be given as for a subprogram definition introduction. Such a declaration should immediately follow the ENTRY statement.

An ENTRY statement can be used in either an external subprogram, including the main program, or an internal function. When control is directed to a subprogram because of a call to an ENTRY statement, the return from the subprogram is to the statement containing the call to the ENTRY statement. In other words, return is made just as if it were a call to the subprogram definition introduction statement. On the other hand, if the call is a function call, the result returned is the value of the result variable referred to by the external or internal function name. The value is not a variable referred to by the entry name. A mode qualification of the entry name, therefore, serves no purpose.

If a subprogram has been last entered via an ENTRY statement, return is to the point from which the ENTRY statement was last called. If an ENTRY statement is one of the defining statements for an internal function, entry via it affects return from the internal function. However, this entry does not affect return from the subprogram containing the internal function.

Example

```
100       INPUT,I
110       PRINT,I,"MEANS",
120       IF(I)SR,A,B
130 SR:   CALL SUBR; STOP "SR"
140 A:    CALLA; STOP "A"
150 B:    CALLB; STOP "B"
160 SUBROUTINE SUBR
170       PRINT "SUBR ENTRY"
180       ENTRY A
190       PRINT "A ENTRY"
200       ENTRY B
210       PRINT "B ENTRY"
220       PRINT "RETURN TO CALLER"
```

Example (Using an External Function)

```
      F=PONS (D(I)+2.)-1.
      FUNCTION PON (X,Y)
      IF (X) XN
      IF(Y) YN
      T=Y*SIN(Y)

EX:PON=T/X*COS(X)
      RETURN
XN:X=ABS(X);GOTO EX
ENTRY PONS(X)
YN:T=1,GOTO EX
```

Example (Using a General Internal Function)

```
      PON(X,Y):
      IF(X) XN
      IF(Y) YN
      T=Y*SIN(Y)
EX:PON=T/X*COS(X)
      RETURN
XN:X=ABS)X);GOTO EX
ENTRY PONS(X)
YN:T=1,GOTO EX
      END PON
      F=PONS(D(I)+2.)-1
```

Return is to the expression PONS (S(I)+2.)-1 with the value of PON. Evaluation of the expression continues, subtracting 1 from this value and assigning it to F. Note that in using the general internal function the definition must precede the reference and the form of the ENTRY statement is the same as when used with external functions.

9. INPUT/OUTPUT

There are four kinds of statements in Mark I FORTRAN that may be used to perform input or output. Two are for terminal input/output, and two for file input/output.

INPUT ... is used for input from the terminal
PRINT ... is used for output to the terminal

READ ... is used for input from a file
WRITE ... is used for output to a file

All of these statements may specify a format to be used in conjunction with the transmission. If the format specification is not given in any of these statements, unformatted input or standard output format is provided.

All these statements may include a list to indicate which values are to be transmitted as output or which variables, array elements, or arrays are to receive values that are transmitted as input. When no list is included in an output statement, all the values are included in the specified format. When no list is included in an input statement, all the values are received by the specified format.

File input/output statements may also include an indication of which file is to be used. If no file indication is given, a standard temporary file is used.

INPUT/OUTPUT LISTS

The list included in an input/output statement may contain one or more list items separated from each other by commas. The kinds of program elements which can be used, either for input or output, or both, follow:

1. Constant --Input or Output

   Decimal integer, Octal integer, Decimal real, Quoted character (1, 2, or 3 characters)

2. Variable or array element --Input or Output

3. Array --Input or Output

4. Part array (indicated by using a special repetition form) --Input or Output

5. Expression --Output only

6. Quoted legend (of any length) --Output only

7. Slew control characters --Output only

Constants are used in output lists to transmit constant values as output. In an output list, constants may occur for use with a T-type or *-type specification in the specified format. (Refer to "Variable Format Specifications" on Page 85.)

```
READ VARY, "E", 16, 5, B

VARY: FORMAT (T*.*)

PRINT ANS, 1., SCORE (1), 2., SCORE (2)
```

Variables or array elements are used in output lists to transmit their values. In input lists, they are used to receive input values. There are no restrictions placed on the kinds of expressions that may be used to subscript the array name to indicate a particular array element.

Example

```
PRINT, A (I↑2, N(L(I))), V, M
```

Input or output of an entire array is indicated by writing the name of the array without an appended subscript expression. Array values are transmitted and received in the same order as they are stored. (Refer to "Array Storage" on page 25.)

Example

```
READ, B, L
```

Input or output of part of an array is indicated by using a special repetition form for input or output lists. The repetition form includes the DO parameters:

- Control variable
- Initial value
- Final value
- Increment value (an integer 1 is used if none is given)

These DO parameters are written in the repetition form exactly as they are in the DO statement:

control variable = initial value, final value, increment value

In the special repetition form, the DO parameters are written after array elements that are to be transmitted repeatedly. They are separated from the array elements by a comma. Both array elements and DO parameters are included within parentheses.

(array element, array element, ..., control variable = initial value,
final value, increment value)

Example

```
INPUT, (L(I), B(I*2),I = 1, 25)

READ , ( (A(I,J), I=1, 10), J=1, 10)
```

As with DO statements, repetition forms may be nested. Notice the positioning of parentheses and the comma separating the final DO parameters from the previous repetition form.

A common use of repeated input is for count control input of a variable number of array elements. In this use, the list names the DO parameter final value prior to the repetition form in the input list.

Example

```
READ, K, (L(I), I = 1, K)
```

provides for reading of as many array elements as the first value read indicates.

If the first value is 25, for example, 25 subsequent values are entered and stored as the first 25 array elements.

In Mark I FORTRAN, an expression may be included in an output list to provide for output of the value of the expression.

Example

```
PRINT, SIN (X) + V * B(J)↑3.1
```

In addition, a quotation may be used in an output list in Mark I FORTRAN. The quotation is written with a quotation mark ("). The quotation may consist of any characters, except a quotation mark which would terminate it.

Example

```
PRINT "EASY TO USE, FREQUENTLY DESIRED"
PRINT "NOT"
INPUT 6, "G", A
PRINT 6, "I", A
PRINT, (T/64) * 64 + "S"
```

The preceding up arrow causes a line feed to be transmitted. Only a line feed is transmitted, not a carriage return line feed combination. When the up arrow precedes the first item in the list, the carriage has usually been returned so that slewing is accomplished. However, the line feed may be transmitted anywhere in the output by preceding any list item with an up arrow. The effect of this is a stepping of the output.

Example

```
PRINT, "D",↑ "O",↑ "W",↑ "N"

Produces:   D
              O
                W
                  N
```

The extra comma at the end of an output list prevents the current output record to the terminal from being ended. The next output to the terminal transmits values to the same record.

Example

```
PRINT, "ONE ",

PRINT, "AND ONLY ONE"

Produces:

     ONE AND ONLY ONE
```

UNFORMATTED INPUT

If no format specification occurs in an input statement, each value is transmitted with conversion based on the characters appearing in the representation of each value.

One value is separated from another value in the same input record by blanks or commas or both.

As much of an input record is transmitted as is required to satisfy the input list. If the list requires fewer values than contained in the current record, the remaining values in the record are used as the first values for the next request for unformatted input from the same source--terminal or file.

Each record, unless it is a terminal input record, is assumed to begin with a line number which is skipped. Consequently, except for terminal input, if the first character in the input record is a digit, it and succeeding digits are interpreted as a line number.

Line numbers are not transmitted in unformatted input. Therefore, if a record to be transmitted via unformatted input begins with a value, and not a line number, it is important that it be preceded by a nondigit to prevent it from being mistaken for a line number.

Apostrophe-introduced comments may be included in the input record, the initial apostrophe being preceded by a delimiter (blank or comma). Such comments are terminated by the end of the record or by the next apostrophe. Comments are not transmitted.

Any value may be indicated to be duplicated by writing a count of the number of times the value is to occur and an asterisk (*) in front of the value. By this means, a single input record suffices for an entire array when all the array elements are to receive the same value. The data count is not carried over to the next input statement.

Example

```
625*-2
Provides 625 values, each of which is a negative real two
```

A duplication count of zero supplies a practically unlimited number of values.

There is no provision for continuing a value from one record to another. The end of record indicates the end of the last value in the record.

Values are converted to one of three types:

- Decimal
- Octal
- Alphabetic

Conversion depends on what characters appear in the value representations.

Decimal. If the first character in a record is a digit, decimal point, or sign (+) or (-), the value is interpreted as a decimal, integer, or real. If the input list requests a real the value is converted as a real, otherwise it is converted as an integer. Notice that a blank must be omitted in a multiplicative factor since it would indicate the end of the value. Use '25E3' not '25E 3'.

Octal. If the first character is a slash (/), succeeding characters in the value are interpreted as an octal, integer or real.

Alphabetic. If the first character is not a digit, period, sign, slash, apostrophe, or asterisk, the value is interpreted as alphabetic. Notice that the alphabetic value may not include blanks or commas. It may not begin with a digit, period, sign, slash, apostrophe, or asterisk. However, these characters may be included by enclosing the alphabetic within quotation marks. If the list item is an integer, a maximum of three characters are received. If the list item is real, a maximum of six are received. Remaining characters in the alphabetic value are retained.

For more details on how conversion is handled, refer to the discussion given for different format specifications:

- Decimal – refer to "F-Format" on page 75.
- Octal – refer to "O-Format" on page 80.
- Alphabetic – refer to "A-Format" on page 83.

Example

```
1.    -14.5, 32, 6E12, 5+1, -13, .2$-3, 100*0.
```

These are all decimals

```
2.    /17777771777777, /37760000000000
```

These are the largest and smallest positive reals in octal.

3. JAN,FEB,MAR,APR,MAY,JUN,JUL,AUG,SEP,OCT,NOV,DEC

These are all alphabetic.

4. " ", "/", S, T, U, "1."

These are also all alphabetic. Quotation marks are used to enclose characters that would otherwise be interpreted as non-alphabetic or be skipped.

5. 1350 150 'COUNT' 150*-1. 'VALUES'

This record begins with a line number which will be skipped; comments may be enclosed on a data line in between pairs of apostrophes. The second apostrophe may not be omitted.

STANDARD OUTPUT FORMAT

If no format specification occurs in an output statement, values are transmitted as output with a conversion based on the mode of the item in the list. The spacing used provides for decimal point alignment except when the numbers are too large or too small to express without a multiplicative factor. To provide for use with dollar and cent values, zeros after the first two digits after the decimal point are suppressed. The standard output format provides for five values per output record.

Examples

```
 373.45          0.20       143721.25       1857.60      42.14
14172.15       -136.16            2.00        117.25
                                   . . .
3.7821E+06         270             -43                32.1417-1.72150E-05
```

Standard output to an external file, or the core file, separates the data field by one blank for integers and two blanks for real numbers according to the following rules.

- F output is used for floating point numbers in the range $10^{-4} < X < 10^{7}$ with trailing zeroes suppressed.
- E output is used for floating point number outside of the above range.
- Records are of variable length with a maximum length of 72 characters.
- Line numbers are automatically provided, beginning with a value of 1000 and incremented by 10 for each line number.

Files written using standard output must be read using unformatted READ statements.

TERMINAL INPUT/OUTPUT

Records transmitted to or from a terminal are restricted to a maximum of 72 characters. This is all that many teletypewriters can display. The 72-character size is a maximum; shorter records can also be transmitted.

PRINT Statement

The PRINT statement is used to transmit values to the terminal. Its general form is:

PRINT format reference, output list

If the format reference is missing, standard output format is used. When no format reference is given, the comma normally written after the format reference, must be written after the word "print."

```
PRINT, output list
```

If the comma is omitted, the first item in the output list is mistaken for the format reference. When the first item in the output list is a quotation, the comma may, but need not be, omitted after the word "print," since it cannot be mistaken for a format reference.

```
PRINT quotation
```

The form

```
PRINT format reference
```

is also allowed. The referenced format may consist of only those format specifications that are not used in conjunction with list values. Such format specifications are the H-format, the X-format, and the slash (/) used to indicate beginning of a new record. When there is no output list, transmission consists of only the characters provided by these format specifications.

Examples

```
PRINT 45, A

PRINT REP, (A(I), I = 1, 10)

PRINT, A

PRINT, (A(I), I = 1, 10)

PRINT, "MONTHLY SUMMARY"

PRINT "VERTICAL VALUE = ", A(3)

PRINT VAR, "E", A

PRINT HDS, "PAYNO", "SCORE", "Q-FACTOR"

PRINT LUBE, VIS,↑"VISCOSITY", API, "API NO."

PRINT, ↑ QUAN, RATE,

PRINT TITLE
```

Two slew control characters are used in Time-Sharing FORTRAN:

- An up arrow (↑) in front of an output list item
- An extra comma (,) at the end of the list.

Traditional FORTRAN uses the first character in a display output record as a slew control character and does not display it. The characters shown below are sometimes used to indicate slewing.

blank	to slew one line
0	to slew two lines
1	to slew to the top of the page
+	to slew no line

Mark I FORTRAN does not use the first character as a slew control character. The first character in the record is displayed like any others.

Slewing can be controlled in Mark I FORTRAN by using either the slash (/) specification in a format or the preceding up arrow and extra comma in the output list. (Refer to "Input/ Output Lists" on page 52.)

Terminal output records are not directly transmitted. Records are stored internally for subsequent transmission. Actual transmission to the terminal occurs when:

- There is no more internal space to store terminal output records. (Space accommodates about thirteen 72-character records.)
- Execution of the program is suspended for terminal input.
- Execution of the program is suspended to give another program a "turn" at execution.
- Execution of the program stops.

When the internal space available for terminal output is filled before transmission occurs, about a minute and one-half of teletypewriting is required to empty it. Execution of the program is not resumed until all of the output transmission is completed.

INPUT Statement

The INPUT statement is used to transmit values from the terminal. Its general form is:

INPUT format reference, input list

If the format reference is missing, unformatted input is used. When no format reference is given, the comma normally written after the format reference, must be written after the word "input."

INPUT, input list

If the comma is omitted, the first item in the input list is mistaken for the format reference.

The form

INPUT format reference

is also allowed. The referenced format may consist of an H-format. In this case, transmission is from the terminal to the format sequence. (Refer to "H-Format" on page 82.)

Examples

```
INPUT REPLY, QUAN
INPUT, (COEF (I), I = 2, L, 2)
INPUT NAME
INPUT 17, CITY, STATE
INPUT, TL, RELOC, BONUS, SS
INPUT 36, 4, "I", 3, M, T
INPUT, SENSESWITCH3
```

When an input statement is executed, a question mark (?) is transmitted to the terminal. At the same time, execution of the program is suspended, and it is not resumed until an input record is provided.

