Time-Sharing Service

Elementary Instruction Guide

Information Systems

Time-Sharing Service

Introduction to Time-Sharing FORTRAN

GENERAL ELECTRIC

INFORMATION SERVICE DEPARTMENT

Time-Sharing Service

INTRODUCTION TO TIME-SHARING FORTRAN

Elementary Instruction Guide

November 1966
Reprinted 6-67, 9-67, & 6-68

GENERAL ELECTRIC
INFORMATION SERVICE DEPARTMENT
227106

PREFACE

This manual provides an introduction to Time-Sharing FORTRAN. Its purpose is to give the user already familiar with Time-Sharing the information he needs to write programs in the FORTRAN language. Once familiar with the information in this manual, he can better understand the more advanced material contained in 206046, Time-Sharing FORTRAN: Reference Manual.

Computer Time-Sharing Service: System Manual (229116) provides information on the use of the Computer Time-Sharing Service of General Electric's Information Service Department. The manual should be used in conjunction with this one.

CONTENTS

APPENDIXES

1. WHAT IS A PROGRAM?

A program is a set of directions, a recipe, that is used to provide an answer to some problem. It usually consists of a set of instructions to be performed or carried out in a certain order. It starts with the given data and parameters as the ingredients, and ends up with a set of answers as the cake. And, as with ordinary cakes, if you make a mistake in your program, you will end up with something else--perhaps hash!

Any program must fulfill two requirements before it can even be carried out. The first is that it must be presented in a language that is understood by the "computer." If the program is a set of instructions for solving linear equations, and the "computer" is a person, the program will be presented in some combination of mathematical notation and English. If the person solving the equations is a Frenchman, the program must be in French. If the "computer" is a high speed digital computer, the program must be presented in a language the computer can understand.

The second requirement for all programs is that they must be completely and precisely stated. This requirement is crucial when dealing with a digital computer, which has no ability to infer what you mean. The computer can act only upon what you actually present to it.

We are, of course, talking about programs that provide numerical answers to numerical problems. To present a program in the English language, while easy on the programmer, poses great difficulties for the computer because English, or any other spoken language, is rich with ambiguities and redundancies. Instead, you present your program in a language that resembles ordinary mathematical notation, which has a simple vocabulary and grammar, and which permits a complete and precise specification of your program. The language that you will use is FORTRAN, which is at the same time precise and easy to understand in its simplest form.

Your first introduction to the FORTRAN language will be through an example. You will study the language in more detail with emphasis on its rules of grammar and on examples that show the application of computing to a wide variety of problems.

This manual is intended only to get you started. The best way to learn how to use computers is to use one to solve your problem. This brings up an important point. A computer, whether a person or a machine, can solve only problems that you know how to solve and can give explicit instructions for each step toward the solution. You need to know the numerical methods as well as a computer language.

A list of references on numerical methods and FORTRAN programming is included in Appendix D.

2. A FORTRAN PRIMER

SAMPLE PROGRAM

The following example is a complete FORTRAN program for solving two simultaneous linear equations in two unknowns with several possible different right hand sides. The equations to be solved are

$$A_1 X_1 + A_2 X_2 = B_1$$

$$A_3 X_1 + A_4 X_2 = B_2$$

Since there are only two equations, we may find the solution by the formulas

$$X_1 = \frac{(B_1 A_4 - B_2 A_2)}{(A_1 A_4 - A_3 A_2)} \qquad X_2 = \frac{(A_1 B_2 - A_3 B_1)}{(A_1 A_4 - A_3 A_2)}$$

Study the example carefully--in most cases the purpose of each line in the program is self-evident.

```
00 30 INPUT, A1, A2, A3, A4, B1, B2
10 D = A1*A4 - A3*A2
20 X1 = (B1*A4 - B2*A2)/D
30 X2 = (A1*B2 - A3*B1)/D
40 PRINT, X1, X2
50 GØTØ 30
60 END
```

We immediately observe several things about the above sample program. First, all lines in the program start with a line number. These serve to identify the lines in the program, each one of which is called a statement. A program is made up of statements, most of which are instructions to be performed by the computer. These line numbers also serve to specify the order in which the statements are to appear in the program, which means that you could type your program in any order.

Before the program is run by the computer, it sorts out and edits the program, putting the statements into the order specified by their line numbers. (This editing process makes the correcting and changing of programs extremely simple, as will be explained in later sections.)

Using FORTRAN Statements

The second observation is that some of the statements start with English words. They are practically self-explanatory. The INPUT, READ, PRINT, GØTØ, and END statements mean just that. The other statements look like algebraic equations. They are. Rather than x for multiplication we use *. The slash (/) is used for division. The minus (-) is used in subtraction.

The third observation is that we use only capital letters.

How the Computer Treats the Statements

Turning now to the individual statements in the program, we observe that the first statement, numbered 00 is an INPUT statement. When the computer encounters an INPUT statement while running your program, it will assign values of input data to the variables whose names are listed after the INPUT statement in the order in which they are listed. Thus when you ask for a run of our sample program you will then have to type numerical values for A1, A2, A3, A4, B1, B2. Note that these numerical values do not appear in the program itself.

The next line, numbered 10, is an algebraic statement. It causes the computer to compute the value of the expression $A_1 A_4 - A_3 A_2$, and to assign this value to the variable D. The expression computed in an algebraic statement can range from the very simple (consisting of only a single variable) to the very complex. The rules for forming these expressions are given in detail in the next section, but for now we point out that:

1. Variable names always start with a capital letter.
2. The symbol * (asterisk) is always used to denote multiplication.
3. Parentheses may be needed to specify the order of the computation.

The lines numbered 20 and 30 complete the computation of the solution, X1 and X2. Notice that the denominator has been previously evaluated as the variable D. Thus it is not necessary to repeat the formula given in line 10. Notice also how parentheses are used to specify that the numerator of the fraction consists of the entire quantity B1*A4 - B2*A2. If the parentheses had been omitted by mistake, the expression computed would have been B1*A4 - B2*A2/D, which is incorrect.

Now that the answers have been computed, they will be printed out for you to see when the computer encounters line 40. Notice in the problem on page 2, that the comma is used to separate the individual items in the list of quantities to be printed out at that time.

