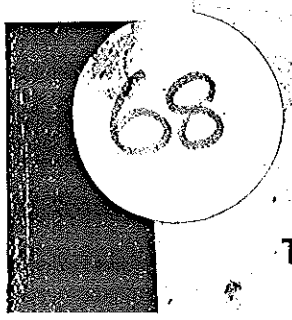
MATSCIENCE REPORT No. 68

LECTURES ON

ELEMENTS OF FORTRAN PROGRAMMING

By
K. SRINIVASA RAO

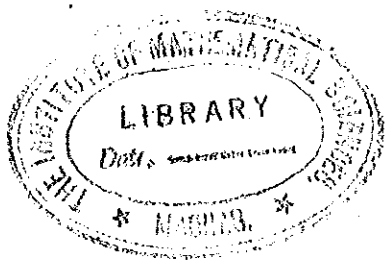


THE INSTITUTE OF MATHEMATICAL SCIENCES, MADRAS-20 (INDIA)

MATSCIENCE REPORT 68

THE INSTITUTE OF MATHEMATICAL SCIENCES
MADRAS-20 (INDIA)

LECTURES ON
ELEMENTS OF FORTRAN PROGRAMMING



By
K. SRINIVASA RAO+

+Senior Research Fellow, MATSCIENCE, Madras-20, India.

PREFACE

This report is based on a series of lectures delivered at Matscience during April-May 1969 to a group of students who were keen to learn the elements of FORTRAN programming. These notes describe in detail the basic FORTRAN language (viz.FORTRAN II) which is acceptable not only to the IBM1620 and IBM1130 computers now in Madras but also to CDC3600 in Bombay. The lecturer wishes to express his deep sense of gratitude and thankfulness to Professor Alladi Ramakrishnan for encoumagement and to Miss P.K.Geetha for her help in the preparation of this report.

K.S.R.

CONTENTS

Elements of
FORTRAN PROGRAMMING

Introduction:-

A PROGRAM is an ordered sequence of instructions acceptable to a computer. Programming is the process of writing up an ordered sequence of instructions in a language which a computer will recognize and accept. Of the many programming languages which have been developed, some of the well-known languages are: FORTRAN, ALGOL, COBOL, CPL and PL/I. Each language has many versions - for e.g. FORTRAN exists in Marks I to IV and in varieties such as Hartran (in use at Harwell, U.S.A.) and as Madtran (in use at M.I.T., U.S.A.); ALGOL has variations such as Smalgol (for "Small Algol") and Balgol (for "Burroughs Algol"). Here,

FORTRAN stands for FORmula TRANslation,

ALGOL stands for ALGOrithmic Language,

COBOL stands for COmmon Business Oriented Language,

CPL stands for Combined Programming Language, and

PL/I stands for Programming Language No.1.

Currently, the most widely used form of Fortran is FORTRAN II developed in 1962 by Rabinowitz for IBM machines. A great step forward in Programming languages was made when ALGOL 60 was introduced in 1960 by Naur. A number of features of ALGOL are included in FORTRAN IV. While FORTRAN and ALGOL are the most widely accepted lahguages which use mathematical notation to

express scientific and engineering problems, COBOL is probably the most widely accepted nonmathematical (business) language. Two of the more recent languages which are essentially improvements over Fortran and Algol are CPL and PL/I.

All statements that specify the problem-solving procedure constitute the Source Program, and it is written in a source language (like Fortran) which is akin to our written language. When the Source Program is punched onto cards and fed into the Computer, the Computer with the help of a "Compiler" or Translator analyzes each source statement and converts it into a machine language instruction. The computer thus translates the Source Program into an object Program which is in the machine language. The machine-oriented language has little in common with the procedure-oriented source language. This translation is absolutely necessary since each computer accepts only one particular language for computation, viz. its own machine language. It is thus the Object Program which is executed by the Computer to obtain results. The compiler, which is also called a processor or translator, is itself a large program of computer instructions and it is usually supplied by the Computer manufacturer. Thus, it is interesting, to note that Computers are used for dual purposes of 'translation' and computation.