In response to the question mark request, an input record is provided by entering the characters in the record and following it with a carriage return to indicate that all have been entered. The characters between the question mark and the carriage return comprise the terminal input record.

If the input list in an INPUT statement with a format reference requires fewer values than the record contains, the remaining values are discarded. If the input transmission is unformatted, however, the remaining values are retained for use in satisfying the requirements of the input list in the next INPUT statement that is executed. (Refer to "Unformatted Input" on page 55.)

If the input list requires more values than the record contains, a request is made for another terminal input record.

An exception occurs when transmitting alphabetic values using either H-format or A-format. When transmitting any input records under H-format or A-format, blanks are inserted in the record after the last character entered to fill the record to 81 characters. If the input list does not require as many characters, the extra ones are ignored. However, if fewer characters are entered than required by the input list, blanks are supplied automatically to satisfy the list. (Refer to "Alphabetic Format" on page 82.)

When the execution of an INPUT statement requests a terminal input record by transmitting a question mark (?) to the terminal, the user may elect to stop rather than resume execution. To indicate that execution is to stop the user enters the word "STOP," as the first characters after "?". However, this may cause the loss of output to external files.

FILE STATEMENTS

The kinds of statements used to perform file operations are listed below in two groups; one for permanent file operations and one for temporary file operations.

Permanent Files:

READ (file reference, format reference) input list

READ (file reference) input list

WRITE (file reference, format reference) output list

WRITE (file reference) output list

BACKSPACE file reference

ENDFILE file reference

IF (ENDFILE file reference) label reference, label reference

REWIND file reference

Temporary File:

READ format reference, input list

READ, input list

WRITE format reference, output list

WRITE, output list

BACKSPACE

ENDFILE

IF (ENDFILE) label reference, label reference

REWIND

Permanent File Definition

Permanent files are defined by saving an item in the user's time-sharing system catalog. The saving is performed the same way as when a user saves a program. This is done by means of the operating system command SAVE.

If the file has initial values, a file large enough to hold them can be defined by entering the initial values, naming them with the file name, and saving them. Like saved programs, file initial values must have line numbers in order to save the program. After the line number, the values should be entered on the line in a format consitent with that by which the file is read. The values should be separated from the line number by a blank. If the file is read without format, the values should be separated from each other by either blanks or commas. As many values as convenient may be entered on each line; no value should be continued from one line to the next. (Refer to "Unformatted Input," page 55.)

If the file is to be read with a format, each line is interpreted as an input record. The format should provide for the line number and for as many values as are entered on the line. In addition, the format should be consistent with the kinds of values entered.

If the file has some initial values, but not enough to fill the file, extra lines should be entered after the initial values until there are enough to reserve a file space of the right size. The operating system command LENGTH may be used to obtain a count of the number of characters so far entered.

On the other hand, if the file is empty, a space for the file must be reserved by saving an item of the same size as the space that is required. For convenience, the library includes items of the six different sizes that are distinguished. The user may request one of these items, rename it with the file name, and save it in his own catalog. The names by which these items are saved in the library and the sizes of each are shown in the table on the following page.

LIBRARY NAME	NO. OF CHARS.	MAX. NO. OF 72-CHR. RECORDS
CH0192***	0-192	2
CH0384***	193-384	5
CH0768***	385-768	10
CH1536***	769-1536	21
CH3072***	1537-3072	42
CH6144***	3073-6144	85

The smaller size files are useful for operations with single records. The larger size files are useful for multi-record operations similar to short tape operations.

Moreover, Mark I FORTRAN may be instructed to consider some files as successors to others. By this means, multi-record operations similar to the more usual length tape operations can be performed. (Refer to "Linked Files" page 63.)

REWRITING FILE OPERATIONS. Like tape operations, file operations in Time-Sharing FORTRAN may require rewriting of all records in the file following one that is changed. When the file has only one record, there are no other records to rewrite. However, a file might, by means of successor files, consist, for example, of 120 records and if the 20th record is changed or removed, the last 100 records may need to be rewritten. In many applications, therefore, it is an advantage to have numerous short files, since only the remainder of the shorter file need be rewritten when a record in it is changed.

Rewriting of individual file records without rewriting subsequent records in the file may be done only if the number of characters in the replacing and replaced records are equal. Note that the size of a record may be lengthened because of field widening. (Refer to "Numeric Field Widening" on page 73.) If the replacing record is longer, part of the record after the replaced one will be lost and this subsequent record will be shortened. If the replacing record is shorter, the part of the record not replaced remains as an extra record that follows the replacing record.

Deletion of a record always requires rewriting of subsequent records if the space required for the deleted record is to be recovered.

Insertion of a record always requires rewriting of subsequent records.

The rewriting of the records subsequent to one that is changed, deleted, or inserted may be best performed by the operating system. To do so requires that each record begin with a line number. To use this facility, in the program, write the changes to a file at the end of the file. Then, after all the changes have been written and the program has stopped, reference the file from the terminal, using the command OLD. Make a dummy change to the file and save it using the command SAVE. For example, a dummy change might be an empty line with a line number not used by any of the records in the file. The dummy change is required to cause the operating system to reorder the file prior to saving it.

$FILE - (PERMANENT FILE REFERENCE). Permanent files are referred to by naming them in a $FILE control line. The names given must be present in the user's catalog of saved items. Moreover, the names must also be acceptable FORTRAN names, that is contain only alphanumeric characters except that the sixth characrer may be an asterisk.

Saved file names are restricted to six characters. When names exceeding six characters are given in the $FILE line, a name consisting of the first six characters is what is looked for in the catalog.

The general form for the $FILE line is:

$FILE name, name, ..., name

There may be several $FILE lines in the program only if they follow one another, or if only declaration statements intervene. The $FILE line or lines must be given near the front of the program. There is no restriction on how many names may be listed in a $FILE line, and a line may be continued to list as many as are required. File names need not be distinct from names used in the program.

The order in which the names are listed in the $FILE line is used to provide a means of referring to files in the program. Outside of the $FILE line, files are referred to in the program by numbers, not names. The numbers indicate the order of the file names in the $FILE line. Program reference to file 1 is a reference to the file whose name is first in the list of names in the $FILE line. File 2 refers to the file whose name is second, and so on.

The program reference to a file may be by means of a constant or variable. The value of the constant or the current value of the variable is truncated to an integer if it is a real. A zero value is taken to be a reference to the temporary file. (Refer to "Temporary File" on page 68.) If the file reference is omitted, a zero value is assumed.

Example

```
$FILE SUMMARY, TEST, MANPOW

WRITE (1) ... means write in the file saved by the name "SUMMAR"

READ(I)   ... when I = 1 means read from the file saved by the name "SUMMAR";
              when I = 2, from file "TEST"; and, when I = 3, from file "MANPOW."
```

Error messages are given if the file reference is a positive value larger than the number of files defined for the program, or negative value.

LINKED FILES. To indicate that one file is to be used as a successor to another, the names are listed with a slash (/) between them, instead of a comma.

Example

```
$FILE PERS1/PERS2/PERS3/PERS4/PERS5, SUM, EXCEPT
```

The files named first, PERS 1 through PERS 5, are indicated to be a single file. Except for the last file of a succession, when no more values are available from one of these files, or no more space is available to put values in, the file named after the exhausted one is used. Similarly, backspace or rewind operations that refer to the first file in a sequence of linked ones, work as though the entire sequence of files were one.

Reference to file 1 in the preceding example, is a reference to the entire linked file PERS1 through PERS5, and reference to file 6 is a reference to a file named SUM. Reference to file 2 is a reference to PERS2, a part of the succession of files.

Record lengths are variable with a maximum of 72 characters including the carriage return.

Records written by means of user-supplied formats may consist of 71 or fewer characters plus a carriage return. The carriage return is provided by the file output program. To conform to requirements of the time-sharing operating system, the carriage return is placed as the 6th, 12th, 18th, ..., 66th, 72nd character. Fill characters are inserted as required between the last data character and the carriage return. The table below gives the record length resulting from different numbers of data characters.

Number of Data Characters	Record Length in Characters
1-5	6
6-11	12
12-17	18
18-23	24
24-29	30
30-35	36
36-41	42
42-47	48
48-53	54
54-59	60
60-65	66
66-71	72

When a file-record write of more than 71 data characters is attempted, the 72nd character is a carriage return and the remaining data characters are written in a second record. Numeric field widening permits the record length to exceed that specified in a format.

"OFF-LINE" FILE OPERATIONS. Files written by executing FORTRAN programs may be listed using the time-sharing system LIST command. The listing of records will be in the order in which they were written whether the records have a record sequence number or not. (The time-sharing system does not reorder lines according to line numbers if the lines are all saved ones, that is, no new lines have been entered from the terminal.)

When each record begins with a record number, the file can be edited using the facilities of the time-sharing system. The file can be merged with others using the EDIT commands, copied using the RENAME and SAVE commands, and reordered according to record numbers using the SAVE command (after making a change to the file from the terminal.) Records can be renumbered using the EDIT RESEQUENCE command, and deleted, inserted, or replaced using their record numbers as line sequence numbers.

A record number is included in each record when the standard output format is used, and can be provided when a format is specified. To meet the requirements of the time-sharing operating system, the record number must be a one to five-digit number (separated by a space from the next datum in the record if it is numeric).

There are no provisions in the time-sharing system for listing or editing entire linked files. Each file in a succession of linked files must be listed or edited separately. To facilitate listings of linked files, the standard output format produces an integral number of records in each link, that is, does not split a record across a file and its successor.

END OF FILE

Definition. The end of file is either a mark indicating the end of data in the file or it is the end of the space in the file. The mark is the octal integer 777755 (in an even location). It is written in one of the following ways:

- By the operating system at the end of the values saved in the file by means of the command SAVE.
- Throughout the files saved in the library that may be used to establish the size of files saved in the user's catalog by means of the commands RENAME and SAVE. (Refer to "Permanent File Definition," page 61.)
- By the FORTRAN statement ENDFILE (described below).

The size of the file is established at the time the file is defined and cannot be changed except by redefining the file. (Refer to "Permanent File Definition", page 61.) The file space that can be written in is shorter than its defined size by six characters to ensure that every file has at least one end-of-data mark.

End-of-File Writing. An end-of-data mark may be written by means of the ENDFILE statement. The form for the statement is:

ENDFILE file reference

The end-of-data mark written by the ENDFILE statement will:

- Terminate listing of the file by means of the LIST command
- Terminate editing of the file by means of the EDIT commands
- Cause reading to continue with the successor file when the file is linked
- Provide an end-of-data response to a test or read statement when the file is not linked.

The end-of-data mark does not mark the end of the space in the file. The file size, as mentioned previously, can be changed only be redefining the file.

A request to perform an ENDFILE statement is treated like a request to perform a WRITE. Once completed, it establishes the last operation on the file to be a write.

The record written by executing an ENDFILE statement consists of three characters, the octal integer 777755, placed in an even-numbered location.

Testing. An IF (ENDFILE) statement is available in Time-Sharing FORTRAN for testing for the end of data or end of space in a file. If the last operation performed on the file (by the current program) was a WRITE, the IF (ENDFILE) statement tests to see whether there is space in the file for at least one more 72-character record. Otherwise, the IF (ENDFILE) statement tests to see whether the next record is an end-of-data mark. The form for this statement is:

IF (ENDFILE file reference) label reference, label reference

If the second label reference is omitted, reference is to the statement following the IF (ENDFILE) statement. The first label reference is used when there is not an end of file. The second is used when there is an end of file. The table below explains statement use:

1. previous WRITE: IF (ENDFILE file reference) not end of space, end of space
2. otherwise: IF (ENDFILE file reference) not end of data, end of data

In a linked file, the IF (ENDFILE) statement tests for an end of space or end of data in the entire file. When there is a successor file, there is more space in the linked file. When there is data in a successor file, there is more data in the linked file.

Effect on Reading and Writing. Writing is affected only by an end of space condition. When the space in a file is exhausted and a WRITE in that file is required, the data is written in the successor file. If no successor file has been defined, the data is not written, and the exception notice "NO FILE SPACE" is transmitted to the terminal. Exectuion is terminated.

Reading is affected by an end-of-data mark. When the next file datum is an end-of-data mark, reading continues from the successor file. If no successor file has been defined, the exception notice "OUT OF DATA" is transmitted to the terminal, execution is terminated.

The exception notice "OUT OF DATA" can be avoided by including a test for the end of data before each read in the program. Upon encountering an end-of-data mark, the program can write past it using an ENDFILE statement and then continue reading. (If the end-of-data mark encountered is at the end of the file space, the attempt to write past it using the ENDFILE statement will cause the exception notice "NO FILE SPACE" to be transmitted to the terminal.)

Examples

1. Writing new data after old:

```
R:READ(3)
  IF (ENDFILE3) R
  WRITE (3) list
```

2. Reading vectors entered by user; entering minus ones for data not entered:

```
DO R, J=1,M
IF(ENDFILE 3)R
MN=M*N
JN=(J-1)*N+1
DO END, L=JN, MN; END: A(L) =-1; GO TO X
R: READ (3),(A(K,J), K=1,N)
X: 'VECTORS INTERED
```

3. Writing on a non-linked file; linking done by programmer:

```
I=1
DO MORE, J=1, 450,5
WRITE (I) A(J), A(J+1), A(J+2), A(J+4)
IF(ENDFILE I) MORE
I=I+1 'WHEN FILE IS FULL, USE NEXT FILE
MORE:  'CONTINUE WRITING
```

4. Reading past end of data mark

```
DO R, J=1,N
IF(ENDFILE 3) R
ENDFILE 3
R: READ (3) A(J)
```

REWINDING AND BACKSPACING. A file may be rewound or backspaced using the REWIND and BACKSPACE statements, respectively. Their general forms are:

REWIND file reference
BACKSPACE file reference

Execution of a REWIND statement causes the current position of the referenced file to be moved so that the next transmission is to or from the beginning of the file. When execution of the program begins, each file is rewound. It remains rewound until the file is positioned by execution of a READ, WRITE, or ENDFILE statement. Execution of a BACKSPACE statement when the file is rewound has no effect and is ignored.

Execution of a BACKSPACE statement causes the current position of the file to be moved to the point where it was positioned before the last transmission of a record to or from the file. If the next file operation is a read, the next record read is the last one read or written. If the next operation is a write, the next record written erases the last one read or written.