Having completed the computation, line 50 tells the computer to execute statement number 30, which is on line 00. The statement number is required whenever the next program step is not on the next line. In this case, we are asking the computer to see if we have more data. If there is none the user will indicate this by typing STØP, followed by a carriage return.

NUMBERS, VARIABLES, AND EQUATIONS

Equations in FORTRAN look much like mathematical equations. They may contain variables, functions, constants, and operations. Variables and constants represent numbers; functions and operations result in numbers when evaluated.

In FORTRAN we must consider two kinds (or modes) of numbers: integers and real numbers. These two modes have different formats when stored in the computer. Consequently there are different ways of representing them in FORTRAN. Integers, or "fixed point" numbers, are written as a string of up to six digits without a decimal point (such as 123456 or -3). The computer understands that the decimal point, if written, would come after the last digit; hence the term "fixed point." In Time-Sharing FORTRAN, integers may take on values between ± 524287, inclusive. Integers are useful for counting, indexing, and simple computations. Real, or "floating point" numbers are written as a string of up to six digits with a decimal point (such as 123.456 or -3.0). The decimal point may come before or after any digit you wish, hence the term "floating point." An alternate way to write real numbers, called the exponential form, adapted from scientific notation, is explained below. This makes it possible to express numbers between $\pm 5.7896 \times 10^{76}$ (57896 followed by 72 zeroes), inclusive.

Numbers may be constants or variables. Constants have fixed values. Variables are any quantities to which we assign names and which we allow to take on new values as the problem proceeds. For example:

```
N = 4
C = -2. + A
```

In these equations N, C, and A are variables. 4 and 2. are constants. Incidentally, 4 is a fixed point constant (no decimal point) but -2. is a floating point constant. Notice that a positive constant need not be preceded by a plus sign but a negative constant must have a minus sign.

Floating point constants may be written in the exponential form. Indeed, they must be written this way if the number has a very small magnitude or a very large magnitude. The exponential form consists of three parts, in order: a real mantissa (decimal point included), the character "E," and an integer exponent (no decimal point) of up to two digits. The mantissa contains up to eight of the most significant (leftmost) digits of the actual number. The "E" simply serves as a marker to separate the mantissa from the exponent. The exponent represents the power of ten by which the mantissa must be multiplied if we want to write out all of the digits and place the decimal point properly. For example:

```
4.732E6 means 4,732,000.
3.52E-4 means .000352
4E-5    means .00004
```

Some possible errors in writing constants in FORTRAN are revealed below:

```
1,896.2 (comma not allowed)
E-3     (no mantissa)
```

Variables may be either fixed point or floating point. In FORTRAN, the names of fixed point variables must start with either I, J, K, L, M, or N. Names of floating point variables start with any other letter. The names of either floating or fixed point variables may have as few as 1 and as many as 30 letters or digits with the first character always a letter. Some fixed point variables might be named NXØ, NEGAT, or LIG2. Some examples of floating point names are B, D1, XAR, AMP1, or X1.

The programmer can assign names to these variables to suit himself as long as he distinguished between floating point and fixed point variables.

Equations are formed in FORTRAN by using these symbols (operators) in conjunction with numbers and variables:

+	means	addition
-	means	subtraction
*	means	multiplication
/	means	division
↑ or **	means	exponentiation (to the power)

Here are some examples of FORTRAN expressions and their mathematical meaning:

FORTRAN EXPRESSION	MEANING
(C-D)/E	$\frac{C-D}{E}$
C-D/E	$C-\frac{D}{E}$
E/(C-D)	$\frac{E}{C-D}$
A/B*D	$\frac{A}{B} \times D$
A/(B*D)	$\frac{A}{B \times D}$
A/B/D	$\frac{A}{B \times D}$
A+C*B**D	$A+C \times B^{D}$
A-(C*B) ↑ D	$A-(C \times B)^{D}$

In the absence of parentheses, operators are applied according to the priority of application each has. The order of priorities is as follows: Exponentiation, multiplication or division, and addition or subtraction. Negation has the same level as addition and subtraction.

A few of the standard functions are available in FORTRAN by name. For example, to compute SIN 3X you would simply write SIN (3*X). These standard functions are all used in this way. Their names are listed in the table below:

Function	Symbolic Name
TRIGONOMETRIC SINE	SIN ( )
TRIGONOMETRIC COSINE	CØS ( )
NATURAL LOGARITHM	LØG ( )
EXPONENTIAL (e**X)	EXP ( )
SQUARE ROOT	SQRT ( )
TRIGONOMETRIC ARCTANGENT	ATAN ( )

These six functions always produce floating-point results. Angles are in radians.

RULES FOR WRITING A FORTRAN PROGRAM

You now have all the information needed to write a simple program similar to the example. Here are a few rules for communicating your program to the computer:

1. Each line must begin with a 1 to 5 digit line number. The first character after the line number must be a blank, unless the entire line is a comment.

2. A line may contain one or more statements. If you enter more than one statement on a line, terminate each statement except the last by a semicolon.

3. A statement may be continued on as many lines as desired. Enter a plus sign (+) as the first nonblank character after the blank following the line number for each continuation line.

4. A statement may or may not be labeled. The label may be a 1 to 5 digit number or a one to 30 character name.

5. Comments may occupy an entire line by entering any nonblank as the first character after the line number. In addition, a comment may be embedded anywhere within a line. Introduce embedded comments with an apostrophe and terminate them with another apostrophe, a semicolon, or the end of the line.

LOOPS AND FLOWCHARTS

One of the most important programming ideas is that of a loop. While we can write useful programs in which each statement is performed only once, such a restriction places a substantial limitation on the power of the computer. Therefore, we prepare programs that have portions which are performed not once but many times, perhaps with slight changes each time. This "looping back" is present in the first program, which can be used to solve not one but many sets of simultaneous linear equations. When a lot of "looping back" is to be done in a program, a flowchart will assist your thinking. Both looping and flowcharting are best shown by an example.

Suppose we want to calculate the steady state AC current in a series RLC circuit for a range of frequency from 10 to 1000 cycles per second, and for a range of resistance from 0 to 50 ohms.

For this program we can use these variables:

A = current (amperes)
C = capacitance
E = volts
F = frequency
H = inductance (henrys)
R = resistance (ohms)
Z = impedance

A flowchart of the calculations might look like this.