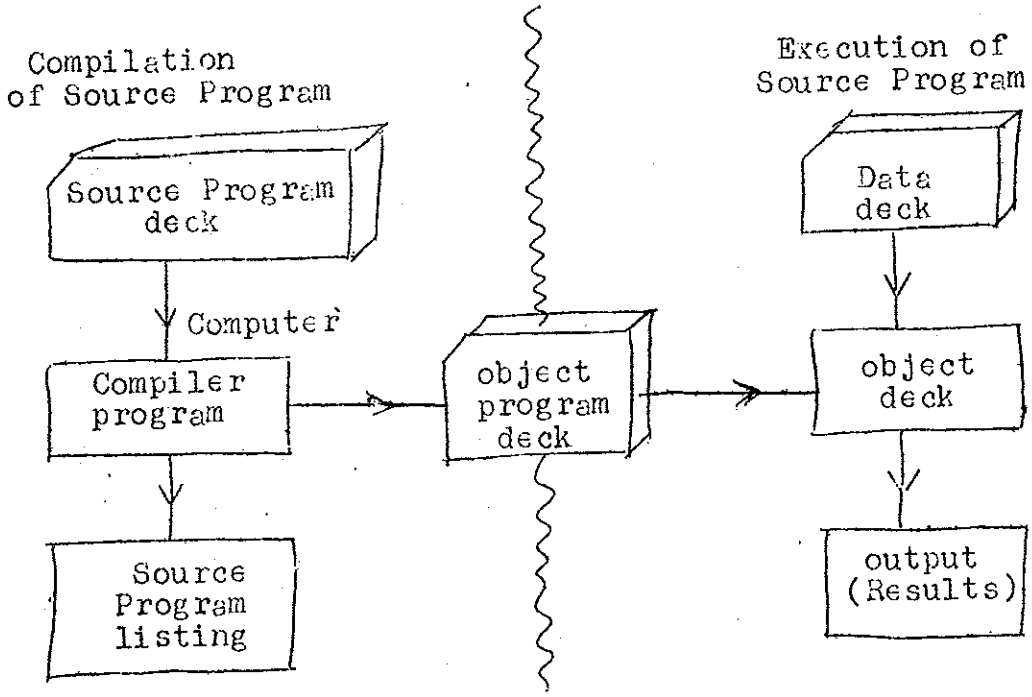
A program written in FORTRAN for IBM 1620, after minor modifications, can also be executed on other computers such as IBM 1130, IBM 7090, IBM 1401, and many others. The writer of a Computer program must be thoroughly familiar with the basic

language units and with the grammar and punctuation of the programming language acceptable by the Computer. A student who has mastered the dialect of basic Fortran programming for IBM 1620 will have no difficulties in adapting himself to other dialects appropriate for other machines. Therefore, in these lectures I will describe in detail the basic Fortran (FORTRAN II) language which, incidentally, is acceptable not only to the IBM 1620 and IBM 1130 Computers in Madras but also to the CDC 3600 Computer in Bombay.

Before proceeding with basic Fortran elements, one should be aware of how a Computer works. The Computer uses only the four basic arithmetic operations of addition, subtraction, multiplication and division for solving all problems. So, after choosing the problem and choosing a general approach to solve it, the programmer must use his knowledge of numerical analysis to express trigonometric functions, differential equations, integrals, square roots, logarithms, etc., in terms of the basic arithmetic operations. The numerical procedure must now be stated, using a programming language, as a precisely defined set of Computer operations. This programming is done in two parts. In the first part the sequence of operations is written in graphical form as a flow chart. This procedure is then elaborated into a sequence of statements in a programming language, which is the Source Program.

The conversion of the source program into the object program by the computer is called the process of Compilation.