Note: When a file written with no format specifications is backspaced, execution via Mark I FORTRAN gives results which differ from those provided by traditional FORTRAN execution. An unformatted write in Mark I FORTRAN creates multiple records, each with a maximum of five values. In traditional FORTRAN, however, a write creates a single record large

enough to hold all values transmitted by the one WRITE statement. This difference is noticeable only when the file is subsequently backspaced. With Mark I FORTRAN, the backspace is over the single long record.

For example, the program

```
DIMENSION ARRAY (10)
WRITE (3) (I,I = 1,10)
BACKSPACE 3
WRITE (3) ARRAY
```

when executed in traditional FORTRAN causes file 3 to hold only the ten elements of the array. In Mark I FORTRAN, the resulting file has five values 1, 2,..., 5 preceding the ten elements of the array.

Examples

```
WRITE (I) A, B
READ (I) A, B
WRITE (2,30) ((A(I, J), I = 1, 10),J = 1, N), T
WRITE (3, TITLE)
READ (3, TITLE)
BACKSPACE N
IF(ENDFILE2) LOOP,EOS
READ (N, 12) VAL, COST, PRICE
REWIND 1
WRITE(3) "MONTHLY SUMMARY"
WRITE(3,12) ↑↑ , LNO, "REPORT TO DATE", F1, F2, F4, F7
READ(5, 20) LNO, (A(K), K = 1, M)
```

Temporary File Definition

$DATA - (Temporary File Definition). A single temporary file is available. It is obtained by including a $DATA line after the last statement in the entire program. If any lines follow the $DATA line, such lines are taken to contain initial values for the temporary file.

The $DATA line may contain as many as six names after the word "data". They are separated from each other by commas. Each name in a $DATA line must be the name of a saved item in the user's catalog. The $DATA line may not be continued since the continuation lines would be interpreted as containing initial values. The general form for the $DATA line follows:

$DATA name, name, ...

The names must be acceptable FORTRAN names which begin with a letter and contain only alphanumeric characters.

Any characters exceeding six are ignored since the time-sharing system only catalogs six character names.

Each line of each saved item is interpreted as containing initial values for the temporary file. These values are included after the values contained in any lines following the $DATA line. Values from lines of the first named item are included first, then those from lines of the second, and so on.

The space available for operations on the temporary file equals the space required for the initial values. The space available for entering the initial values in the temporary file depends on the size of the program compiled. A space large enough to contain about 17,000 characters is available for both the (compiled) program and the temporary file.

To make a temporary file of larger size than the space required for the initial values of the file, a saved item of appropriate size may be named in the $DATA line. The item may be a renaming of one of the library items of a different size or it may be any item of the right size.

Example (Numbers on the left are line numbers.)

```
 200   READ, N,(ARRAY (I), I = 1, N)
       . . .
1000   $DATA
1010   52, 10 * 1.2, 25 * .3, 15 * - .3, 2 * 0.
```

FILE REFERENCE AND OPERATIONS.

The temporary file is referenced by omitting the file reference in the statements:

```
READ format reference
WRITE format reference
IF (ENDFILE)
ENDFILE
REWIND
BACKSPACE
```

The temporary file may also be referenced by supplying a file reference whose value is zero in any of the permanent file statements.

The file operations using the temporary file are the same as those using the permanent files. Except for any initial values of a temporary file that are already saved, the temporary file values are unavailable when execution of the program stops. Any values written in the temporary file during the execution of the program will not be available after the execution stops unless they are also written in a permanent file.

Examples

```
READ, A, F, G
READ 16, B, D
READ TITLE
READ VAR, "G", TABLE1, TABLE3
```

```
READ, (A(I), I = 1, 10)

READ SOME, (B(K), K = 1, M, L)

IF(ENDFILE) MORE, EOS

REWIND

BACKSPACE

ENDFILE

WRITE "FIRST TABLE VALUE"

WRITE, (A(I), I = 1, N)

WRITE 16, B, D

WRITE TITLE
```

If no format reference is given, unformatted input and standard output format are assumed. The comma normally written after the format reference must appear after the statement word "read" or "write" to prevent the first item in the input/output list from being mistaken for the format reference. When the first item in an output list is a quotation, the comma after the word "write" may be omitted.

Reading from the temporary file is much faster than reading from either permanent files or from the terminal. Writing in it has limited usefulness, since whatever is written only in the temporary file will be lost when the program stops. For this reason, it is best used as a buffer for transmission between permanent files or between permanent files and the terminal.

10. FORMATTED INPUT/OUTPUT

FORMAT DEFINITION

Input/output may be performed by using a sequence of specifications called a FORMAT. The specifications indicate the method of conversion to be used between internal and external representation of values. For example, the external representation "35" can be converted to various internal values:

- Integer 35, treating "35" as a decimal integer
- Integer 29, treating "35" as an octal integer
- Integer 197, treating "35" as an alphabetic ($197_{10} = 0305_{8}$)
- Real .35, treating "35" as a decimal fraction
- Real 3.5, treating "35" as a mixed decimal

The format sequence consists of a parenthesized list of conversion specifications. They are usually written with a comma separating them, and except for quoted or counted characters, blanks may be used as desired.

The format sequence may be written in a FORMAT statement as discussed below, or it may be stored in an array. Since a format sequence is a sequence of characters, it can be transmitted alphabetically as input and received in an array. (Refer to the discussion of "Format Statements Read at Execution" on page 85.)

Individual characters in a format sequence can be referred to as array elements when the format is stored in an array. By this reference, the format can be changed during execution. A format sequence in a FORMAT statement, on the other hand, cannot be changed except as noted below in the discussion of the H-format on page 82. Also, refer to the discussion of "Variable Format Specifications" on page 85.

FORMAT STATEMENT

A FORMAT statement is always given a statement label so that the input/output statements may reference the contents of the format sequence by its label. The FORMAT statement consists of the word "format" followed by the format sequence written in parentheses.

Examples

```
27 FORMAT (I4, 4I13,)
HEAD1: FORMAT("NO. OF CASES TREATED")
AFORM: FORMAT(6(5E13.4), 2E13.4/)
```

A FORMAT statement may occur anywhere in a program. It is not executed, but its statement label may be referred to within control statements to direct control to subsequent statements.

FORMAT REFERENCE

In each of the four Mark I FORTRAN input/output statements, a format may be referred to.

INPUT format reference, list PRINT format reference, list	terminal
READ (file reference, format reference) list WRITE (file reference, format reference) list	permanent file
READ format reference, list WRITE format reference, list	temporary file

If the format reference is a number, it is interpreted as the label number of a FORMAT statement in the same subprogram. If the format reference is a name, it is interpreted as:

- The name of an array containing a format sequence
- The name of a label variable, the current value of which is the label of a FORMAT statement
- The label name of a FORMAT statement

A format reference name is interpreted as an array name only if the array has been previously declared.

NUMERIC FORMAT SPECIFICATION

Four types of conversion are available in Mark I FORTRAN for input or output of numeric values. Each type is indicated by a different format specification. The table which follows shows the kinds of conversion and how each is specified. The examples show typical number representations that can be transmitted to a file or the terminal using each type.

Format Specification	Name	External Form	Examples	Internal Form
Iw	Integer	Decimal whole number	49, -3	Integer
Fw.d	Fixed point	Decimal with decimal point	49., -.30, 87.351	Real
Ew.d	Exponent	Decimal with tens exponent multiplier	-1.5E+03, .257E-49 48E+12, .1E+05	Real
Gw.d	General	Decimal with or without tens exponent multiplier	87.351, .1E+05 -.30, 48.E+12	Real

The letters used in the preceding table have the following meaning:

w a positive integer giving the number of character positions in the external representation. It should be large enough to include all characters appearing in the representation:

- sign
- digits
- decimal point
- four character tens-exponent multiplier

In Mark I FORTRAN, w need not provide room for a sign unless the number is negative.

d a non-negative integer giving the number of digits in the fractional part. For F and E statements the fractional part will be rounded to a d significant figures.

For output, a number is considered to be right-justified within the field specified. Character positions to the left that are not used for the number representation are blank filled. The w specifications can thus be used to provide appropriate spacing between successive numbers in the same output record.

For input, a number is also considered to be right-justified within the field specified. But blanks anywhere within the field, to the left or right, or in the midst of the number representation are ignored.

Input is the same, whether an F, E, or G format specification is used. However, a scale factor specification affects only F and G type input.

Numeric Field Widening

In Mark I FORTRAN, when an I, F, E, or G format specification does not provide a wide enough field to contain all of the characters required to represent an output number, rather than suppress some of the characters, the field is temporarily widened, and all of the characters are transmitted.

This field widening may cause the values in one output record to be misaligned with values in other records being transmitted with the same format. Moreover, the field widening may cause the inclusion of too many characters in an output record. If this occurs, another record is automatically provided. No notification of field widening is given.

Note: With I or F format specification, too narrow a field width is often due to a value having a larger than anticipated number of digits. And, with either of these or E or G format specifications, the width specified often does not provide room for a minus sign because a negative value is not anticipated.

Another common omission is space for the multiplicative factor which is required only sometimes in a G-format.

Another record is automatically provided when a terminal output record either because of field widening or specification, exceeds the maximum of 72 characters allowed in a terminal output record. Rather than split a field contents across two records, the entire last field of the current record is deleted from it and placed at the beginning of the next record.

I-Format

General Form: Iw
nIw

where n represents the number of times the field is repeated
w represents the total number of characters within the field

Examples I5
2I7

INPUT. Input numbers to be converted under type I specifications may consist of from 1 to 6 decimal digits, with or without a preceding plus or minus sign. If no sign is provided, a number is considered to be positive.

The magnitude of a number must be less than (524288). Numbers exceeding this are received as ±524287.

The variables to receive values converted under type I specifications may be either integer or real. Integer numbers are automatically converted to real form before being stored as real variables.

Examples

If the next record in the temporary file held values represented as follows:

```
                    1         2         3
Column Nos. 12345678901234567890123456789 0
            020 30  -1009874  38 3
```

and it were read by the following FORTRAN statements:

```
    READ 10, I, A, J, K, B, M
10  FORMAT (2I3, I6, 3I4)
```

the values would be stored in memory as I=20, A=0.3x10^2, J=-100, K=9874, B=0.38x10^2, and M=3.

Note: Blanks within a field are not significant.

OUTPUT. Either integer or real values may be transmitted for I format output. Real values are automatically truncated to integers before conversion to output characters.

Examples

If the values I=-1263, J=200, A=0.207x10^2, and M=-2 were contained in memory and printed by the following FORTRAN statements

```
      PRINT 10, I, J, A, M
10    FORMAT (I3, 2I5, I3)
```

the numbers would appear within the following columns of the record:

```
                      1         2         3
Column Nos.  12345678901234567890123456789 0
             -1263  200   20 -2
```

Note that the width of the first field is extended from three to five characters in length to avoid truncation of the value of I.

F-Format

General Forms: Fw.d
nFw.d

where n represents the number of times the field is repeated
w represents the total number of characters within the field
d represents the number of digits in the fractional part

Examples
```
F12.2
3F10.2
```

INPUT. Input specifications for F, G, and E are exactly the same. The input is a real number from the terminal consisting of from 1 to 11 decimal digits with optional sign, decimal point, and exponent. Blanks are ignored. Digits after the first 11 are ignored except as place markers. If no decimal point appears, its position is indicated by the fraction width parameter, d. The decimal point is then assumed to be d places to the left of the end of the mantissa field. The mantissa field is the entire field unless an exponent appears, in which case, it is the part of the field to the left of the exponent.

Decimal scale factors are of four forms:

E±ee $±ee ±ee E ee

These supply the power of ten by which the number must be multiplied to produce the true value.

For example:
1.32E4 represents a number with the value 1.32x10^4
0.05-4 represents a number with the value 5.0x10^{-6}
7.5$4 represents a number with the value 7.5x10^4

The magnitude of a real number must be less than (approximately 5.0x10^{76}). Numbers which are larger are received as approximately y ± 5.0*10^{76}. A maximum of nine significant digits may be retained internally. The variables which are to receive values converted with type F specifications may be either integer or real. Real values are automatically truncated to integer before being stored as integer variables.

Example

If the next record in the temporary file held values represented as follows:

```
                    1         2         3         4
Column Nos.  12345678901234567890123456789012345678 90
             4720 146321  - 2.3     88.13   67 . 4
```

and the file was read by the following statements:

```
     READ 10, LNO,A, B, I, C, D
10   FORMAT(I4, 2F8.3, 2F5.1, F7.0)
```

The values would be stored internally as LNO = 4720
$A = 0.146321 \times 10^{3}$, $B = -0.23 \times 10^{1}$, I = 8, C = 0.13, $D = .674 \times 10^{2}$

OUTPUT. Variables which supply values to be converted with type F specifications may be either integer or real. Integer values are automatically converted to real before conversion to ouput characters. A maximum of eight significant digits are available in a real value. The field width, w, should include a space for a decimal point, a space for the sign if negative values can occur, as well as space for blanks between successive numbers.

Examples

If the values I=-200, $A=0.30266 \times 10^{2}$, $B=-0.1003 \times 10^{3}$, $C=0.1 \times 10^{1}$ and $D=-0.047 \times 10^{-2}$ were contained in storage and printed by the following FORTRAN statements:

```
     PRINT 10, I, A, B, C, D
10   FORMAT (F6.2, 2F8.2, F7.3, F10.3)
```

the numbers would appear within the following columns of the record:

```
                    1         2         3         4
Column Nos.  12345678901234567890123456789012345678 90
             -200.00  30.27 -100.30  1.000     -.000
```

Note that the width of the first field is extended from six to seven characters in length to provide room for the sign.

E-Format

General Forms: Ew.d
nEw.d

where n represents the number of times the field is repeated
w represents the total number of characters within the field
d represents the number of digits in the fractional part.

Examples

```
E12.2
3E10.2
```

INPUT. Input via E-format is the same as input via F-format. The input is a real number from the terminal consisting of from 1 to 11 digits with optional sign, decimal point, and exponent. Blanks are ignored. Digits after the first 11 are ignored except as place markers. If no decimal point appears, its position is indicated by the fraction width parameter, d. The decimal point is then d places to the left of the end of the mantissa field. The mantissa field is the whole field unless an exponent appears, in which case it is the part of the field to the left of the exponent.

The four forms an exponent may take are the same as in types F and G:

E±ee $±ee ±ee E ee

Examples

If a terminal transmitted information as follows:

```
                      1         2         3
Column Nos.  12345678901234567890123456789 0
              1.33E4  -434E-3    2.0   5
```

and it was read by the following FORTRAN statements:

```
      INPUT 10, A, B, I, C
10    FORMAT (E7.4, E9.2, E7.4, E5.1)
```

the values would be stored in memory as $A=0.133x10^{5}$, B=-.434, I=2, and C=0.50.