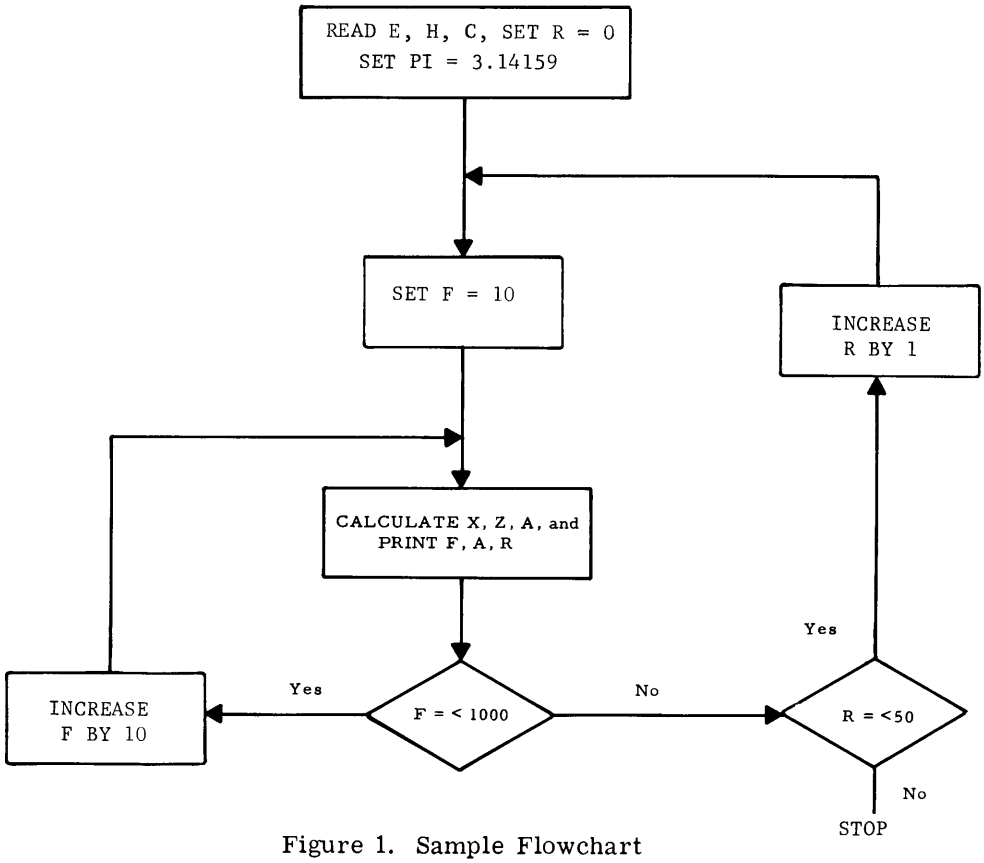

Figure 1. Sample Flowchart

The flowchart shows the looping needed to do this. Notice that it need not be as specific as a program. For instance, one block says calculate X, Z, and A but doesn't give any formulas. It will serve our purpose if we include in the flowchart at least these items:

1. Data that is needed to get started
2. Calculations that can be made with looping
3. Tests that are needed to get in and out of the loops.

Our flowchart shows we must start with values of E, H, C, R, and frequency. Current A is calculated and printed for frequencies up to 1000 in multiples of 10. Then R is increased; F is reset and the process is repeated. This is done until R = 50 which is a signal to stop.

FORTRAN STATEMENTS USED FOR LOOPING

Because loops are so important, and because loops of the type shown in the example arise so often, FORTRAN provides three statements to aid in their programming: the GØ TØ, the IF, and the DØ. Two programs will be shown for our example problem. The first will use the GØ TØ and IF statements; the next will accomplish the same result with fewer program steps by means of the DØ statement.

The first of our two programs looks like this:

```
00 INPUT, E, H, C
10 PI = 0
15 R = 0
20 10 F = 10
30 30 X = 2*PI*H*F - 1/(2*PI*C*F)
40 Z = SQRT(R*R + X*X)
50 A = E/Z
60 PRINT, F, R, A
70 IF(F - 1000) 20, 40, 40
80 20 F = F + 10
90 GØTØ 30
100 40 IF(R - 50) 60, 90, 90
110 60 R = R + 1
120 GØTØ 10
130 90 STØP
140 END
```

You will notice the program follows the flowchart explicitly. The statement on line 00 requires that we input the voltage, the inductance, capacitance, and a starting value of resistance. The next line calculates PI. Then statement 10 establishes the initial frequency F = 10. Lines 30 through 60 are used to calculate and print current. The next statement on line 70 does two things. First it tests whether frequency is less than, equal to, or greater than 1000 cycles per second. Then, depending on the outcome of the test, it branches to a different part of the program, going to statement 20 if the test is negative or statement 40 if the test is zero or plus.

Statement 40 is another IF statement to loop on resistance. Statement 20 increments frequency; 60 increments resistance. They are both followed by GØ TØ statements to complete the loops. Note that the statement numbers referred to in this discussion are not line numbers. A Time-Sharing FORTRAN program <u>can never</u>, in any way, refer to a line number.

Our next program, using the DØ statement, looks like this:

```
00 INPUT, E, H, C
10 PI = 3.14159
20 DØ 20 R = 0, 50
30 DØ 20 F = 10, 1000, 10
40 X = 2*PI*H*F - 1/(2*PI*C*F)
50 Z = SQRT(R*R + X*X)
60 A = E/Z
70 20 PRINT, F, R, A
80 END
```

This program accomplishes the same result with five fewer program steps. The first DØ on line 20 tells the computer to execute all statements down through 20 for all values of the variable R from 0 to 50 in increments of 1. The next DØ of line 30 tells the computer to execute all statements down through 20 for all values of the variable F from 10 through 1000 in increments of 10. The calculations will be done like this: With R = 0 the current will be calculated and printed for all values of frequency. Then R will be changed to 1 and the process will be repeated. This continues until all values of R up through 50 are used.

This use of the DØ statements within a DØ loop is powerful when properly applied. Textbooks on FORTRAN programming detail many additional uses of the DØ loop and identify many common errors that can be made.