After compilation, the computer starts executing the instructions contained in the program. A schematic representation of the complete process of compiling and executing a Fortran program is shown below:

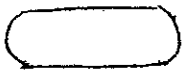


Though the computer is only a calculating machine having no capacity to think, the great power of a digital computer is derived from its high speed - a single arithmetic statement takes only a micro-second for execution.

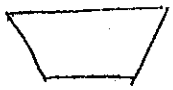
FLOW CHARTING:

Flow charting is an important tool of programming. The flow chart or block diagram allows the programmer to plan the sequence of operations in a program before writing it in a programming language. A flow chart is made up of a set of boxes

of different shapes along with connecting lines and arrows, which shows the "flow of control" between the various operations. Thus, a flow chart provides visual assistance for programming. The flow chart contains the following symbols:

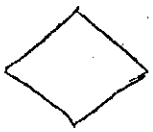


- An oval indicates the beginning/end point of a source program.



- A trapezoid indicates an input/output operation.

- A rectangle indicates any processing operation except a decision.



- A diamond indicates a decision. The lines leaving the box are labelled with the decision results that cause each path to be followed.

- A small circle indicates a connection between two points in a flow chart, where a connecting line would be clumsy. This is called a remote connection box and it contains a Greek letter to indicate the first ending and the second beginning of the flow chart.

- Arrows indicate the direction of flow through the flow chart; every line connecting two boxes should have an arrow on it.

It should be mentioned at the outset that there is no such thing as <u>THE</u> flow chart for a given computational problem.

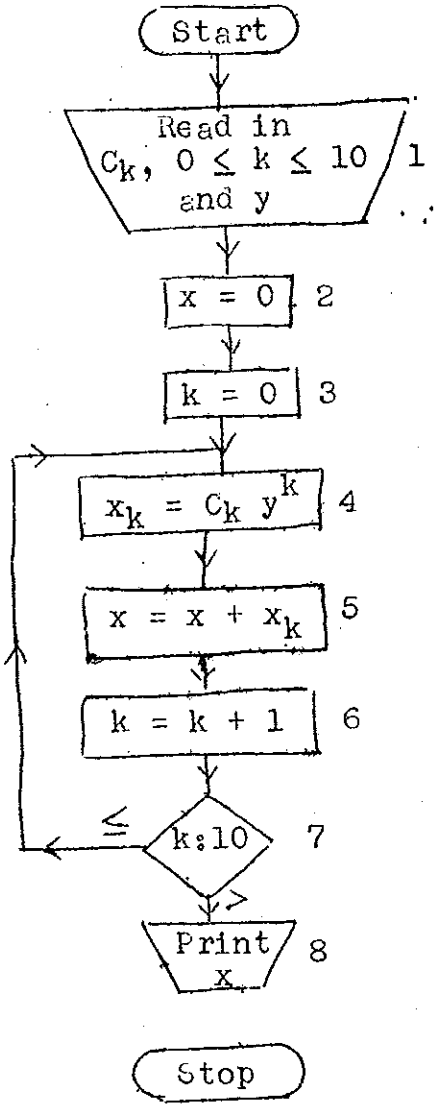
Example 1: To illustrate flow charting we will now consider a simple example to evaluate the polynomial:

$$x = \sum_{k=0}^{10} C_k y^k$$

where C_k $(0 \leq k \leq 10)$ are known constants. The problem, to be solved by the computer, involves the following step by step procedure:

1. Start reading in all known values of C_k and y.
2. Set x = 0, i.e. store zero in the location for x.
3. Set k = 0, i.e. store zero in the location for k.
4. Compute $x_k = C_k y^k$. Note that x_k has its own location.
5. Add x_k to x and store the sum in the location for x. This automatically erases the value of x.
6. Increase k by 1.
7. Check whether the present value of k is less than or equal to 10. If $k \leq 10$, go to step 4; if $k > 10$, the required answer for x has been obtained.
8. Print out the answer and stop the calculation.