OUTPUT. Variables which supply values to be converted with type E specifications may be either integer or real. Integer values are automatically converted to real before conversion to output numbers.

Type E specifications have the advantage that information is not lost due to large variations in the size of numbers. Large, as well as small numbers may be made to fit into a relatively small field of the output record.

Examples

If the values $A=-0.14432x10^{9}$, $B=0.146x10^{1}$, and I=2341 were contained in storage and printed by the following FORTRAN statements:

```
      PRINT 10, A, B, I
10    FORMAT (E10.4, E10.2, E.12.4)
```

the numbers would appear within the following columns of the record:

```
                      1         2         3         4
Column Nos.  12345678901234567890123456789012345678 90
             -.1443E+09  .15E+01  .2341E+04
```

Note that the width of the first field is extended from ten to eleven characters in length to provide for the sign.

G-Format

General Forms: Gw.d
nGw.d

where n represents the number of times the field is repeated
w represents the total number of characters within the field
d represents the number of significant figures

Examples: G12.3
5G18.4

INPUT. The input via G-format is the same as input via F-format. The input is a real number from the terminal, consisting of 1 to 11 digits with optional sign, decimal point, and exponent. Blanks are ignored. Digits after the first 11 are ignored except as place markers. If no decimal point appears, its position is indicated by the fraction width parameter, d. The decimal point is then d places to the left of the end of the mantissa field. The mantissa field is the entire field unless an exponent appears, in which case, it is the part of the field to the left of the exponent.

The four forms an exponent may take are the same as in types F and E:

E±ee $±ee ±ee E ee

Examples

Refer to the input example under F conversion.

OUTPUT. The G-format is used to provide numbers having a specified number of significant figures no matter what the magnitude of the number. The d in the format specification:

Gw.d

indicates how many significant figures are to be provided. The number is right-justified in the space of w character positions.

When the number is either too large or too small to represent without supplying zeros before or after the decimal point, a multiplicative factor is used. The table below shows how G-format output uses either F- or E-format output depending on the magnitude of the number.

Magnitude	Representation
magnitude<0.1	Ew.d
0.1≤magnitude<1.0	Fw.d
1.0≤magnitude<10.	Fw.(d-1)
10.≤magnitude<100.	Fw.(d-2)
. . .	
10^{d-2}≤magnitude<10^{d-1}	Fw.1
10^{d-1}≤magnitude<10^{d}	Fw.0
10^{d}≤magnitude	Ew.d

Notice that the w specified as the field width should allow four spaces for the multiplicative factor.

If a scale factor (refer to "Scale Factors" below) is specified, it affects only cases where the E-format conversion is used.

Example

When the internal values A = 34.756, B = .004763, C = .5, D = -3.777, I = -936 are transmitted for output using the statements:

```
   PRINT GG, A, B, C, D, I
GG:FORMAT(G8.4, G12.3, 2G6.3 , G9.2)
```

the output record would have the appearance shown below.

```
                    1         2         3         4
Column Nos.  12345678901234567890123456789012345678901
               34.76   .476E-02   .500 -3.78-.94E+03
```

Scale Factors

Scale factors may precede F, E, or G format specifications. A scale factor is written using a signed integer followed by the letter "P" which stands for "power." If the sign is omitted, a positive integer is assumed. If the integer is omitted, an integer +1 is assumed.

Examples

```
3PF12.2
0P2E10.1
-2P4F10.6
PE14.3
P3G16.5
```

When used with a type F specification, the scale factor gives the power of ten by which values are to be multiplied prior to transmission. For example, if numbers in a file represent thousands, values like 4,35, or .3 can be stored in the file and converted to 4000, 35000, 300 by input transmission using a 3P scale factor. Similarly, transmission to a file or the terminal can use a -3P scale factor to represent values in terms of thousands.

When a scale factor is used with an E specification on input it has no affect; on output the value is not changed. Instead the scale changes the position of the decimal and adjusts the exponent.

When the scale factor is used with a G format on input the value is mulitiplied by the appropriate factor of ten; however, on output it follows the rules for E or F type format according to the type of representation used.

The scale factor remains in effect until turned off with a 0P specification.

Whenever a scale factor is indicated for one format specification by writing the scale factor in front of that format specification, it also is indicated for all format specifications between that one and another scale factor. For example, in the format

(PE14.3, F10.2, G16.2, 0P4F13.2)

the 1P scale factor applies to the E14.3 format specification and also to the F10.2 and G16.2 format specifications. The 0P scale factor restores the normal scaling ($10^0 = 1$) to the subsequent specification 4F13.2.

Examples

If a terminal transmitted information as follows

```
                    1         2         3
Column Nos.  12345678901234567890123456789 0
              200  3.3  9726 83.41
```

and it was read by the following FORTRAN statements

```
      INPUT 10,A,B,C,D
10    FORMAT (-2PF5.1,1PF5.2,2PE6.0,-PE6.0)
```

the values would be stored in memory as $A=0.2x10^0$, $B=0.33x10^2$, $C=0.9726x10^4$, and $D=0.8341x10^2$.

Note that the scale factor has no affect on E-format input transmission.

If the values $A=0.6328x10^2$, $B=-0.926x10^{-1}$, $C=0.1x10^3$, and $D=0.45102x10^{-2}$ were contained internally and printed by the following FORTRAN statements

```
      PRINT 10,A,B,C,D
10    FORMAT(-1PF6.2, 2PF8.1, 1PE10.2, -1PE10.4)
```

the numbers would appear within the following columns of the record:

```
                    1         2         3         4
Column Nos.  123456789012345678901234567890123456789 0
              6.33    -9.3  1.00E+02 .0451E-01
```

Note that the scale factor does not change values of E-format output transmission, only the position of the decimal point in the representation.

OCTAL FORMAT SPECIFICATION

O-Format

General Forms: Ow
nOw

where w represents the number of characters within the field
n represents the number of times the field is repeated

Examples

```
10    FORMAT(05,3014)
```

Notes

1. Only blanks and octal digits may appear within type O fields.
2. The maximum number of octal digits and imbedded blanks allowed within a type O field is 7 if the value is to be assigned to an integer variable, or 14 if the value is to be assigned to a real variable. If more are provided for input,the right-most are transmitted.
3. The seventh and fourteenth digits from the right should be less than 4. High-order portions of digits larger than this are lost.
4. Variables which supply or receive values to be converted with type O specifications may be either integer or real.
5. Integer values produce 7 octal digits. Real values produce 14 octal digits.
6. Leading zeros are not suppressed.
7. Blanks are treated as zeros.

Examples

If the next record of the temporary file contained:

```
                        1         2
Column Nos.   12345678901234567890123456 7
              0020  64  773777777 7777766
```

and it were read by the FORTRAN statements

```
      READ 10, I, A, B, K
10    FORMAT(2O4, O11, O8)
```

the values would be stored in memory as follows:

LOCATION	VALUE
I	0000020_8
A	0000000_8
A+1	0000064_8
B	0000077_8
B+1	3777777_8
K	3777766_8

If the values of $A=0.1x10^1$, $I=52_{10}$, and $K=524287_{10}$ were contained in memory and printed by the following FORTRAN statement:

```
      PRINT 10, A, I, K
10    FORMAT (O15,2O8)
```

the numbers would appear within the following columns of the record:

```
                    1         2         3
Column Nos.  12345678901234567890123456789 01
              00060000000000 0000064 1777777
```

ALPHABETIC FORMAT SPECIFICATION

Two types of specifications are available for input and output of alphabetic (Hollerith) information: H and A. Both of these specifications transmit alphabetic information between storage and terminal devices. The difference between the two is that information transmitted under control of type A specification is assigned to a variable or array but information transmitted under type H is contained or received in a FORMAT statement or array.

H-Format

General Form 1: wHxxx...x

where w represents the total number of characters, x...x, which follow the H, including blanks.

Example
```
14HEND OF PROGRAM
16H"END OF PROGRAM"
```

General Form 2: "xxx...x"

Example
```
"END OF PROGRAM"
" 'END OF PROGRAM' "
```

Notes

1. Type H specifications in FORMAT sequences may be written within quotation marks instead of with a count of the number of characters followed by an H and then the characters.

2. Type H specification in a FORMAT statement, whether counted or quoted, may be continued on the next statement line. In this case, the carriage return, line feed, intervening comment lines (if any), line number, and blanks up to the continuation mark are not included in the counted or quoted characters.

3. Commas which normally separate specifications with a FORMAT sequence are not necessary following a type H field. They may be used if desired.

4. When used with input, the characters following the type H specification are replaced by an equal number of characters from the input record. Only the wH form may be used for this purpose.

5. When used with output statements, the type H specification causes the Hollerith information to be inserted into the output record at the place called for in the FORMAT statement.

Examples

If the value of A were $0.5x10^3$ and the following output statement were executed

```
      PRINT 10, A
10    FORMAT (22HLARGEST VALUE OF A IS  ,F8.1)
```

The following line would be transmitted to the terminal:

```
                    1         2         3         4
Column Nos.  12345678901234567890123456789012345678901
             LARGEST VALUE OF A IS    500.0
```

If a terminal input record was:

```
                    1         2         3         4
Column Nos.  12345678901234567890123456789012345678901
             SMALLEST VALUE OF A IS   -2.00
```

and it was transmitted using the same format statement, the FORMAT statement would become:

```
10    FORMAT (22HSMALLEST VALUE OF A IS, F8.1)
```

and A would become $-0.2x10^1$.

A-Format

General Forms: Aw
nAw

where w represents the total width of the field
n represents the number of times the field is repeated.

Examples
```
A7
2A2
24A4
```

Notes

1. The maximum number of characters which may be stored within an integer variable is three. If w is greater than three, only the three right-most characters are considered. If w is less than three, the characters are left-justified and filled with blanks.

2. The maximum number of characters which may be stored within a real variable is six. If w is greater than six, only the six right-most characters are considered. If w is less than six, the characters are left-justified and filled with blanks.

3. Variables which supply values for A specification conversion may be either integer or real. Integer variables supply a maximum of three characters. Real variables supply a maximum of six characters.

4. Transmission to field widths larger than three or six fills the field from the left with w-3 or w-6 blank characters. Fields with width of less than three or six characters receive the w left-most characters.

5. For input, blanks are supplied on the right of any characters included to fill to 84 characters. The program below, for example, can be used to receive a variable length string of characters.

```
INTEGER ALPHA (84)
PRINT, "NAME"
INPUT A1, ALPHA
A1:FORMAT (84A1)
```

Examples

If a terminal transmitted information as follows:

```
                    1         2         3
Column Nos  12345678901234567890123456789 0
            ERROR IS END   VALUES GONOGO
```

and it was read by the following FORTRAN statements:

```
      INPUT 10, A, I, K, B, C
10    FORMAT (A6, A2, A4, 2A9)
```

the values would be stored in memory as follows:

Variable	Alphabetic Value	Octal Value
A	ERR	0255151
	OR	0465160
I	IS	0316260
K	END	0254524
B	VAL	0652143
	UES	0642562
C	NOG	0454627
	O	0466060

If the value of A was stored as read in the example above and I = 524287, then the following FORTRAN statements

```
      PRINT 10, I, A
10    FORMAT ("I=" I8, A6)
```

would produce the following printed record:

```
                    1         2
Column Nos.  12345678901234567890
             I=   524287ERROR
```

FORMAT STATEMENTS READ AT EXECUTION

FORMAT sequences may be read as alphabetic values, and received in an array. The name of the array may then be used as a format reference in an input/output statement. Note that the array must be previously declared. Otherwise the format reference is taken to be the label name of a FORMAT statement.

The FORMAT sequence in an array is used exactly as one in a FORMAT statement. It must begin with an open parenthesis and end with a close parenthesis. The open parenthesis may be preceded by the word "format," or any other words, but this is not required because the first open parenthesis signals the beginning of the FORMAT sequence. And the close parenthesis ending the FORMAT sequence may be succeeded by any characters since these are not used as part of the FORMAT.

Example

The following program illustrates how a FORMAT sequence may be read during execution.

```
      DIMENSION J(6)
      INPUT 10, J
10    FORMAT (6A3)
```

If a terminal transmitted information as follows:

```
                    1         2
Column Nos.  12345678901234567890
             (E15.4,F10.2,I8)
```

the FORMAT sequence in that record can be used within the program by referring to J in a format reference.

Type X

General Form: nX
where n represents an unsigned, decimal integer.

Example: 5X

Notes

1. The type x specification is used to skip characters of an input record and insert blank characters in an output record.

2. The count n, gives the number of characters to be skipped or the number of blanks to be inserted.

VARIABLE FORMAT SPECIFICATIONS

Two additional types of format specifications are available that allow varying the format during execution.

T-Format

A T-specification indicates that the next item in the list is to be interpreted as an alphabetic character to replace the T in the format. For example, if the list supplies an "E" the "E" is then used in place of the "T" in the format specification. The revised format specification is then used for the next list item.

*-Format

A * specification is used in the same way to indicate that the integer value of the next item in the list replaces the * in the format. For example, the statements:

```
      PRINT 10, J, IW, IDP, A
10    FORMAT (T*.*)
```

could be used to print the real variable A with F16.5 format by giving statements J = "F"; IW = 16, IDP = 5 and with E20.6 by giving statements J = "E"; IW = 20; IDP = 6.

FORMAT-LIST CORRESPONDENCE

When a formatted input/output statement is first executed, the first format specification in the referenced format sequence is used to transmit the first item in the input or output list. Transmission is to or from the beginning of the next record. If the item has multiple elements, each element requires a format specification.

A format specification may be prefixed with a count that indicates for how many elements it is to serve. When the list contains more elements than the first format specification is to serve, the next format specification is used for the next element.

When either the H-type and X-type format specification is encountered, it is applied but does not serve for any list element. (Refer to "H Format" on page 82, and "X Format" on page 85.)

When all list elements are served, transmission ceases. If there are any format specifications that have not been used, they remain unused.

When all format specifications have been used, and there remain list elements to be served, the sequence is repeated. The first format specification in the sequence is used again, this time to serve the next list element. If the sequence contains groups of format specifications, described below, the entire sequence is not repeated.

FORMAT SPECIFICATION GROUPS

A format sequence may contain groups of format specifications. The group is written by enclosing one or more format specifications in parentheses. A count may prefix the group to indicate how many times it is to be used before subsequent specifications in the sequence, if any, are to be used.