ERRORS AND DEBUGGING

It may occasionally happen that the first run of a new problem will be error-free and give the correct answers. But it is much more common that errors will be present and have to be corrected. Errors are of two types: (1) errors of form, or grammatical errors, that prevent even the running of the program; (2) logical errors in the program which cause wrong answers or even no answers to be printed.

Errors of form will cause error messages to be printed out instead of the expected answers. These messages give the nature of the error, and the line number in which the error occurred. Logical errors are often much harder to uncover, particularly when the program appears to give nearly correct answers. In any case, when the incorrect statement or statements are discovered, the correction is made by retyping the incorrect line or lines, by inserting new lines, or by deleting existing lines. These three kinds of corrections are made as follows:

Changing a line	Type it correctly with the same line number.
Inserting a line	Type it with a line number between those of the two existing lines.
Deleting a line	Type the line number only.

To be able to insert a line make sure that the original line numbers are not consecutive numbers. In other words, start out by using line numbers that are multiples of five or ten.

Corrections can be made at any time, either before or after a run. They may even be made in an earlier part of the program while you are typing the later lines. Simply retype the offending line with its original line number, and then continue typing the rest of the program.

The Time-Sharing FORTRAN Reference Manual contains a complete list of the Time-Sharing FORTRAN error messages. The messages listed in Appendix E are those which you will most often encounter.

The Process of Writing and Debugging a Program

The whole process of locating errors or "debugging" a program is illustrated by a case history which starts on the next page. It takes us from the introductory sequence to the final successful printing of the correct answers.

The problem is to locate the maximum point of the sine curve between 0 and 3 by searching along the x-axis. The searching will be done three times, first with a spacing of 0.1, then with spacings of 0.01, and 0.001. In each case the following will be printed a) the location of the maximum, b) the maximum, and c) the spacing. The program as first written down on paper was:

```
00   INPUT, D
10   10 DØ 20 X = 0, 3, D
20   IF SIN(X) M GØTØ 20
30   XO = X
40   M = SIN(XO)
50   20 PRINT XO, M, D
60   END
```

Before typing the program, it was noticed that M is a fixed-point variable. It was decided to substitute P as the program was typed.

Actual Input and Error Detection

```
USER NUMBER--P23456
SYSTEM--FØRTRAN
NEW ØR ØLD--NEW
PRØBLEM NAME--MAXSIN
WAIT.
READY.

00'MAXSIN   A.B.PRØGRAMMER
10  5 INPUT*←, D
20  10 DØ 20 X = 0, 3, D
30  IF (SIN(X) -P 20,20
40  XO - ØØ←←←←←←XO = X
50  P = SIN(X)
60  PRINT, XO, P, D
70  20 CØNTINUE
80  END
RUN
```

1. Notice the use of the backward arrow to correct mistakes as you go along.

```
MAXSIN   11:48

30  IF (SIN(X) -P 20,20
NØT A STMT.----------↑
ERRØRS, NØ EXEC.
TIME AT LINE NØ. 80, 10/6

30  IF (SIN(X) -P) 20,20
RUN
```

2. An inspection of line 30 shows that a parenthesis is missing so we correct the line.

```
MAXSIN   11:50

?0.1

          .10          .0998        .10
          .20          .1987        .10
          .30          .2955        .10
          .40          .3894        .10
STØP

STØP.
READY.
```

3. This time we compiled without error. We type in our first value for D.

4. Only one line of output was planned. We stop the output by typing STØP.

```
60  20 CØNTINUE
70  PRINT, XO, P, D
RUN

MAXSIN   11:52
?0.1
         1.60          .9996        .10
```

5. We reverse the order of lines 60 and 70, placing the PRINT statement after the CØNTINUE statement. (The CØNTINUE statement is often used as the statement labeled with a DØ-end label.)

```
AT LINE NØ. 80: STØP END,

RUN

MAXSIN    11:53

? 0.01
         1.57          1.00          .01


AT LINE NØ. 80: STØP END,

RUN

MAXSIN    11:53

? 0.001
         1.571         1.00          .001
AT LINE NØ. 80: STØP END,
```

6. The output looks good this time but the program was interrupted and we have to type RUN again. The identification line is printed again before each ? for input data.

```
80 GØTØ 5
90 END
```

7. In order to continue with additional values of D, we must return to the INPUT statement. We insert a GØTØ 5 for line 80 and move END to line 90.

```
LIST
MAXSIN    11:54

00'MAXSIN    A.B.PRØGRAMMER
10   5 INPUT, D
20   10 DØ 20 X = 0, 3, D
30   IF(SIN(X) -P) 20,20
40   XO = X
50   P = SIN(X)
60   20 CØNTINUE
70   PRINT, XO, P, D
80   GØTØ 5
90   END

RUN
```

8. To check the program as it now exists, we LIST it.

```
MAXSIN   11:54

? (0.1)                                          <-- 9.
 1.60          .9996          .10
? (0.01)
 1.57          1.00           .01
? (0.001)
 1.571         1.00           .001
```

Following the program listing we type RUN. Fine! At the end of each line we get a new request to insert data. The circled values were all typed by the user.

```
? STØP                                           <-- 10.

STØP.
READY.
```

We type STØP to stop program execution.

```
SAVE                                             <-- 11.
READY.
```

Then we SAVE the program for future use.

```
BYE                                              <-- 12.
*** ØFF AT 11:55
```

We sign off by typing BYE. The time off is printed and we are disconnected from the system.

3. MORE ADVANCED FORTRAN

MORE ABOUT PRINT

In the previous discussion, the method of printing numerical answers has been explained; however, the programmer may also want to print out alphabetic information such as titles, column headings, and messages. With the Computer Time-Sharing Service it is quite simple to do so.

Suppose we want to title our first example problem shown on page 2. It could be done as follows.

```
05  PRINT "SØLUTIØN ØF TWØ EQUATIØNS"
```

Examining this statement, we see that instead of a comma in the PRINT statement we have substituted a message. The message printed will be exactly what the programmer includes between the two quotation marks. This means all the spaces, punctuation, and even misspellings will be printed out when we reach this print statement.