The first statement is an input statement, while the second and third are called "Initialization" statements. The fourth, fifth and sixth statements are arithmetic operations while the seventh is an arithmetic decision making statement and the eighth is an output statement. Having thus categorized all the statements involved in the computation we are now in a position to draw the following flow chart:



Note that in step 5, $x = x + x_k$ <u>does not mean</u> $x_k = 0$, as it would in common arithmetic language. It actually means that the initial value of x is increased by the amount x_k and the sum $x + x_k$ is now placed in the location for x. Thus, in FORTRAN, the equality sign is interpreted as "replaced by" and not as "equal to". Similarly, statement 6, k = k+1, means that the value of k is replaced by k+1. After executing statements 4,5,6

and 7, the value of k is checked and if it is found to be less than 10, then the sequence 4,5,6,7 is again repeated. Since the index k takes the eleven values 0,1,2,...,10; the sequence of steps 4,5,6,7; 4,5,6,7;... is repeated eleven times. In other words the sequence of statements 4,5,6,7 are in a LOOP, which is executed eleven times.

Notice that step 7 is important in that advantage has been taken of the Computer's decision making ability by comparing one quantity with another. Based on this comparison, there are two possible paths of subsequent operations, and the two arrows coming out of the diamond shaped box serve to indicate these two possibilities.

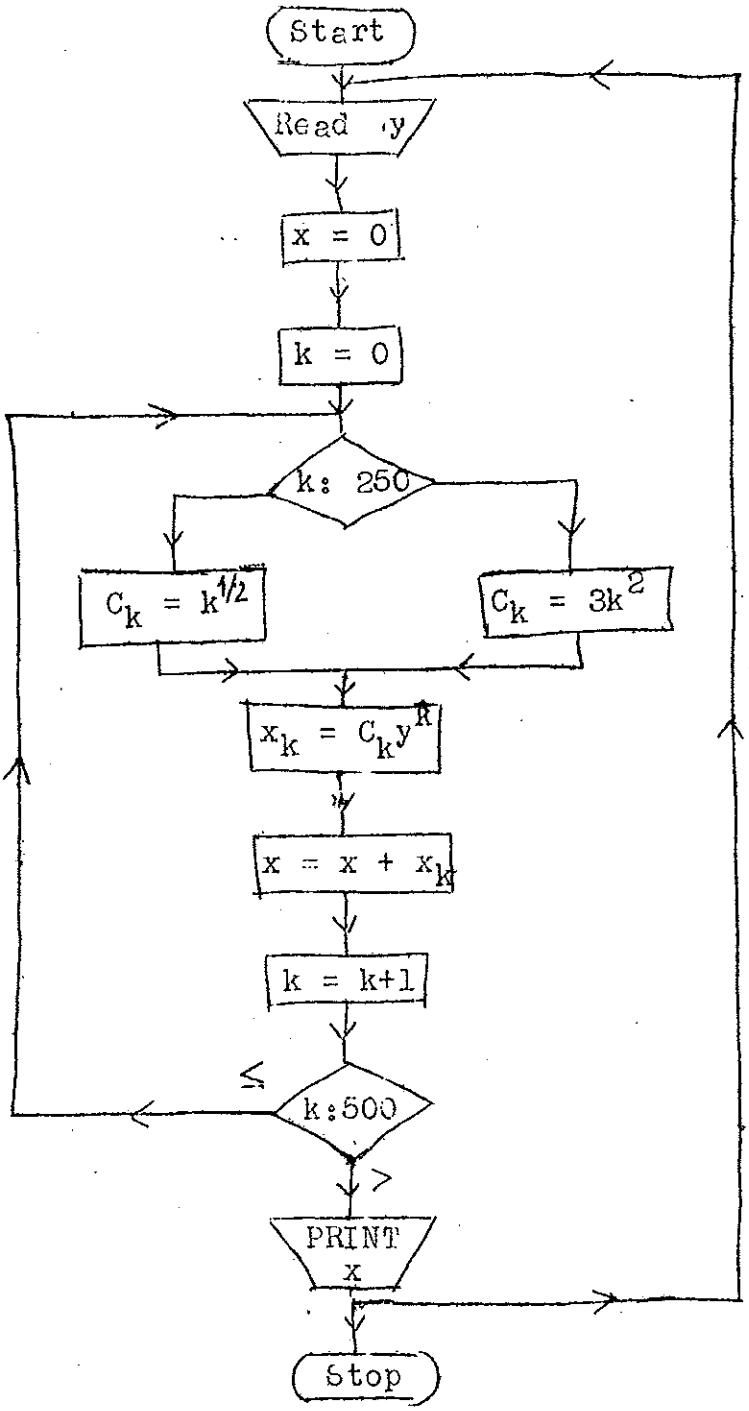
Example 2: Let us now consider the case:

$$x = \sum_{k=0}^{500} C_k y^k$$

where $C_k = k^{1/2}$ for $0 \leq k \leq 250$

and $C_k = 3k^2$ for $251 \leq k \leq 500$

The C_k's can be generated in the computer since their general form is known. This process of generation is faster than reading in numbers. And suppose we wish to evaluate the above polynomial for n different values of y. Then, the modified flow chart will be as shown below:



In the flow chart we indicate our intentions of evaluating the polynomial:x for various values of y, by instructing the computer

to go to Read another value of y after printing the result for the first read value of y. This process can be repeated n times. The flow chart illustrates the existence of a Loop within a loop. An extension to a system of many loops is thus possible.

Exercise.1 Draw a flow chart to compute

$$C = \sum_{k=1}^{62} A_k B_k$$

given that $A_k = k \text{ Sin } k, \quad k \leq 30,$

$= k^2 \text{ Sin } k, \quad k > 30,$

and $B_k = e^k.$

ELEMENTS OF FORTRAN STATEMENTS:

The elements of Fortran Statements are:

(i) Constants,

(ii) Variables,

(iii) Operation symbols, and

(iv) Expressions.

(i) Constants: Constants appear either in Fortran statements in the Source program or as input data. They may be written in one of two forms:

Fixed-point constants or integers; and

Floating-point constants.

A fixed-point constant never has a decimal point associated with it, while a floating-point constant always has a decimal point associated with it. If the constant is to be used in floating-point calculations, it should be written in the floating-point form; if it is to be involved in fixed-point calculations only, it should be written in the fixed-point or integer form. If a constant is to be used in both fixed-point and floating-point calculations, then it must appear in both forms.

Fixed-point constants may be positive or negative. If positive, the sign is optional, but the sign must precede the constant if it is negative.

Examples: 374; 9; +6; 06; -1792; 0; 25; etc.

Floating-point constants may be written with or without exponent. The preceding plus sign is optional, while the preceding minus sign is essential. The decimal point must be written.

Examples of floating-point constants without exponent are:

3.14159265 ; +3600.0 ; -82.5 ; .007 ; 6. ; -0.032; etc.

In the case of floating-point constants with exponent, the mantiss is followed by the letter E which is followed by a sign and a one or two digit integer. The exponent is a fixed-point constant that signifies the power of 10 by which the mantissa is to be multiplied. The positive sign of the exponent is optional, while the negative sign is essential.

Examples of floating-point constants with exponents are:

1.71E03 (1.71×10^3) ; -16.0E + 06 (-16.0×10^6);
+397.02E2 (397.02×10^2) ; 2.998E10 (2.998×10^{10});
-6.0E-03 (-6.0×10^{-3}); -132.6E+12 (-132.6×10^{12}); etc.,

Note that the exponent is restricted to the range -49 to +50.

(ii) Variables: Variables, like constants, may be fixed-point or floating-point variables, depending on whether they are being used to represent integral values or decimal values. Names are assigned by the programmer for variables with the help of the following characters:

Alphabets - A to Z (Capital letters only)

Numbers - 0 to 9

To distinguish between zero and 0, it is customary to write the letter 0 as Ø. The following are the rules to be followed by a programmer while giving names to variables:

(a) The symbolic name must be from one to five alphabetic or numeric characters in length.