Examples

FORMAT (I6, F10.4, 2(E20.8, A6))
indicates that the format specifications are to be used in the order: I6, F10.4, E20.8, A6, E20.8, A6

FORMAT (F16.3, 3X, 3(I4, A4), (E20.8))
indicates that the format specifications are to be used in the order: F16.3, 3X, I4, A4, I4, A4, I4, A4, E20.8

If a group is included in a format sequence and all format specifications have been used without exhausting the input/output list, the last group in the sequence, and not the entire sequence, is repeated. In the second example, the group with the single specifications E20.8 is repeated.

Example

FORMAT (3I10, 2(F16.3), A6)
indicates that the format specifications are to be used in the order: I10, I10, I10, F16.3, F16.3, A6 and thereafter, F16.3 is used since it is the last group in the sequence.

MULTIPLE RECORD FORMAT

In a format sequence both the slash and right parentheses characters indicate the end of a record. If items of a list remain to be considered at that point, the format repeats from the last left parenthesis.

The statement

```
      PRINT M, (A(I), I = 1, 23)
M:    FORMAT (E20.8, 2F12.4/(5E20.8))
```

provides for one record of information with the format:

```
E20.8, F12.4, F12.4
```

followed by records of information with the format:

```
E20.8, E20.8, E20.8, E20.8, E20.8
```

The statement

```
      PRINT MM, (A(I), I = 1, 27)
MM:   FORMAT (E20.8, F10.2/3E20.8)
```

would provide alternate records of information:

```
E20.8, F10.2
E20.8, E20.8, E20.8
E20.8, F10.2
E20.8, E20.8, E20.8
    and so on
```

If the sum of the field specified for a terminal output record exceeds 72 characters, another record is automatically provided. It holds only the excess from the previous record, and not any fields from the next specified record. (Refer to "Numeric Field Widening" on page 73.)

11. MONITOR LINES

In Mark I FORTRAN, four types of monitor lines are provided. Two of these have already been discussed:

- $FILE -- used to refer to permanent files
- $DATA -- used to obtain the temporary file

(Refer to pages 63 and 68 respectively.)

This chapter describes the remaining two monitor lines:

- $USE -- used as an aid to program composition
- $CHAIN -- Provides for execution of linked programs
- $OPT -- provides options for compiling and executing programs

$USE

The $USE line may be used to compose a program from statements saved separately. A subprogram, for example, saved separately, in one's own catalog or in the library, may be referenced in a $USE line to include that subprogram in one's own program. In addition to a subprogram, the items referred to in a $USE line may be statements, such as COMMON declarations. In a punched card environment, such statements would be reproduced several times and included with each of several subprograms to ensure that all subprograms declare common storage identically.

The $USE line must contain only a single name. The name must be the name of an item saved either in one's own catalog or in the library. To indicate a library name, append one or more asterisks to the name:

```
$USE name   -an item saved in one's own catalog
$USE name*  -an item saved in the library
```

The name must be an acceptable FORTRAN name. It must begin with a letter and contain only alphanumeric characters. Only the first six characters given are used, since the operating system catalogs items by names no longer than six characters. Note that the operating system treats leading or embedded blanks as significant, but that these are ignored in FORTRAN.

Do not place any other statements on the same line with a $USE statement. Any entry following the first name will be considered as a comment.

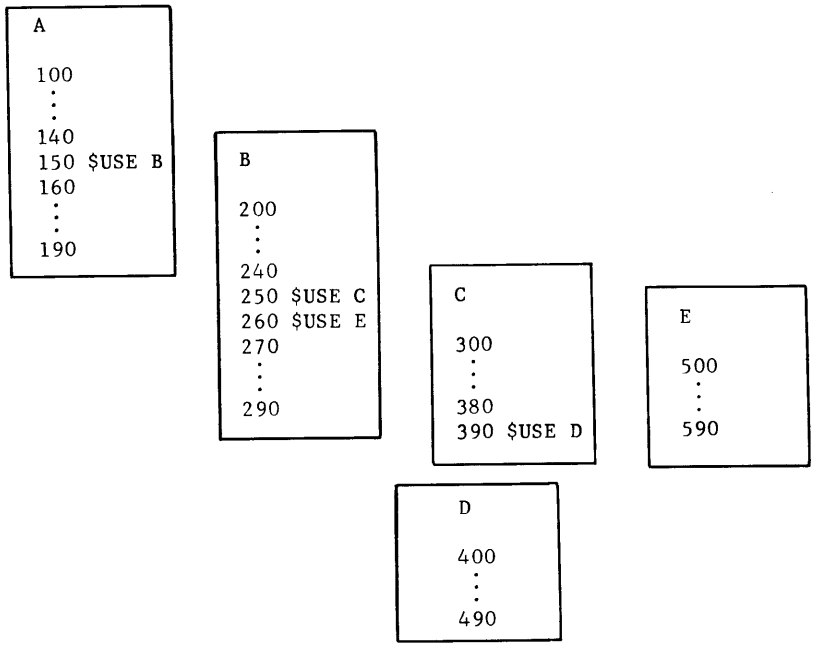
When a $USE line is encountered, the Mark I FORTRAN system interrupts the compiling of statements in the present item and continues compiling with the first statement in the saved item. When all the statements in that item have been compiled, compiling resumes

with the first statement on the line after the $USE line in the original item. It is as though the statements in the saved item were inserted in the original item in place of the $USE line.

Compiling does not mean executing. In the Mark I FORTRAN system all statements are compiled to an executable program. If there are no errors detected in the statements, the executable program that has been compiled is executed. This two-phase operation--compilation and execution--occurs every time the operating system command RUN is given.

The $USE line simply permits the statements that are compiled to be in more than one saved item. It does not allow the executable program to be saved and then included with a newly prepared executable program.

An illustration of how $USE lines provide for program composition from several saved items follows.



When the $USE lines occur in the places shown in the illustration above, statements are compiled in the following order:

100-140	(from A)
200-240	(from B)
300-380	(from C)
400-490	(from D)
500-590	(from E)
270-290	(from B)
160-190	(from A)

A maximum of six levels of $USE statements may be used. In the example illustrated above, four levels are used.

In the example above, each saved item from which the program is composed has different line numbers. This is advisable, since an exception or error report made by the FORTRAN system during execution refers only to the line number of the statement and not to both line number and name of saved item.

During compilation, the names of the saved items are transmitted to the terminal as compilation begins or resumes with the statements in each saved item. If composition error is detected during compilation, the transmitted name points out in which saved item the erroneous statement is detected. In the above example, the following messages would occur on the terminal in the order shown:

```
IN B
IN C
IN D
IN B
IN E
IN . FIRST
```

The last message indicates a resumption of compiling in the item in which compilation began (the name of which is not known).

$CHAIN

$CHAIN allows the user to link any number of programs together so that they run consecutively without further intervention. In essence, the command $CHAIN achieves the same result as entering OLD and RUN on the teletypewriter. $CHAIN is employed as a statement in the program which calls the next program to run. Hence,

```
100 $CHAIN PROG2
```

will terminate the program in which the command appears and cause the program named PROG2 to run. PROG2 itself may have a $CHAIN command and may call a program which also has a $CHAIN command, and so on. There are, for all practical purposes, no limits to the number of programs that may be chained together. Core storage is only reserved for the program currently running and is not affected by the programs waiting to be called by a $CHAIN command. The $CHAIN command must not be the first executable statement in the program.

$OPT

The monitor supplied with Mark I FORTRAN provides a number of options for compiling and executing programs. These options are obtained by writing a monitor line in the following format. The line begins with the word $OPT and continues with one or more words denoting options. An option has effect on the next and subsequent lines until cancelled by a reversing option order or by the end of the program.

The words used to order options are listed below. A more complete description follows the list. Although full words are shown in the list, only the first three characters of each word are significant. Abbreviations may be used. For example, INT serves as well as INTEGER, SUM as well as SUMMARY.

Options Available

$OPT SS	causes all subscripting to be checked for conformance with declared dimensions.
$OPT NO SS	causes subscripting not to be checked.
$OPT REAL	causes all names to be assumed real.
$OPT INTEGER	causes all names to be assumed integer.
$OPT MIXED	causes names to be assumed real or integer in the usual FORTRAN manner.
$OPT IFF name	causes compilation of subsequent lines to be suspended if there is no previous mention of the name given after IFF. If no name is given, compilation is unconditionally suspended.
$OPT*	causes resumption of compilation if it has been suspended, either conditionally or unconditionally
$OPT SIZE	causes the size of the part of the program so far compiled to be announced.
$OPT TIME	causes the time spent so far in compiling the program to be announced.
$OPT SUMMARY	causes a summary of each subprogram to be listed.
$OPT SOURCE	causes each line of the program as well as the summary to be listed.

The qualifier "NO" before SOURCE, prevents the program from being listed.

Subscripting Checking — $OPT SS, $OPT NO SS

Subscripted assignments are always checked to see that assignment outside of dimensioned storage is not made. Furthermore, all subscripting of arrays declared to have three or more dimensions is always checked for conformance with the declared dimensions. For example, for a mention A(I, J, K) when the dimensions A(12, 7, 2) are given, an error indication is transmitted unless $1 \le I \le 12$, $I \le J \le 7$, $1 \le K \le 2$. The SS option merely applies this checking to all arrays dimensioned after the option is ordered, whatever their dimensions are.

The NO SS option discontinues the checking for one- or two-dimensioned arrays declared after the NO SS option is given. It does not discontinue the checking on subsequent mentioned arrays previously declared.

Names are implied to be real unless their initial letter is I, J, K, L, M or N; in which case they are implied to be integer. This implication is overridden by the REAL or INTEGER options and restored by the MIXED option. This only applies for names whose first mention is subsequent to the option. Names can be declared to be real or integer by mentioning them in REAL or INTEGER statements. Such a declaration overrides an implication based on initial letter or an assumed mode.

The mode options have no effect on constants or intrinsic function names.

Conditional Compilation — $OPT IFF; $OPT *

Parts of a program may be effectively deleted by means of the IFF option. The deletion can be unconditional or conditional, depending upon whether prior mention of the name has been made of the word appearing after IFF.

The deleted part consists of the lines after the IFF line up to and including a deletion end line. The deletion end line is written $OPT or with punctuation after the word OPT, for example $OPT ****** or $OPT //////.

Instructions are not compiled for the deleted lines. Errors are not detected in the deleted lines, nor are the lines listed, regardless of what listing has been requested. Monitor lines are ignored, up to the $OPT*.

If a name is written after the word IFF, deletion is conditional, depending on prior mention of the name. If the name has been mentioned in the same external subprogram (SUBROUTINE, FUNCTION, or MAIN PROGRAM) in lines prior to the IFF line, the IFF and the deletion end line are ignored, and the lines between them are compiled in the usual way. If the name has not been previously mentioned in the same external program, however, the lines between are deleted. To be recognized, the name must be a variable or array name. Constants, labels, or external subprogram names are treated as though not previously mentioned, whether they are or not.

Use of a name after the word IFF is not considered a mention of the name by subsequent IFF lines, nor, of course, are mentions of a name within deleted lines.

When no name is written after the word IFF, subsequent lines are unconditionally deleted.

Size and Time Announcements — $OPT SIZE, $OPT TIME

The size announcement contains three decimal integers indicating the number of machine words used so far for the following three purposes.

1. The number of words used for instructions, undeclared storage, and constants.
2. The number of words used for dimensioned storage not declared to be COMMON.
3. The number of words used for COMMON storage.

The time announcement contains a single decimal number giving the number of 1/6 seconds spent so far in compiling the program.

For convenience, both the size and time announcements provide the number of the $-line to which the announcement is a response.

Compilation Summary $OPT SUMMARY, $OPT SOURCE

The source program is listed line by line.

The summary lists each name or constant referred to in each external subprogram. Names of intrinsic functions and of constants or subprograms whose first reference was in a previous subprogram are not listed. Names and numbers used as labels are colon suffixed. One or two words are printed after each name or number. The first gives the octal location assigned to the name or number and the kind of program element the name or number refers to. The second, if given, indicates that the name is dimensioned and in addition, gives one of the following. If the array is single dimensioned, the line has the size in octal of the array. Otherwise, the line has the octal location in the object program in which the array's dimensioning information is kept.

Each summary is preceded by a size legend and a count of the number of entire line comments in the subprogram being summarized.

The qualifier NO in front of either of the words, SOURCE, or SUMMARY, causes a cancellation of that listing request. By this means, various combinations of listings can be specified. The list below shows all combinations and how each is specified:

Stop listing source	$OPT NO SOURCE
Source, summary	$OPT SOURCE
Source	$OPT SOURCE, NO SUMMARY
Summary	$OPT SUMMARY

Unless a listing option is used, no listing is provided. But, once a listing option is used, subsequent listing options combine with it. For example, the order: $OPT SUMMARY requests only a summary listing if there has been no previous request. However, if the previous request was $OPT SOURCE, NO SUMMARY, then the summary and the source are requested.

Use $OPT NO SOURCE to cancel previous requests and return to no listing.

12. HANDLING OF ERRORS

Most programs contain errors. Some errors are detected during compilation, for example, a call to a subprogram whose definition is not included. Others are detected only during execution, for example, seeking the logarithm of a negative value. Many errors remain undetected because the Mark I FORTRAN system attends only to the form and not to the purpose of a program.

A program may be thought of as a sequence of written directions. Errors may be detected in the writing of the directions. Punctuation may be used incorrectly; names may be used inconsistently; numbers may appear where names should appear . . . and so on.

Other errors may be noted as the directions are being followed: the directions may ask for impossible or inconsistent actions. But because the directions are followed without considering their purpose, detection of many errors remain for the writer to discover.

COMPOSITION ERRORS

When a composition error is detected, it is reported and compilation continues. But when compilation is finished, execution of the compiled program does not start if there are any composition errors.

The Mark I FORTRAN system detects and accepts as correct many nonstandard FORTRAN idioms, for example, mixed mode expressions. However, about 130 ways of mis-writing are detected and reported.

In most cases, the miswriting is detected as the statement is being compiled and the mis-written statement is transmitted to the terminal. Underneath it, a line is transmitted that contains an up arrow (↑) pointing to the punctuation that is miswritten or that marks the end of a miswritten name or number. Before or after the arrow is a message, that may indicate in what way the statement is miswritten.

Examples

```
140 FAR (3) = 3.2/J
  - - - - ↑ UNDIM. ARRAY?

150 DO 25, INDEX = JOB + 4, N
ONLY COMMA - - - - - - - ↑

160 NAMELIST A, B, C, K
WHAT? - - - - ↑

170 COMMENT LINE HAS BLANK AFTER LINE NUMBER
COMMENT? - - - - - - - - - - - - - - - - - ↑

180 SUR = CHANGE (ABO (I+12)
UNPAIRED PAREN. - - - - - - ↑
```

Whenever a name is mentioned in a declaration, that name must not have been referred to before in the same external subprogram, except in another declaration. If a name is mentioned in more than one declaration:

- No statements other than declarations may intervene between these declarations.
- The name must be dimensioned in only one of these statements.
- The mode and equivalence, if declared, must be consistent with any previous mode and equivalence declarations.