ADVANCED OUTPUT

One of the conveniences of the Computer Time-Sharing Service is that the formatting of results is done automatically for the beginner. The FORTRAN language does, however, permit a greater flexibility for the more advanced programmer who wishes to specify a more elaborate output.

In the use of the PRINT statement, the beginner may not realize that he is specifying a definite format for his answers. The specification is actually within the quotation mark in the statement. Advanced programmers may insert a statement label before the comma and then compose their own individual format.

As an example,

```
120  PRINT 50, A1, G, M
130  50 FØRMAT (E14.6, F6.3, I3)
```

will cause A1 = -22300.5, G = 7.6, and M = 12 to be printed as

```
-0.223005E+05 7.600 12
```

Actually the rules for using these various formats are too involved for this presentation and the reader is referred to the Time-Sharing FORTRAN Reference Manual. However, to familiarize the reader, we give the following table of the various types of format statements available with Time-Sharing FORTRAN.

Type	Data	Representation
I	Integer	Integer
F	Real	Fixed-point decimal
E	Real	Floating-point decimal
Ø	Integer	Octal integer
G	Real	Fixed-point or floating point
H	Alphabetic	Titles, headings
X	Alphabetic	Skips, spaces
A	Alphabetic	Identification, dates

LISTS AND TABLES

In addition to the ordinary variables used by FORTRAN, there are variables that can be used to designate lists or tables. For instance, A(7) would be denote the seventh item in a list called A; B(3,7) denotes the item in the third row and seventh column of the table called B. We commonly write A_7 and $B_{3,7}$ for those same items, and use the term subscripts to denote the numbers that point to the desired items in the list or table. (The reader may recognize that lists and tables are called arrays by mathematicians.)

The name of an array is like any other variable name; it begins with a letter. The subscripts may be any expression, no matter how complicated, as long as they have non-negative integer values. The following are acceptable examples of array items, though not necessarily in the same program:

B(I + K) B(I, K) Q(A(3, 7), B - C)

A Sample Program Using Arrays

The example on the next page shows a simple use of arrays. We might think of this program as one that computes the total sales for each of five salesmen selling three different goods. The P gives the price of the three goods. The table S gives the individual item sales of the five salesmen, where the rows stand for the items and the columns for the salesmen. We assume that the items sell for $1.25, 4.30, and 2.50, respectively, and that salesman 1 sold 40 of item 1, 10 of item 2, and 35 of item 3, and so on.

First Solution Using Arrays

```
SALES    10:52

00  DIMENSIØN P(3),S(3,5)
10  PRINT "ITEM PRICES"
20  INPUT, P
30  PRINT "NØ. ØF ITEMS PER SALESMAN"
40  DØ 20 I = 1,5
50  PRINT "MAN NØ."
60  PRINT, I
70  PRINT "NØ. ØF ITEMS"
80  20 INPUT, S(1,I), S(2,I), S(3,I)
90  PRINT "SALESMAN NØ. SALES"
100 DØ 40 J = 1,5
110 SALES = 0.
120 DØ 30 I = 1,3
130 30 SALES = SALES + P(I) * S(I,J)
140 40 PRINT, J, SALES
150 END


RUN

SALES     10:53


ITEM PRICES
? 1.25,4.30,2.50
NØ. ØF ITEMS PER SALESMAN
MAN NØ.
          1
NØ. ØF ITEMS
? 40,10,35
MAN NØ.
          2
NØ. ØF ITEMS
? 20,16,47
MAN NØ.
          3
NØ. ØF ITEMS
 ? 37,3,29
MAN NØ.
          4
NØ. ØF ITEMS
? 29,21,16
MAN NØ.
          5
NØ. ØF ITEMS
?42,8,33
SALESMAN NØ.  SALES
          1     180.50
          2     211.30
          3     131.30
          4     166.55
          5     169.40
```

By way of explanation, lines 10 through 20 read in the values of the list P. Lines 30 through 80 read in the values of the table S. Lines 90 through 140 compute the total sales for the five salesmen and print each answer as it is computed. The computation for a single salesman takes place in lines 110 through 130. In lines 100 through 140, the letter I stands for the item number, and the letter J stands for the salesman number.

FORTRAN provides that each variable is assigned storage for one number. Thus to use arrays, the programmer must tell the computer to assign more space. To do this, the programmer uses the DIMENSIØN statement. For example,

```
00   DIMENSIØN A(17)
```

indicates to the computer that the subscript of the list A runs from 1 to 17, inclusive; similarly,

```
10   DIMENSIØN B(15,20), S(3)
```

means that the subscripts of B run from 1 through 15 for rows, and 1 through 20 for columns, and that the subscript of the list S runs from 1 through 3. The numbers used to denote the size of an array in a DIMENSIØN statement must be integer numbers.

It should be mentioned that using a DIMENSIØN statement does not require the user to use all of the spaces so allocated.

Another Method for Using Arrays

This same problem might have been solved using a READ statement and a $DATA control line. The Time-Sharing FORTRAN Reference Manual explains the use of control lines. As you begin to write more complex programs you will find these lines useful. The following sample program illustrates the use of READ and the $DATA control line.

Second Solution Using Arrays

```
SALES1     15:14

100 DIMENSIØN P(3),S(3,5)
110 READ,P(1),P(2),P(3)
120 DØ 20 I = 1,5
130 20 READ, S(1,I),S(2,I),S(3,I)
140 PRINT "SALESMAN NØ.  SALES"
150 DØ 40 J = 1,5
160 SALES = 0.
170 DØ 30 I = 1,3
180 30 SALES = SALES + P(I) * S(I,J)
190 40 PRINT,J, SALES
200 END
210 $DATA
220 1.25,4.30,2.50
230 40,10,35
240 20,16,47
250 37,3,29
260 29,21,16
270 42,8,33

RUN


SALES1     15:16


SALESMAN NØ.    SALES
      1          180.50
      2          211.30
      3          131.65
      4          166.55
      5          169.40
```

SUBPROGRAMS

So far we have talked about writing a single FORTRAN program. You can write several programs in such a way that they fit together and form a large program. We call the program which controls the other ones the main program. The main program may reference another program which we call a subprogram, or sometimes a subroutine. Two FORTRAN statements, FUNCTIØN and SUBRØUTINE, define these other programs as subprograms. Although a subprogram is used by the main program, such a set of instructions can be considered as a complete program in itself.