(b) Special characters Viz. + - . , / $ ' = ( ) ; should not be used as part of a variable name.

(c) The first character of the variable name must be alphameric.

(d) The first character of a Fixed-point variable must be one of the following: I,J,K,L,M or N.

(e) The first character of a Floating-point variable must be alphameric and other than I,J,K,L,M or N.

Examples

Fixed-point variables: I,J123,MASS,LUCK,KING,K1N2,etc.

Floating point variables: ALPS,ASS,GAS,F$\phi\phi$L,DELTA,H12,etc.

Variable names not allowed: XA/B, 2L1,A+3,I-K,etc..

At present we are dealing with only single variable names, later we will introduce subscripted variables.

It is advisable to choose names with a high mnemonic content for variables. For example, if one wishes to compute electric current from Ohm's law: I = E/R, the programmer might choose to write the Fortran statement as CURR = VϕLT/ϕHM, if the variables I,E and R take floating point values or as ICURR = IVϕLT/IϕHM if the variables take fixed-point or integer values.

(iii) Operation Symbols:

There are only five basic operations associated with the Fortran language. Each operation is represented by a specific symbol as follows:

Operation	Symbol	Example
Addition	+	A + B
Subtraction	-	C - D
Multiplication	*	A * B
Division	/	C / D
Exponentiation	* *	A**B(A^B)

Besides these the "equals" symbol is also available. As mentioned earlier it actually means, in Fortran language, "repla by" and not "equal to". We will see later how this symbol is us only while writing a Fortran statement.

(iv) Expressions:

An expression in Fortran is a sequence of one or more constants and/or variables joined by any one of the operation symbols (except the equals symbol) to indicate a quantity or series of calculations to be performed.

Examples of simple Fortran expressions are:

A+B-C ; A*B*C ; HE/JACK ; etc.

Rules for forming expressions:

(a) In an expression, no two operating symbols should appear side by side. For example, the algebraic expression A(-B) should be written in Fortran as A*(-B), where use has been made of parenthesis to separate the two operation symbols and - .

(b) Parentheses are used in expressions to indicate groupings just as in ordinary mathematical notation. For example, the expression A**(B+C) would mean that the addition is performed before A is raised to the power (B+C); while A** B+C would mean $A^{B}+C$. The most common programming error involving the omission of parenthesis is to write A/B*C for A/(B*C). The golden rule to follow is: WHEN IN DOUBT PARENTHESIZE.

(c) In the absence of parenthesis the hierarchy of operations is as follows:

Rank	Operation	Symbol
First	Exponentiation	* *
Second	Multiplication, Division	* , /
Third	Addition, Subtraction	+, -

Thus, the Fortran expression A+B * C* *D would be interpreted to mean A + (B x C^{D}) while A+(B*C)* *D would be interpreted

to mean $A+(B \times C)^D$.

(d) When expressions contain a string of operation symbols of equal rank, the order of operations is taken from left to right. Thus, A*B*C/D*E/F means ((((A*B)*C) / D)*E) / F).

Note: Left-to-right rule does not apply to fixed-point multiplication and division. For example, the parenthesis in (L*M)/N should not be omitted.

(e) Any expression may be raised to a power that is a positive or negative integer quantity, but only a real expression can be raised to a real power. An exponent may itself be an expression of floating-point or integer type. Thus X**(I+2), A**(B+2.0) are acceptable.

In ordinary mathematical notation the expression A^{B^C} is meaningful. However, the corresponding Fortran expression A**B**C is not allowed in the Fortran language. It should be written as A**(B**C) if A^{B^C} is meant or as (A**B)**C if $(A^B)^C$ is meant. [For example note that $(2^3)^2 = 8^2 = 64$ while $2^{(3^2)} = 2^9 = 512$.]

Exceptions: (e.1) A floating point variable may have a fixed point exponent. Example: A**I, B**2, etc.

(e.2) An integer cannot be raised to a floating-point variable power, because, in general, the result would have a fractional part and hence cannot be expressed in an integer form. Thus I**A is not permissible.