Examples

```
200   SUM = ARRAY (I)
210   DIMENSION ARRAY (45)
NAME USED PREV. - - - - - ↑   (previous use of name)

220   COMMON FAULT (12)
230   FAULT (2) = 25.2        (nondeclaration intervenes)
240   INTEGER ANTY, FAULT
NAME USED PREV. - - - - - ↑

250   COMMON SELF (12), OTHERS (12)
260   EQUIVLAENCE (SELF (3), OTHERS)   (inconsistent with previous declaration)
NAME USED PREV. - - - - - - - - - - ↑
```

The message written before or after the up arrow may or may not indicate what is miswritten. If in doubt about what is wrong with the statement, consult first the list of composition error messages given below, and then, the statement checklist on page 108 for that kind of statement.

When a statement contains more than one miswriting, only the first error in the statement may be discovered. In many cases, scrutiny of the miswritten statement is continued to see if there are other composition errors in the same statement. But when further scruntiny might erroneously mark something as miswritten, the remainder of the statement is skipped.

In a few cases the miswriting is not detected until all statements in a group have been compiled. In such cases, the line number of the last statement in the group, or the first statement in the next group, is transmitted. After the line number, the name that is not used correctly is transmitted, and then a message indicating what is incorrect.

Example

```
AT LINE NO. 180:  AFT:  IS UNDEF.  --- Undefined statement name.

AT LINE NO. 180:  16:  IS UNDEF. - - - - Undefined statement no.

AT LINE NO. 400:  DART EXCEEDS STORAGE - - Not enough space for array

AT LINE NO. 8020:  FANT IS UNDEF. - - - Undefined external subprogram or perhaps
                                        an undeclared array
```

When the compiled program is too large, compiling continues with all counting restarted. Before continuing, a size legend is transmitted to indicate the size of the part of the program already compiled. This size check is made only at the end of each external subprogram.

Example

```
AT LINE NO. 600:  SUBPROG. EXCEEDS STORAGE
SIZE AT LINE NO. 600:  2614, 1528, 1158
```

EXECUTION ERRORS

There are two kinds of execution errors. With one kind, execution continues, and with the other, execution is aborted. If there is some chance that a standard remedial action will allow execution to continue usefully, the trouble is reported, the remedial action taken, and execution continued. When no standard remedial action is likely to work, however, the trouble is reported and execution stopped.

Messages for conditions which permit execution to continue contain the word WARN. Messages for conditions which stop execution contain the word QUIT.

The lists on the following page, show the word or words used to report each kind of trouble. In the case of WARN messages, the remedial action taken is described. Also listed are the two other messages, STOP and PAUSE, which have the same form as WARN and QUIT messages.

Examples

```
AT LINE NO. 430:  QUIT, BEYOND; RAN 7/6 SEC.
AT LINE NO. 140:  QUIT, FORMAT; RAN 41/6 SEC.

AT LINE NO. 960:  WARN, I ↑ I; RAN 25/6 SEC.
AT LINE NO. 1150:  WARN, WIDEN; RAN 35/6 SEC.

AT LINE 170:  STOP RUN; RAN 4/6 SEC.
AT LINE 1030:  STOP END; RAN 102/6 SEC.

AT LINE 50:  PAUSE (1); RAN 12/6 SEC.
?       (this is the request for terminal input)

AT LINE 315:  PAUSE ROW; RAN 62/6 SEC.
?
```

Execution Not Started — NO EXEC.

Message	Meaning
BAD	A $DATA statement not followed by file names or data lines may result in a BAD message. Otherwise, if the message occurs during the run of a program whose object length is less than the maximum allowed, a system error is indicated. Notify your Time-Sharing representative.
2nd MAIN PROG.?	A statement not preceded by a SUBROUTINE or FUNCTION. Previous statement has already been taken as the first of the main program so another such occurrence is marked erroneous.
CHAR. CONSTANT?	More than three characters quoted in a context where a longer quotation is not permitted. (Refer to page 10.)
COMMENT?	A string of characters longer than 30 characters. Might be a miswritten comment or perhaps punctuation has been omitted between two strings.
DO-END	An arithmetic statement is defined within a DO loop or a variable within a DO loop is undefined.
COMMON LARGER THAN FIRST	Present common declaration specifies more storage than the first common declaration in the program. (Refer to page 23.)
EQUIV. ODD	Real value declared equivalent to a value assigned an odd-numbered location. (Refer to page 26.)
EVIL DO-END	Refer to page 33.
EVIL SUBSCRIPT	More subscripts given for an array name than there are dimensions declared.
LATE	A $FILE line or $OPT PLOT placed too late in program to be processed. Move it closer to front of program.
MISSING	A label reference or do-parameter omitted.
NAME USED PREV.	Name appearing in a declaration has already appeared in a nondeclaration or in a declaration inconsistent with the present one. Or a DO-name already in use. (Refer to page 34.)

Message	Meaning
NO MAIN PROG.	All statements included within subprogram definitions; hence no statement with which to begin execution.
NO.?	String taken to be a numeric constant not written properly.
NOT A STMT.	Expression on left of equal sign or unrecognized use of comma.
NOT SAVED	$FILE, $DATA, or $USE name not in catalog.
ONLY COMMA	Punctuation other than comma used where only a comma is acceptable. Perhaps an expression used in place of a DO-parameter. (Refer to page 35.)
STMT. NAME USED PREV.	Name now used as statement name already in use as name of another statement or as label variable name.
SUBPROG.?	Name previously used in a way that made it appear to be a subprogram name. Name now used in a context where a subprogram name is not permitted. (Refer to page 13.)
SUBSCRIPT TOO BIG	A constant subscript given which if applied would cause storage outside dimensioned storage.
SYS. MALFUNCTION	Hardware trouble; hopefully temporary.
TOO BIG	Subprogram just ended exhausted space available for instructions, constants, and storage. (Size legend given afterwards shows size of excess.) If preceded by name, array requires more storage than available.
TOO MUCH	Too much nesting for the compiler to handle: More than 6 $USE ' started, not finished More 10 DO's started, not finished More than 14 function calls started, not finished More than 14 subscripts started, not finished More than 110 specifications started, not finished More than 15 dimensions specified for an array More than 4 repeated I/O lists started, not finished More than 20 file names in a $DATA statement
TOO MUCH TO REMEMBER	Too many names for the compiler to keep track of. Split into subprograms or reduce length or quantity of names: label, variable or array names.
UG1	More than 4500 instructions and words of source included data.
UG2	More than 64 edits required for instructions compiled in the last segment of about 1500 instructions.

Message	Meaning
UNDEF.	• If preceding name or number is colon suffixed, statement label referred to but no statement with that label given in subprogram just ended. • If an array name is misspelled. • If preceding name not colon suffixed, subprogram or entry name referred to but not defined. (May be an undimensioned array mistaken to be function call.)
UNDIM. ARRAY?	Name used in a way that makes it appear to be an array name but no dimension declaration has been given for it. (Refer to page 12.)
UNENDED QUOTE	End of statement seen before quotation begun in it has been concluded. Perhaps a missing continuation mark or too large a Hollerith count in FORMAT.
UNPAIRED PAREN.	Number of open and close parentheses must be the same.
USE FORTRAN	Attempt to use FORTRAN version not available to public.
WHAT?	Indicates a condition the compiler does not understand: • A statement or option word not recognized • Unrecognized punctuation • Operator or operand missing • Number used where only a name is understood • Name or expression used where only a number is understood • Miswritten octal constant • Zero dimension

Execution Aborted — QUIT

ARG. CALL	Attempt made to call other than a subprogram using a call to a dummy argument. Line number is of statement in which call is attempted.
ARG. STORE	Attempt made to store in other than a variable by means of a store in a argument. Line number is of statement in which storage is attempted.
BEYOND	Attempt made to store outside storage space. Line number is of statement in which storage is attempted.
FILE NO.	File reference error. Value of file number less than zero or exceeds number of files named in $FILE statements.

Message	Meaning
FORMAT	Format sequence miswritten. Line number is of statement that references the format.
GOTO	Computed GOTO error. Value of GOTO control variable is smaller than one or exceeds number of label references given in GOTO list.
NO FILE SPACE	Attempt to write on a file whose space is exhausted. Attempt is ignored. Line number is a WRITE statement.
NOTREAL	Output data supposed to be real not in correct form for a real number. Line number is of output statement. Data may be alphabetic or obtained from outside of storage space.
OUT OF DATA	Attempt to read from a file positioned at an end of data mark. Data read is that which follows end-of-data mark, if any, or is 524287 if no data follows. Line number is of READ statement.
SUBSCRIPT	Subscript error. Value of subscript smaller than one or exceeds size specified for dimension. Line number is of statement in which the subscript is given.
BAD DATA	Input data unrecognizable. Does not conform to format or requirements for unformatted input. Line number is of read statement.
SYS MALFUNCTION	Indicates a hardware disc error. Notify your Time-Sharing representative.
ILLEGAL DISC ADDR	Indicates a program or language complication. Notify your Time-Sharing representative.

Execution Continued		WARN
DIVBYZERO		Real division by zero. Zero is provided. Line number is of statement in which division occurs.
EXP		Raising of e to a magnitude greater than 176. Zero is provided if raising to a negative value. Largest possible number provided if raising to a positive value. Line number is of statement containing call to EXP.
I ↑ I	•	Raising of zero integer to zero or negative integer. Zero is provided.
	•	Raising of integer to integer power results in number with magnitude too large for an integer. Number of same sign and magnitude 524287 provided. Line number is of statement in which integer ↑ integer occurs.

Message	Meaning
INTEGER	Real value of too large a magnitude to be used as an integer. Number of same sign and magnitude 524287 is used. Line number is of statement in which one of the following occurs: • Storage as integer • Call to intrinsic function requiring integer argument • Subscript
LOG	Logarithm requested for zero or negative value. Zero or logarithm of positive number of same magnitude, respectively, is provided. Line number is of statement containing call to LOG.
LONG RECORD	Write of file record requested where record contains more than 71 data characters. Only the first 71 characters are written. Line number is of WRITE statement.
MOD	Call to MOD in which (argument one/argument two) is too large in magnitude. Quotient of largest possible magnitude and same sign is used. Line number is of statement containing call to MOD.
OVERFLOW	Any real calculation, not reported elsewhere, that results in too large a magnitude. The largest possible magnitude is used. Line number is of statement containing calculation.
↑R	• Raising to a real power of a negative real or integer. Positive real or integer of same magnitude is used. Raising of a real or integer zero to a zero or negative real. Zero is provided. • Raising of a real or integer to a real power results in too large or too small a number to be represented. If too large, largest possible magnitude is provided; if too small, zero is provided. Line number is of statement in which real ↑ real or integer ↑ real occurs.
R ↑ I	• Raising of a real zero to a zero or negative integer zero is provided. • Raising of a real to an integer power results in too large or too small a magnitude to be represented. If too large, largest possible magnitude is provided; if too small, zero is provided. Line number is of statement in which real ↑ integer occurs.
REALINPUT	Real input data has magnitude too large or too small to be represented. If too large, the largest possible magnitude is used; if too small, zero is used. Line number is of input statement.

Message	Meaning
SIN/COS	Sine or cosine requested for value with magnitude too large. Result of same sign and largest possible magnitude is provided. Line number is of statement containing call to SIN or COS.
SQRT	Square root requested for negative value. Square root of positive value with same magnitude provided. Line number is of statement containing call to SQRT.
UNDERFLOW	Any real calculation, not reported elsewhere, that results in too small a magnitude. A zero is used. Line number is of statement containing calculation.

Execution Stops	STOP
STOP END	Execution of the last statement in the main program. Execution of a RETURN statement in the main program when main program has not been called (via an ENTRY statement).
STOP xxx	Where xxx represent any three quotable characters. Execution of a STOP statement with xxx as the alphabetic value of the constant or variable written after the word STOP.

Execution Suspended	PAUSE
PAUSE xxx	Where xxx represent three quotable characters. Execution of a PAUSE statement with xxx as the alphabetic value of the constant or variable written after the word PAUSE. Execution resumed when carriage return is provided. If a value is entered before the carraige return, and xxx is the value of a variable, the entered value replaces xxx as the value of that variable. If the value entered is negative, it is stored as a numeric value. Otherwise, as an alphabetic.
INPUT DATA ERROR, RETYPE	Data read by the INPUT statement indicated by the line number was in error. Retype the record and execution will continue.

APPENDIX A. FORTRAN STATEMENT RULES

This appendix provides a list of rules for each of the Time-Sharing FORTRAN statements. The page number following each statement heading refers to the individual statement discussions.

1. Every Statement

    1. If first name is not statement word, statement is interpreted as arithmetic statement or internal function definition introduction.

    2. If comment line is written with blank after line number, it is interpreted as a statement.

2. Labeled Statement (page 17)

    1. Label number must be distinct from all other label numbers in the subprogram.

    2. Label name must be distinct from all other label names or label variable names in the subprogram.

    3. DO-end label must be properly nested with other DO-ends. (Refer to page 33.)

3. First Nondeclaration after Declaration Statements (page 22).

    1. Common storage must be no larger than first declared common storage. Dimensioned storage must be declared after largest common storage declaration.

    2. Declared storage must not exceed space available.

    3. The size declared for single array must not exceed 8191.

    4. Real name must not be assigned odd storage space by common and equivalence declarations.

4. Arithmetic Statement (page 21)

    1. There must be an equal sign after the first operand.

    2. Only a name or name followed by parenthesized list may precede equal sign. If name followed by list is previously declared an array, list is subscript list. Otherwise, statement is interpreted as introducing internal subprogram definition.

3. Every operator must be followed by an (possibly signed) operand.

4. The number of left and right parentheses must be equal.

5. Only characters used for operators may be used between operands.

6. Comma may occur only in parenthesized list.

7. Colon may occur only after statement label.

5. Internal Subprogram Definition (page 46)

1. Name must not be previously used.

2. There must be a (possibly empty) dummy argument list.

3. Dummy argument list must consist of only unsigned names separated by commas.

4. If an Arithmetic Statement Function, dummy argument list is followed by equal sign and expression. End of expression is end of ASF definition.

5. If an Internal Function, dummy argument list is followed by colon and first statement of function definition. End of Internal Function definition is first END statement (refer to "END," 13) or next internal subprogram definition.