Subprograms save space in the computer because they may be executed many times, under control of the main program but need only be stored once.

FUNCTION and SUBROUTINE Statements

Subprograms must begin with the FUNCTION or SUBROUTINE statement and must end with an END statement. The form of the FUNCTION and SUBROUTINE statements is as follows.

FUNCTIØN FNAME (Arg 1, Arg 2, ..., Arg n)
SUBRØUTINE SNAME (Arg 1, Arg 2, ..., Arg n)

FNAME or SNAME can be any name by which we wish to refer to the subprogram we write. The name must begin with a letter, and otherwise conform to the rules for writing the names of variables. The items enclosed in parentheses are parameters known as formal or "dummy arguments." They represent constants, variables, arrays, and other subprogram names, defined or known in the main program, which will be used in the current subprogram. In other words, the main program will "pass" necessary information to our subprogram by means of the "parameter list" we establish in parentheses in the FUNCTIØN or SUBRØUTINE statement. If our subprogram does not require any information from the main program the parameter list, including parentheses, may be omitted.

After the FUNCTIØN or SUBRØUTINE statement, we write any number of FORTRAN statements which define the procedure we wish the computer to execute. We may use quantities from the main program by writing the appropriate parameter name(s) in our statements. We may "pass back" values to the main program by writing a parameter name to the left of an equality sign. Somewhere in our instructions we must write the one-word statement "RETURN." This tells the computer to stop executing instructions in our subprogram and return to where it was processing in the main program. We may put as many RETURN statements in a subprogram as we need. The last statement in a subprogram must be the one-word statement "END." No more than one END may be specified in a subprogram.

SUBROUTINE Example

Let us now consider an example of a subroutine. Suppose we have a problem in statistics and at several points in the main program we need to know the standard deviations of several sets of numbers. Now we could, if we desired, write the equations in the main program to calculate each standard deviation. We note, however, that the form of all the equations is the same. The only things that can change are the names and sizes of the arrays where our numbers are stored, plus the arithmetic means of those numbers. Hence we would have duplicate coding if we were to calcuate our standard deviations in the main program. On the other hand, by writing a subroutine we need write the equations only once. Then the main program could pass the name, size, and mean of an array to our subroutine; the subroutine would run the calculation and pass back to the main program the desired standard deviation.

We pick a name for our subroutine, say SIG (since statisticians normally use the Greek letter sigma to represent a standard deviation). Next we need names for our parameters, say A for an array name, I for the array size, AMEAN for the arithmetic mean, (MEAN BEGINS WITH "M" and so would be interpreted as an integer, unusable for us), and SD for the standard deviation. Our subroutine follows:

```
100  SUBRØUTINE  SIG(A,I,AMEAN,SD)
110     DIMENSIØN A(200) 'INDICATE A IS AN ARRAY
120     IF(200-I)BAD,5,5 'CHECK ARRAY SIZE
130  5 SUM = 0.
140    DØ 10 J = 1,I 'SET INDEX J TØ STEP THRØUGH A
150 10 SUM = (A(J)-AMEAN)**2+SUM 'TAKE SUM ØF SQUARES ØF DIFFERENCES
160    SD = SQRT(SUM/(I-1)) 'PASS SD TØ MAIN PRØGRAM
170    RETURN 'CØNTINUE PRØCESSING IN MAIN PRØGRAM
180 BAD:PRINT "INVALID ARRAY SIZE."  'TELL USER ABØUT ERRØR
190    STØP  'THIS CØULD BE "RETURN" INSTEAD ØF "STØP"
200    END
```

How to "Call" a Subroutine into the Main Program

Having defined our subroutine, we must now insert statements into the main program to 1) cause our subroutine to be executed, 2) pass needed information to the subroutine and 3) set up storage in the main program to receive information back from the subroutine. All three functions are performed simultaneously by the CALL statement. The form of the CALL statement, strangely enough, corresponds to the form of the SUBRØUTINE statement, i.e.

CALL SNAME (Arg 1, Arg 2, ..., Arg n)

Here SNAME must be the name of a defined subroutine, such as SIG in our hypothetical case. Also the parameter list will contain actual arguments from the main program rather than the dummy arguments specified in the SUBRØUTINE statement. The actual arguments must be listed in the same order and correspond to the same types of items as those in the dummy argument list.

If a dummy argument is an array of real numbers, the actual argument specified in the corresponding CALL statement must also be an array of real numbers. Likewise for integer arrays, real and integer arrays elements, real and integer variables, and constants. (It should be noted that array elements, variables and constants used as actual arguments in a CALL statement may all correspond to variables in a dummy argument list. However, array elements and constants may not themselves be used as dummy arguments in a SUBRØUTINE or FUNCTIØN statement.)

Let us now continue our statistical example by considering various features of a main program which might call our SIG subroutine. Suppose we have four sets of numbers A, B, C and D, consisting of 73, 200, 150, and 100 numbers, respectively. We calculate the arithmetic mean of each set and store them as variables ABAR, BBAR, CBAR, and DBAR (since mathematicians often indicate that a variable is an average by drawing a line over it, like $\bar{x}$). We also want the standard deviations stored in an array, say, called DEVS. Forgetting what other computations might be provided in our main program, the portions necessary to obtain the four standard deviations would look as follows:

```
1000   DIMENSIØN A(73),B(200),C(150),D(100),DEVS(4)
 .          .
 .          .
 .          .
1200   CALL SIG(A,73,ABAR,DEVS(1))
1210   I = 200
1220   CALL SIG(B,I,BBAR,BD)
1230   DEVS(2) = BD; J = 3
1240   CALL SIG(C,150,CBAR,DEVS(J))
1250   Q = DBAR
1260   CALL SIG(D,100,Q,Q)
1270   DEVS(J = J + 1) = Q
 .          .
 .          .
 .          .
```

Note that we have great flexibility in specifying our actual arguments. Referring to our original SUBRØUTINE definition of SIG, notice that the dummy argument A is known to the computer as an array of real numbers because 1) it appears in a dimension statement in the subroutine and 2) its name does not begin with I, J, K, L, M, or N. The corresponding actual arguments A, B, C, and D appearing in the first position of the parameter list are likewise identified in the main program as arrays of real numbers. One of the actual arrays has the same name as the dummy array, namely A. The other three arrays do not. Note also that the dummy array A is dimensioned to have 200 elements, the maximum array size dimensioned in the main program. Look at the second position. The dummy argument is an integer variable, I. It is not an integer array because there is no dimension statement for I in the subroutine. Likewise in the main program an integer I is used once and constants are used the other times. We shall leave a detailed analysis of the other two arguments to the reader.