6. ASSIGN label or label variable, TO label variable (page 30)

1. There must be a comma after label or label variable name in front of TO.

2. The word TO must be present.

3. The label variable named after TO must be distinct from label name.

4. The label variable named after TO must not be DO-end label.

7. BACKSPACE file reference (page 67)

The file reference must be either an unsigned integer or a variable, or it may be omitted.

8. CALL subprogram or entry (actual argument list) (page 37)

1. The subprogram or entry must be named after the CALL.

2. The argument list may be omitted.

9. mode qualifier COMMON array, array (dimension specification), variable, ... (page 23.)

   1. Any name previously used must be used only in program definition or in declaration statements; none but declaration statements may intervene.

   2. A name, if dimensioned here, must not have been previously dimensioned.

   3. If name is mode qualified, mode must agree with previous mode qualifications, if any.

   4. A name, if previously used in an equivalence, must not be used in an equivalence containing any other name declared to use common storage.

10. CONTINUE (page 34)

    Refer to "Labeled Statement," 2.

11. mode qualifier DIMENSION array (dimension specification), variable, ... (page 25)

    1. Any name previously used must be used only in program definition or in declaration statements; none but declaration statements may intervene.

    2. A name, if dimensioned here, must not have been previously dimensioned.

    3. If name is mode qualified, mode must agree with previous mode qualifications, if any.

12. DO label control variable = initial value, final value, increment (page 32)

    1. A DO-end label must be of a subsequent statement.

    2. Label name must be comma suffixed.

    3. Control variable must not be negated nor a subprogram name.

    4. Control variable must not be in use as DO name.

    5. Initial, final, and increment value must be unsigned constants or variables.

    6. Increment may be omitted. If it is, comma after final value must be omitted.

13. END (page 38)

    1. If END is followed by any character or characters, it is interpreted to be the end of the internal function definition. Otherwise, it is interpreted to be the end of external subprogram definition.

    2. (Refer to "End of Subprogram," 14 or "End of Entire Program," 15.)

14. End of Subprogram (External or Internal)

    1. Every number referred to in label reference must be defined by its use to label statement.

    2. Every name referred to in label reference must be defined by its use to label statement or as label variable. Check unformatted input/output statement for comma after statement word.

15. End of Entire Program

    1. Definition of all referenced external subprograms must be included. Omitting the dimension of an array may cause the array to be mistaken for an external function.

    2. There must be a main program.

16. ENDFILE file reference (page 65)

    File reference must be either an unsigned integer or variable; or it may be omitted.

17. ENTRY entry (dummy argument list) (page 49)

    1. Entry name must be distinct from intrinsic, external subprogram, and other entry names in the entire program. It must also be distinct from internal subprogram names in the present subprogram.

    2. Dummy argument list must consist of only unsigned names separated by commas; or may be omitted.

    3. If any name in the dummy argument list has been previously used in the program, the first such previous use must have been in a dummy argument list.

18. mode qualifier EQUIVALENCE (array, array element, variable, ...), (equivalence), ... (page 26)

    1. Parentheses must enclose each equivalence.

    2. An array element may be named using only a possibly signed integer, as single subscript.

    3. Any name previously used must be used only in program definition or in declaration statements; none but declaration statements may intervene.

    4. Only one name in an equivalence may be previously named in another equivalence.

    5. Only one name in an equivalence may be declared to use common storage.

    6. If name is mode qualified, mode must agree with previous mode qualifications, if any.

19. EXTERNAL subprogram, entry (page 28)

    1. Names must not be dimensioned.

    2. Any name previously used, must be used only in program definition or mode declaration statement.

20. FORMAT (format specification list) (page 71)

    If a counted or quoted Hollerith specification is included, it must be terminated before end of the list. Blanks are ignored.

21. mode qualifier FUNCTION external function (dummy argument list) (page 41.)

    1. Function name must be distinct from intrinsic, external subprogram, and entry names in the entire program.

    2. Dummy argument list must consist of only unsigned names separated by commas; or may be omitted.

22. GOTO label or label variable (page 29)

    If parenthesized list is given after label or label variable, comma must intervene.

23. GOTO (label, label variable, ...), control variable (page 29)

    1. List must contain only labels or label references separated by commas.

    2. Comma may or may not appear after list.

    3. Control variable must not be negated nor may it be subprogram name.

24. IF (expression) label reference, ... (page 30)

    1. Expression may contain equal sign. If it does, it must conform to requirements for arithmetic statements. (Refer to "Arithmetic Statements," 4.)

    2. One, two, or three label references must be given. There may be no more and no less.

25. IF(ENDFILE file reference) label reference, ... (page 31)

    1. File reference must be either unsigned integer or variable; or may be omitted.

    2. One or two label references must be given. There may be no more and no less.

26. INPUT format reference, input list (page 59)

    1. The format reference may be omitted.

    2. Comma must precede the input list, whether there is a format reference or not.

    3. Input list must not contain expressions, quotations longer than three characters, nor slew control characters (preceding up arrow, extra comma.)

    4. Repeated input must conform to requirements for repeated input/output lists. (Refer to "Repeated Input/Output List," 33.)

    5. All list items must be separated by commas.

    6. Input list may be omitted.

27. INTEGER array, array(dimension specification), external subprogram, variables, ... (page 41)

    1. Any name previously used must be used only in a program definition or declaration statement; none but declaration statements may intervene.

    2. Name, if dimensioned here, must not have been previously dimensioned.

    3. Mode must agree with previous mode qualifications for names, if any.

28. PAUSE constant or variable (page 38)

    Constant or variable must be unsigned; or it may be omitted.

29. PRINT format reference, output list (page 57)

    1. The format reference may be omitted.

    2. Comma must precede output list whether there is format reference or not. Comma may be omitted if first item in list is quotation.

    3. The output list may contain expressions. In expressions, every operator must be followed by an (possibly signed) operand. The number of left and right parentheses must be equal.

    4. Slew control characters (preceding up arrow and extra comma at end of a list) may be used.

    5. Quotations of any length may be used.

    6. Repeated output must conform to requirements of repeated input/output lists. (Refer to "Repeated Input/Output List," 33.)

    7. All list items must be separated by commas.

    8. There may be no list if there is a format reference.

30. READ format reference, input list (page 69)

1. The format reference may be omitted.

2. Comma must precede input list whether there is format reference or not.

3. Input list must not contain expressions, quotations longer than three characters, or slew control characters (preceding up arrow and extra comma).

4. Repeated input must conform to requirements for repeated input/output lists. (Refer to "Repeated Input/Output, "

5. All list items must be separated by commas.

6. Input list may be omitted.

31. READ (file reference, format reference) input list (page 60)

1. Format reference may be omitted. If omitted, the preceding comma also must be omitted.

2. File reference must be an unsigned constant or variable; or may be omitted if comma after it is given.

3. Input list must conform to requirements for input list given for READ above. (Refer to READ, 33)

32. REAL array, array (dimension specification), external subprogram, variable, ... (page 41)

1. Any name previously used must be used only in a program definition or a declaration statement; none but declaration statements may intervene.

2. A name, if dimensioned here, must not have been previously dimensioned.

3. Mode must agree with previous mode qualifications for names, if any.

33. Repeated Input/Output List (list item, list item, ..., control variables = initial value, final value, increment) (page 87)

1. Repeated list may occur only in INPUT, PRINT, READ, and WRITE statements.

2. The entire list must be parenthesized and comma separated from other items.

3. Nested repetition is confined to depth of four and has the form:
   ((repeated I/O list), control variable=initial value,
   final value, increment)

4. Repetition parameters must conform to requirements for DO parameters. (Refer to "DO," 12.)

34. RETURN (page 37)

If anything is written after word RETURN, statement is interpreted as arithmetic statement.

35. REWIND file reference (page 67)

File reference must be unsigned constant or variable; or may be omitted.

36. STOP constant or variable (page 38)

Constant or variable must be unsigned; or may be omitted.

37. SUBROUTINE subroutine (dummy argument list) (page 41)

1. Subroutine name must be distinct from intrinsic, external subprogram, and entry names in the entire program.

2. Dummy argument list must consist of only unsigned names separated by commas; or may be omitted.

38. WRITE format reference, output list (page 69)

1. The format reference may be omitted.
2. Comma must precede output list whether there is a format reference or not. Comma may be omitted if first item is quotation.
3. The output list must conform to requirements for output list given for WRITE ( ). (Refer to "WRITE," 39)

39. WRITE (file reference, format reference) output list (page 60)

1. The format reference may be omitted. If omitted, the preceding comma must be omitted.

2. The file reference must be an unsigned constant or variable; or may be omitted if comma after it is given.

3. The output list may contain expressions. In expression, every operator must be followed by a (possibly signed) operand. The number of left and right parentheses must be equal.

4. Slew control characters (preceding an up arrow and extra comma at end of list) may be used.

5. Quotations of any length may be used.

6. Repeated output must conform to requirements of repeated input/output lists. (Refer to "Repeated Input/Output List," 33.)

7. All list items must be separated by commas.

8. Output list may be omitted.

APPENDIX B. CHARACTER SET

The Mark I FORTRAN system uses part of the American Standard Communication Information Interchange (ASCII) character set. Figure 1 shows what part is used and gives the input, internal and output characters.

The Input Character column shows the terminal keys which, when depressed, produce internally the number shown in the Internal Octal column.

The Input ASCII column lists the ASCII code (in octal) which, when transmitted as input, will produce internally the number shown in the Internal Octal column.

The Output Character column shows what terminal character will be displayed or what action will be displayed or what action will be performed when the number in the Internal Octal column is transmitted for output.

The Output ASCII column lists the ASCII code (in octal) which is transmitted to the terminal when the number shown in the Internal Octal column is output.

Depressing any terminal keys or transmitting any ASCII input codes other than those listed, does not produce any internal number that can be used within a Time-Sharing FORTRAN program.

INPUT		INTERNAL	OUTPUT	
CHAR	ASCII	OCTAL	CHAR	ASCII
0	060	00	0	060
1	061	01	1	061
2	062	02	2	062
3	063	03	3	063
4	064	04	4	064
5	065	05	5	065
6	066	06	6	066
7	067	07	7	067
8	070	10	8	070
9	071	11	9	071
'	047	12	'	047
:	072	13	:	072
(	050	14	(	050
;	073	15	;	073
=	075	16	=	075
#	043	17	#	043

Figure 1. Time-Sharing FORTRAN Character Set
(Characters do not include parity.)

Characters given in this table are exclusively of parity.

INPUT CHAR	INPUT ASCII	INTERNAL OCTAL	OUTPUT CHAR	OUTPUT ASCII
%	045	17	#	043
\	134	17	#	043
+	053	20	+	053
&	046	20	+	053
A	101	21	A	101
B	102	22	B	102
C	103	23	C	103
D	104	24	D	104
E	105	25	E	105
F	106	26	F	106
G	107	27	G	107
H	110	30	H	110
I	111	31	I	111
C/G(bell)	007	32	(bell)	007
!	041	33	.	056
.	056	33	.	056
"	042	34	"	042
?	077	35	?	077
<	074	36	<	074
carriage return (cannot input)	–	37	(cr) (lf)	015/012
–	055	40	–	055
J	112	41	J	112
K	113	42	K	113
L	114	43	L	114
M	115	44	M	115
N	116	45	N	116
O	117	46	O	117
P	120	47	P	120
Q	121	50	Q	121
R	122	51	R	122
C/V	026	52	–	177
$	044	53	$	044
*	052	54	*	052
cannot input	–	55	eom	177
>	076	56	>	076
↑	136	57	↑	136
(space)	040	60	space	040
/	057	61	/	057
S	123	62	S	123
T	124	63	T	124
U	125	64	U	125
V	126	65	V	126
W	127	66	W	127
X	130	67	X	130
Y	131	70	Y	131
Z	132	71	Z	132
		72	if	012
,	054	73	,	054
)	051	74	)	051
[	133	75	[	133
]	135	76	]	135
–	–	77	fill	177

APPENDIX C. TABLE OF INTRINSIC FUNCTIONS

The modes of arguments are automatically adjusted to the required argument mode for each library function. The mode of the result is automatically converted to an integer value, if used in an integer expression (as a subscript expression for example), or to a real, if used in a real expression.

Table of Intrinsic Functions for Time-Sharing FORTRAN

Name	No. of Arguments and Assumed Mode	Result Mode	Definition
ABS	1 Real	Real	Absolute Value of argument
ABSF	1 Real	Real	
XABSF	1 Integer	Integer	
IABS	1 Integer	Integer	
ATAN	1 Real	Real	Principal angle in radians whose tangent is argument
ATANF	1 Real	Real	
COS	1 Real	Real	Cosine of angle in radians
COSF	1 Real	Real	
EXP	1 Real	Real	e raised to the given power
EXPF	1 Real	Real	
FIX	1 Real	Integer	Given real converted to an integer
IFIX	1 Real	Integer	
XFIXF	1 Real	Integer	
FLOAT	1 Integer	Real	Given integer converted to a real
FLOATF	1 Integer	Real	
AINT	1 Real	Real	Sign of argument times largest integer less than or equal to argument in magnitude
INTF	1 Real	Real	
LOG	1 Real	Real	Natural Logarithm of argument
LOGF	1 Real	Real	
ALOG	1 Real	Real	

Table of Intrinsic Functions for Time-Sharing FORTRAN, Continued

Name	No. of Arguments and Assumed Mode	Result Mode	Definition
AMAX1	≥2 Real	Real	Maximum of arguments
MAX1F	≥2 Real	Real	
MAX1	≥2 Real	Integer	
XMAX1F	≥2 Real	Integer	
AMAX0	≥2 Integer	Real	
MAX0F	≥2 Integer	Real	
MAX0	≥2 Integer	Integer	
XMAX0F	≥2 Integer	Integer	
AMIN1	≥2 Real	Real	Minimum of arguments
MIN1F	≥2 Real	Real	
MIN1	≥2 Real	Integer	
XMIN1F	≥2 Real	Integer	
AMIN0	≥2 Integer	Real	
MIN0F	≥2 Integer	Real	
MIN0	≥2 Integer	Integer	
XMIN0F	≥2 Integer	Integer	
AMOD	2 Real	Real	Remainder on dividing argument 1 by argument 2
MODF	2 Real	Real	
MOD	2 Integer	Integer	
XMODF	2 Integer	Integer	
RND	1 Real	Real	1. If arg = 0, provides next in sequence of pseudo-random numbers uniformly distributed, $0 < n \leq 1$ 2. If arg >0, initiates a new sequence and provides a number as above; starting value of sequence depends on arg 3. If arg <0, as above except starting value chosen arbitrarily
SIGN	2 Real	Real	Magnitude of argument 1 with sign of argument 2
SIGNF	2 Real	Real	
ISIGN	2 Integer	Integer	
XSIGNF	2 Integer	Integer	
SIN	1 Real	Real	Sine of angle given in radians
SINF	1 Real	Real	
SQRT	1 Real	Real	Square root of argument
SQRTF	1 Real	Real	
TIME	1 Real	Real	1. If arg < 0, gives elapsed chargeable time for execution (including compilation). 2. If arg ≥ 0, gives hours since midnight.