FUNCTION Example

Next let us recast our SIG subroutine into a function. The main feature of a function is that the function name itself represents a value. In our example to follow we shall let our function name SIG represent the standard deviation. Hence we can omit one parameter. The function follows:

```
100   FUNCTIØN SIG(A,I,AMEAN)
110     DIMENSIØN A(200)
120     IF(200-I)BAD,5,5
130   5 SUM = 0
140     DØ 10 J = 1,I
150  10 SUM = (A(J)-AMEAN)**2+SUM
160     SIG = SQRT(SUM/(I-1)) 'FUNCTIØN NAME GIVEN DEVIATIØN
170     RETURN
180   BAD:PRINT "INVALID ARRAY SIZE."
190     STØP 'THIS SHØULD NØT BE "RETURN",SIG IS UNDEFINED
200     END
```

At line 160 above note our use of the intrinsic function SQRT to take the square root of SUM/(I-1). An intrinsic function is like any other function except that it is already defined in the FORTRAN System. Thus the user does not have to write a function definition subprogram for SQRT like we have just finished for SIG. Further examination of line 160 also gives a clue as to the use of functions in a main program. We do not use a call statement; we merely reference the function name in an equation (with its actual arguments listed inside parentheses following the name).

Recasting our last example of main program features so as to use our new SIG function we have the following:

```
1000   DIMENSIØN A(73),B(200),C(150),D(100),DEVS(4)
 .          .
 .          .
 .          .
1200   DEVS(1) = SIG(A,73,ABAR)
1210   I = 200
1220   BD = SIG(B,I,BBAR)
1230   DEVS(2) = BD; J = 3
1240   DEVS(J) = SIG(C,150,CBAR)
1250   Q = DBAR
1260   Q = SIG(D,100,Q)
1270   DEVS(J = J + 1) = Q
 .          .
 .          .
 .          .
```

Points to Remember When Using Subprograms

Before leaving the subject of functions and subroutines it might be well to mention points that are not immediately obvious about the samples.

1. At line 1250 of both main programs, we assigned an arithmetic mean to the variable Q. Then at line 1260 we called for execution of our subprograms with Q as the actual argument where each subprogram expects an arithmetic mean. So far so good. However, we wrote the statements in such a way that upon completion of our subprograms, the standard deviation would be assigned to Q! Thus upon entering each subprogram, Q represents the arithmetic mean; and when we come back from each subprogram and execute the next statement, Q represents the standard deviation. This unusual usage does not lead to any difficulty because both of our subprograms are written in such a way that all references to the dummy argument for an arithmetic mean are completed before the subprograms pass back any values for the standard deviation.

2. Note that the line numbering scheme to the left of each FORTRAN statement is designed to prevent duplication and/or intermixing of line numbers. This is because the computer, prior to compiling our total program, sorts all of the statements in ascending line number order. If line numbers were written to overlap, we could get subroutine statements mixed in with main program statements and vice versa.

SOME IDEAS FOR MORE ADVANCED PROGRAMMERS

Documenting a Program

An important part of any computer program is the description of what it does, and what data should be supplied. This description is commonly called documentation. One of the ways a computer program can be documented is by supplying remarks along with the program itself. FORTRAN provides for this capability with the comment line. To do this, we follow the line number with a nonblank character. For example,

```
00C  THIS PRØGRAM SØLVES LINEAR EQUATIØNS ØF
05C  THE FØRM
10C  A1*X1+A2*X2 = B1, A3*X1+A4*X2 = B2.
20C  THE DATA MUST FIRST LIST THE FØUR VALUES ØF
30C  A IN ØRDER, THEN THE DESIRED RIGHT HAND SIDES
40C  FØR WHICH SØLUTIØNS ARE NEEDED.
```

might reasonably be added to the original example for solving linear equations. For longer programs, more detailed comment may be needed, especially ones spotted throughout the program to remind you what each of the parts does. The embedded comment, introduced by an apostrophe and terminated by another apostrophe, a semicolon, or the end of the line, will also prove helpful for program documentation.

Each user quickly learns how much documentation he needs to permit him to understand his program, and where to put comments. But it is certain that comments are needed in any saved program. It should be emphasized that these comments have absolutely no effect on the computation.

Round-Off Errors

One of the most difficult problems in computing is that of round-off error. It exerts its influence in subtle ways, and sometimes in ways not so subtle. A full treatment of the effects of round-off error is beyond the scope of this presentation, but one fairly common situation will be discussed.

Most programmers eventually write or encounter a program something like this:

```
00      S = 0
10      X = 0
20   10 S = S + X
30      IF (X-2) 20, 30, 30
40   20 X = X+0.1
50      GØTØ 10
60   30 PRINT, S
70      END
```

for computing the sum of all the non-negative multiples of .1 less than or equal to 2. The correct answer is 21, but invariably the program will produce 23.1 as the answer. What is wrong? The explanation is that the computer works in the binary number system, and cannot express .1 exactly. Just as 1/3 cannot be expressed in terms of a single decimal number, neither can .1 be expressed in terms of a single binary number. It turns out that .1 in the computer is a number very slightly less than .1. Thus, when the loop in the above example has been performed 21 times, the value of X is not 2 exactly, but is very slightly less than 2. The IF statement in line 30 determines that the final value, exactly 2, has not yet been achieved or exceeded, and so calls for one more passage through the loop.

If the programmer had known that the computer treats .1 as a number slightly less, he could have compensated by writing 1.95 in place of 2 in statement 30. The computer performs exactly correct arithmetic for integers. The user may thus count the number of times through the loop with integers. The example may be rewritten with a DO loop as follows:

```
00   S = 0
10   DØ 10 N = 1, 20, 1
20   10 S = S + 0.1*N
30   PRINT, S
40   END
```

Methods for Checking Programs

One of the most exasperating problems confronting programmers is that of a fairly long and complex program that looks as if it should work, but simply refuses to do so. (Presumably, all errors of form have been detected and removed.) The locating and removing of logical errors is called debugging, and the methods to be used depend on the nature of the program and also on the programmer himself.

The first thing to do with an apparently incorrect program is to check very carefully the method used. Calculate the problem by alternate means and compare the results with the output program. If that doesn't uncover the bug, then examine very carefully your programming to see if you have mixed up any of the variables. It is often difficult to spot such errors because one tends to see in a program what he expects to see rather than what is there.

Another method that is extremely useful in providing clues as to the nature and location of the bug or bugs is tracing. In FORTRAN this tracing may be accomplished by inserting superfluous PRINT, statements at various places in your program to print the values of some of the intermediate quantities. When the program is RUN, the values of these intermediate quantities often suggest the exact nature of the error. When the program has been debugged and is working properly, these statements are removed.

APPENDIX A. GENERAL INFORMATION

1. The name of a floating point variable must not begin with I, J, K, L, M, N, or a number. The name must have no more than 30 characters.

2. A variable name must not be the same as any of the listed intrinsic names.

3. A symbol for a variable must appear in an input list or be defined by an arithmetic statement before the first statement in which it is used.

4. Use a nonblank character as the next character after the line number for comments to be printed with the source program listing only. Or use embedded comments, introduced by an apostrophe and terminated by another apostrophe, a semicolon, or the end of the line.

5. If a statement is too long to fit on a single line it can be continued over as many additional lines as necessary. Continuation lines must have a plus (+) sign as the first nonblank character after the blank following the line number.

6. Arithmetic operations are performed in the following order: exponentiation; multiplication and division; addition, subtraction and negation. Operations are performed from left to right.

7. All floating point data numbers must contain a decimal point. They must lie in the range $\pm 5.7896 \times 10^{76}$ (57896 followed by 72 zeroes), inclusive.

APPENDIX B. SUMMARY OF STATEMENTS

This appendix summarizes the Time-Sharing FORTRAN statements and arithmetic operators.

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 110 77 FORMAT ("NO. OF CASES", I2, 3A3)
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

STATEMENTS cont'd

Statements	Examples
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) EOF, 45 110 IF(ENDFILE J) ENDF
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 $ 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

OPERATORS

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

APPENDIX C. TABLE OF FUNCTIONS

The following functions are available in Time-Sharing FORTRAN. (The terminating F is to provide compatibility with FORTRAN II. The functions performed are identical.)

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	
LABS	1 Integer	Integer	
ATAN	1 Real	Real	Principal angle in radians whose tangent is argument
ATANF	1 Real	Real	
CØS	1 Real	Real	Cosine of angle in radians
CØSF	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
FIXF	1 Real	Integer	
IFIX	1 Real	Integer	
XFIXF	1 Real	Integer	
FLØAT	1 Integer	Real	Given integer converted to a real
FLØATF	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	
LØG	1 Real	Real	Natural Logarithm of argument
LØGF	1 Real	Real	
ALØG	1 Real	Real	
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	

Name	No. of Arguments and Assumed Mode	Result Mode	Definition
AMIN1	≥2 Real	Real	Minimum of arguments
MIN1F	≥2 Real	Real	
MIN1	≥2 Real	Integer	
XMIN1F	≥2 Real	Integer	
AMINO	≥2 Integer	Real	
MINOF	≥2 Integer	Real	
MINO	≥2 Integer	Integer	
XMINOF	≥2 Integer	Integer	
AMØD	2 Real	Real	Remainder on dividing argument 1 by argument 2
MØDF	2 Real	Real	
MØD	2 Integer	Integer	
XMØDF	2 Integer	Integer	
RND	1 Real	Real	1. If arg = 0, provides next in sequence of pseudo-random numbers uniformly distributed, 0 < n ≤ 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. REFERENCES

J. B. Scarborough - Numerical Mathematical Analysis - John Hopkins, Press, 1950.

Cecil Hastings, Jr. - Approximations for Digital Computers, Princeton University Press, 1955 (Chebyshev Polynominals).

Grabbe Ramo Wooldridge - Handbook of Automation, Computation and Control, Wiley, 1958.
Vol. 1 - Section A - General Mathematics
Vol. 1 - Section B - Numerical Analysis
Vol. 1 - Section C - Operations Research
Vol. 2 - Computers and Data Processing

W. E. Milne - Numerical Solution of Differential Equations, Wiley, 1953.

M. G. Salvadori - Numerical Methods in Engineering, Prentice-Hall, 1952.

D. R. Hartree - Numerical Analysis, Oxford Clarendon Press, 1952.

Hildebrand, Forsythe & Wasow - Finite Difference Methods for Partial Differential Equations, Wiley, 1960.

McCracken & Dorn - Numerical Methods and FORTRAN Programming, Wiley.

McCormick and Salvadori - Numerical Methods in FORTRAN, Prentice-Hall.

Time-Sharing FORTRAN Reference Manual (206046), General Electric, 1966.

Time-Sharing System Manual (229116), General Electric, 1966.

APPENDIX E. ERROR MESSAGES

The following error messages are those which most frequently occur during execution of a FORTRAN program.

Execution is aborted after the following messages:

Message	Meaning
BEYØND	Attempt made to store outside storage space. Line number is of statement in which storage is attempted.
NØTREAL	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.
SUBSCRIPT	Subscript error. Value of subscript smaller than one or exceeds size specified for dimension. Line is of statement in which the subscript is given.

Execution is continued after the following messages:

DIVBYZERØ	Real division by zero. The largest possible positive number is provided. Line number is of statement in which division occurs.
EXP	Raising of exponent 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.

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 function requires an integer argument Subscript
LØG	Logarithm requested for zero or negative value. Zero or logarithm of positive number of same magnitude, respectively, is provided. Line number is of statement in which division occurs.
MØD	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.
ØVERFLØW	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.
R ↑ R	Raising of a real zero to a zero or negative integer zero is provided.
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.
SIN/CØS	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.

Message	Meaning
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.
UNDERFLØW	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 after the following messages:

STØPEND	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).
STØP 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 is suspended after the following message:

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 carriage 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.

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

227106 (5M) 6-68