APPENDIX D. SAMPLE PROBLEMS

To illustrate the versatility of Time-Sharing FORTRAN, samples of scientific/engineering and business problems follow.

A scientist or engineer may need to evaluate a series for several sets of arguments. He could enter his program as follows. Note that no format specifications are given for input or output.

```
10' FORTRAN PROGRAM TO EVALUATE THE SERIES:
20' T(RHO,THETA)=100/PI↑3*
30' (RHO/12*SIN(THETA)-1/5↑3*(RHO/12)↑5*SIN(5*THETA)+
40' 1/9 3*(RHO/12)↑9*SIN(9*THETA)-+...)
50
60
100  PRINT"RHO AND THETA (BOTH IN RADIANS)=",
110  INPUT,RHO,THETA
120  PRINT,↑↑"FOR RHO=",RHO;PRINT"  THETA=",THETA;PRINT,↑"   TEMP=",
130  ALMOST=1E-6
140  TERMS=0;X=RHO/12;X4=X↑4;ANGLE=THETA;FACTOR=SIGN=1 'INITIALIZE
150  DO LOOP,I=1,40
160  IF((TEST=X/FACTOR↑3)-ALMOST) ENOUGH
170  TERMS=TERMS+SIGN*TEST*SIN(ANGLE) 'ACCUMULATE
180  X=X*X4;ANGLE=ANGLE+4*THETA;FACTOR=FACTOR+4 'INCR. BY 4
190 LOOP: SIGN=-SIGN 'ALTERNATE SIGN
200 ENOUGH: PRINT,100/3.1415926↑3*TERMS
210  PRINT↑↑↑↑↑
```

After the engineer types RUN, the programmed message entered in line 100 above would be printed, followed by a question mark (?) requesting input data. He would enter his data followed by a carriage return.

```
RHO AND THETA (BOTH IN RADIANS)=?10,.7
```

The answers would be typed as follows:

```
FOR RHO=      10.00
  THETA=        .70

   TEMP=       1.735
```

An accountant might wish to calculate a 12-month moving average. An example of a program which performs this calculation follows. Data for the program is included with the program. The table of moving totals and averages prepared by the program for this data is also shown.

```
100' FORTRAN PROGRAM TO CALCULATE 12 MO. MOVING AVERAGES
110
120  REAL MONTHLY VALUES(12,2)
130  READ,(MONTHLY VALUES(I,2),I=1,12)
140  DO FIRST YEAR, I=1,12
150  READ,MONTHLY VALUES(I,1)
160 FIRST YEAR: TOTALS=TOTALS+MONTHLY VALUES(I,1)
170  PRINT,  ↑↑ "    MONTH    VALUE    MOVING TOTAL    MOVING AVG."
180  PRINT,  ↑  "    FIRST YEAR =    ",TOTALS,↑
190  DO ALL,1=1,12
200  TOTALS=TOTALS-MONTHLY VALUES(I,1)+MONTHLY VALUES(I,2)
210  PRINT, I,MONTHLY VALUES(I,2),TOTALS,
215  PRINT ALL,TOTALS/12
216 ALL: FORMAT(F12.2) 'FINISH DO LOOP HERE AND PRINT CENTS ONLY
220 $DATA 'FIRST 12 VALUES ARE FIRST YEAR, SECOND 12 ARE SECOND YEAR
230  1249.45,2345.12,1876.34,2001.75,1457.23,1650.00,1763.36,1925.10
240  1368.38,2107.34,1834.24,1765.90
245  1340.95,2508.75,1763.24,1962.10,1561.03,1705.91,1822.54,2018.00
250  1700.19,1569.77,2077.32,1654.92
```

```
MONTH    VALUE    MOVING TOTAL    MOVING AVG.

FIRST YEAR =        21684.72

  1      1249.45     21593.22        1799.43
  2      2345.12     21429.59        1785.80
  3      1876.34     21542.69        1795.22
  4      2001.75     21582.34        1798.53
  5      1457.23     21478.54        1789.88
  6      1650.00     21422.63        1785.22
  7      1763.36     21363.45        1780.29
  8      1925.10     21270.55        1772.55
  9      1368.38     20938.74        1744.89
 10      2107.34     21476.31        1789.69
 11      1834.24     21233.23        1769.44
 12      1765.90     21344.21        1778.68
```

APPENDIX E. SUMMARY

GENERAL ELECTRIC

TIME-SHARING FORTRAN

STATEMENTS

Statements	Examples
Arithmetic	100 A=B=1. : C=D=E=5. : GC=AS/I+2/TLA
	110 F(T+B/3) = FR+(FI=FS*GS/3)*COS(FI)
	120 ML=MAN-"LIC"
Arithmetic Statement Function	100 SINH(X)=.5*(EXP(X)-EXP(-X))
Internal Function	100 RUST(X): ...
	110 KANS(A,B): ...
	120 REAL,JUST( ): ...
	130 INTEGER,FIRST(L): ...
ASSIGN ...TO ...	100 ASSIGN 6 TO J
	110 ASSIGN FORMAL, TO R
	120 ASSIGN A5, TO A6
BACKSPACE	100 BACKSPACE
	110 BACKSPACE 3
	120 BACKSPACE T
CALL	100 CALL FEN(S, A(5), B(L/2))
	110 CALL OUT
	120 CALL B(N)
COMMON	100 COMMON A(12). B,C(3,2), K, J(4,3,2,2)
	110 INTEGER COMMON SUM, LINE, BSL(15)
CONTINUE	100 25 CONTINUE (Not needed because empty statements may be labelled.)
	110 NEXT:CONTINUE
	120 NEXT:
$DATA	100 $DATA
	110 $DATA PAYNOS, CASES
DIMENSION	100 DIMENSION A(5), LOST(45,12)
DO	100 DO 16I = 1,10
	110 DO ALL, L = K,J,2
	120 DO 25, X = 3.5,17., .5
END	100 END (Not needed except before main program.)
END internal	100 END INTERNAL
	110 END RUST
ENDFILE	100 ENDFILE
	110 ENDFILE 5
	120 ENDFILE UNIT
ENTRY	100 ENTRY BACKALWAYS (TO,FROM)
	110 ENTRY AFT1(X)
	120 ENTRY SUMPKB
	130 ENTRY REPEAT(N)
EQUIVALENCE	100 EQUIVALENCE(BEGIN, START, INITIATE), (D(3), B(5), L(2), K(5))
	110 INTEGER EQUIVALENCE (TFG, TFD, TFR)
EXTERNAL	100 EXTERNAL HUNCH, DRAG
$FILE	110 $FILE MP,MCOST,VENDOR,INV1 INV2 INV3 INV4,SUM
FORMAT	100 LINE:FORMAT (I4, 4E12. 5)
	110 77 FORMAT ("NO. OF CASES", I2, 3A3)

STATEMENTS cont'd

Statements	Examples
FUNCTION	100 FUNCTION AFT
	110 INTEGER FUNCTION HUNCH(L,T)
GOTO	100 GOTO 13
	110 GOTO EXTRA
GOTO(...)	100 GOTO(12,LAST,KONLY, 15)AFTER
	110 GOTO (M1,M2,MT3),M
IF(...)	100 IF(A) 25, 26, 27
	110 IF(A*SIN(B)) AGAIN, EXCEPT
	120 IF(J=K/3) 3
	130 IF(IFR-"YES") TRUST,UNT
IF(ENDFILE)	100 IF(ENDFILE 3) 45, EOF
	110 IF(ENDFILE J) MORE
INPUT (terminal)	100 INPUT REPLY, QUAN
	110 INPUT,(COEF(I), I=2, L,2)
INTEGER	100 INTEGER J(3), TRUD(12,2,3)
	110 INTEGER A, S, TLA(16), TLB(16)
$OPT	100 $OPT SS
	110 $OPT SIZE
	120 $OPT REAL
	130 $OPT SUM
	140 $OPT IFF BOTH
PAUSE	100 PAUSE
	110 PAUSE SENSESWITCH3
	120 PAUSE "YES"
PRINT (terminal)	100 PRINT 45,A
	110 PRINT REP, (A(I), I=1,10)
	120 PRINT, A, A*B, A/BCA+3.2
	130 PRINT "MORE OR LESS",↑TABLE,T(I+2)*B
READ (temporary file)	100 READ, A, F, G
	110 READ TITLE
	120 READ 12, F, SYT
READ(...) (permanent file)	100 READ (3,TITLE)
	110 READ (N, 12) VAL, COST, PRICE
	120 READ (UNIT) (T(J), J=1,N),S
REAL	100 REAL A(10), K, NET(6,2,2)
RETURN	100 RETURN
REWIND	100 REWIND
	110 REWIND 2
	120 REWIND KRAK
STOP	100 STOP
	110 STOP "UGH"
	120 STOP V
SUBROUTINE	100 SUBROUTINE KRUG
	110 SUBROUTINE LIMP(V,W,X)
$USE	100 $USE EXCEPN
	110 $USE MATRIX*
WRITE (temporary file)	100 WRITE 16, B, D
	110 WRITE "FIRST TABLE VALUE"
	120 WRITE, (T(K),K=1,25),X,X↑3
WRITE (...) (permanent file)	100 WRITE (3) "MONTHLY SUMMARY"
	110 WRITE (3,12)↑↑ LNO, "REPORT TO DATE",F1,F2,F4,F7

INPUT/OUTPUT

- Uses FORTRAN-IV style I/O statements
- Terminal I/O; temporary and permanent file I/O
- May use unformatted input and standard format output (including list-directed slewing)
- Or may use extensive formatting facilities:

Alphabetic	Octal
Exponential	Power
Fixed-point	Type (substituted type)
General (E or F)	X (blanks)
Hollerith	* (substituted number)
Integer	" (quoted rather than counted Hollerith)

SUBPROGRAMS

• Intrinsic	Accepts all the FORTRAN II closed and open functions with either FORTRAN II or FORTRAN IV names
• External	FUNCTION, SUBROUTINE, main program
• Internal	Arithmetic Statement Function (FORTRAN IV naming rules) Internal Functions (a generalization of ASF)
• Entry	ENTRY statement allows multiple entry to external and internal subprograms and to main program

MODES (INTEGER OR REAL)

• Implied	If first letter of user-supplied name is I, J, K, L, M, or N, named variable, array, or subprogram has implied mode INTEGER. Otherwise it has implied mode REAL.
• Assumed	If an optional single mode $OPT REAL or $OPT INTEGER is given, all user-named items are assumed to have that mode.
• Declared	Items named in INTEGER or REAL statements (or INTEGER or REAL qualified declarations or subprogram definition statements) have declared mode whether or not it agrees with implied or assumed mode.
• Mixed	Integer values used in real calculations are automatically converted to real form; and, real values used in a context where an integer value is required are automatically truncated.

OPERATORS

Operator Symbol	Operation Specified
=	Assignment
+	Addition
-	Subtraction or negation
*	Multiplication
/	Division
** or ↑	Exponentiation

CONSTANTS

Integers within the range: $-524287 \leq \text{integer} \leq 524287$

Reals within the range: $.863616852 \times 10^{-77} \leq |\text{real}| \leq .578960444 \times 10^{77}$

Octal integers within the range: $0 \leq \text{octal integer} \leq 3777777_8$

Quoted characters from one to three characters

NAMES

Composed from letters, digits and $
First character must be a letter
May be from 1-30 characters in length
Blanks are ignored

ARRAYS

May have from 1-15 dimension arrays

For more information about Time-Sharing FORTRAN offered by General Electric's Information Processing Centers refer to TIME-SHARING FORTRAN Reference Manual (IPC-206046).

GENERAL ELECTRIC COMPUTER TIME SHARING SERVICE SYSTEM COMMANDS

COMMANDS	USE
DIRECTIVE	
Control@	To cause the computer to stop whatever it is doing with the program when printing is occurring.
BYE	To disconnect from the System.
GOODBYE	To disconnect from the System.
NEW	To introduce a new program and destroy the working copy of the current program.
OLD	To retrieve from saved store a previously saved file and destroy the working copy of the current program.
Return•	To terminate a program line, cause the System to take action based upon input provided, and act as a normal carriage return.
RUN	To compile and execute a program.
SAVE	To save permanently the working copy of a program.
SCRATCH	To eliminate from the working copy of a program everything but the program name.
STOP	To cause the computer to stop whatever it is doing with the program except when printing is occurring.
UNSAVE	To release and destroy a previously saved program.
User Number	Six characters that identify the user to the System.
EDIT	
Alt Mode• or Escape• or Control with X	To delete an input line as if nothing had been typed.
Arrow (←)	To erase the last character(s) typed. SHIFT key must also be depressed.
EDIT DELETE	To erase portions of a program.
EDIT EXTRACT	To retain portions of a program.
EDIT MERGE	To combine saved files into working store and to resequence line numbers.
EDIT RESEQUENCE	To resequence line numbers in program in working store.
LIST	To list the current working copy of a program.
LIST--X	To list the current working copy of a program beginning at line X (X = 1-5 digits).
RENAME	To change program name but not working copy contents.
INFORMATIVE	
CATALOG	To list a user's catalog of saved programs.
LENGTH	To request the number of characters in working copy of program.
STATUS	To request status of user on system.
TTY	To learn which channel of the DATANET-30* is being used for your connection and to print current user number, problem name, system name, and status.
MODE	
ALGOL	To denote programming language.
BASIC	To denote programming language.
FORTRAN	To denote programming language.
KEY	To reset terminal operation to normal after reading in paper tape.
SYSTEM	To change name of the system under which you are working.
TAPE	To inform the System that paper tape will be read in.

• Special key on Teletype unit
* DATANET is a Reg. Trademark of the General Electric Company

GENERAL ELECTRIC
INFORMATION SERVICE DEPARTMENT
IPC-102046B

INDEX

Computer Centers and offices of the Information Service Department are located in principal cities throughout the United States.

Check your local telephone directory for the address and telephone number of the office nearest you. Or write . . .

General Electric Company
Information Service Department
7735 Old Georgetown Road
Bethesda, Maryland 20014

GENERAL ELECTRIC

INFORMATION SERVICE DEPARTMENT

206046C (10M) 10-68