(f) Fixed and floating-point variables and/or constants must not be mixed in the same expression. In other words, all the variables and the constants in the same expression should be in the same mode. A few examples of incorrect and correct Fortran expressions are given below:

Incorrect	Correct
A + 2*C	A + 2.0*C
I*J + 2.0	I*J + 2
A + B - J	A + B - AJ

(g) Unlike the ordinary rule of mathematical notation, parenthesis in an expression does not imply multiplication. Thus the expression (A+B)(C+D) is incorrect and it should be written as (A+B)*(C+D).

(h) When an integer is divided by another integer; the result is not usually an integer. In Fortran, integer division is arranged to truncate a quotient having a fractional part to the next smaller integer, which means simply to ignore any fractional part. Thus, the result of integer division (7/2) is 3 and not 3.5. If the floating-point result 3.5 is desired, then the numerator and denominator must be written as floating-point constants (7.0/2.0).

Occasionally, these peculiarities of fixed-point arithmetic can be used to advantage. For instance, the expression N-(N/2)*2 has the value 0 or 1 according to whether N is even or odd. This fact can be exploited for determining the sign of the factor $(-1)^N$. (see page 75).

Exercise 2: Write valid Fortran expressions for: XY^2, B^{K+2}, X^{A+B}, $a + \frac{b}{c + \frac{e}{f}}$, pq/rs, $\frac{A}{B} + N$, $AX^2 + BX + C$, A^{X+2} and $(b^2 - 4ac)^{1/2}$.

Before dealing with the various types of allowed Fortran Statements, let us consider how a Fortran source program is to be punched onto cards using punching machines.

PROGRAM PUNCHING.

As already stated, a Fortran source program is an ordered sequence of statements and the program is normally given to the computer as a deck of punched cards, each Fortran statement being punched on one (or, if necessary, more) cards. Normally, each punched card corresponds to one Fortran statement. A card has 12 rows and 80 columns. The rows are numbered from top to bottom as (12,11)0,1,2,...,9 while the number of columns are numbered from left to right as 1,2,....,80. The two punching positions in a column are divided into two areas, numeric and zone. The first nine punching positions from the bottom are numeric punching positions and have an assigned value of 9.8,7,6,5,4,3,2 and 1, respectively. The remaining three positions 0,11 and 12 are the zone positions. The zero position is considered to be both a numeric and a zone position.

The numeric characters 0 through 9 are represented by a single punch in a vertical column. For example, 0 is represented by a single punch in the zero zone position of the column. The numeric 5 is represented by a single punch at the 5 position of the column.

The alphameric characters, A to Z (Capital letters only), are represented by two punches in a single vertical column, one zone punch and one numeric punch. The alphameric characters A to I use the 12 zone punch and a numeric punch 1 to 9. In this way, the letter A is punched as 12-1, B as 12-2,... I as 12-9. The alphameric characters J to R use a 11 zone punch and a numeric punch 1 to 9, respectively; i.e. J is punched as 11-1, K as 11-2,..., R as 11-9. The alphameric characters S to Z use the 0 Zone punch and numeric characters 2 to 9, respectively; i.e. S is punched as 0-2, T as 0-3,..., Z as 0-9. Special characters, viz. + - , ( ) * $. = ' ≠ : are represented by one, two or three punches in a single column of the card and consist of punch configurations not used for numeric or alphameric characters.

The standard arrangement of the information on a Fortran statement card is as follows:

Columns 1 to 5: These columns are used to assign statement numbers to the Fortran statements. A statement number is a fixed point constant. Actually each statement in the source program can be assigned a statement number, but it is advisable to number only those Fortran statements that will be referred to in the main program. Unnecessary, unreferred statement numbers waste core storage and delay the compilation process. A statement number can run from 1 or 00001 to 99999 and it can be written anywhere in the field of the first five columns of the card. This only means that blanks will be ignored. Statement numbers need not be assigned sequentially since they serve only as identifying

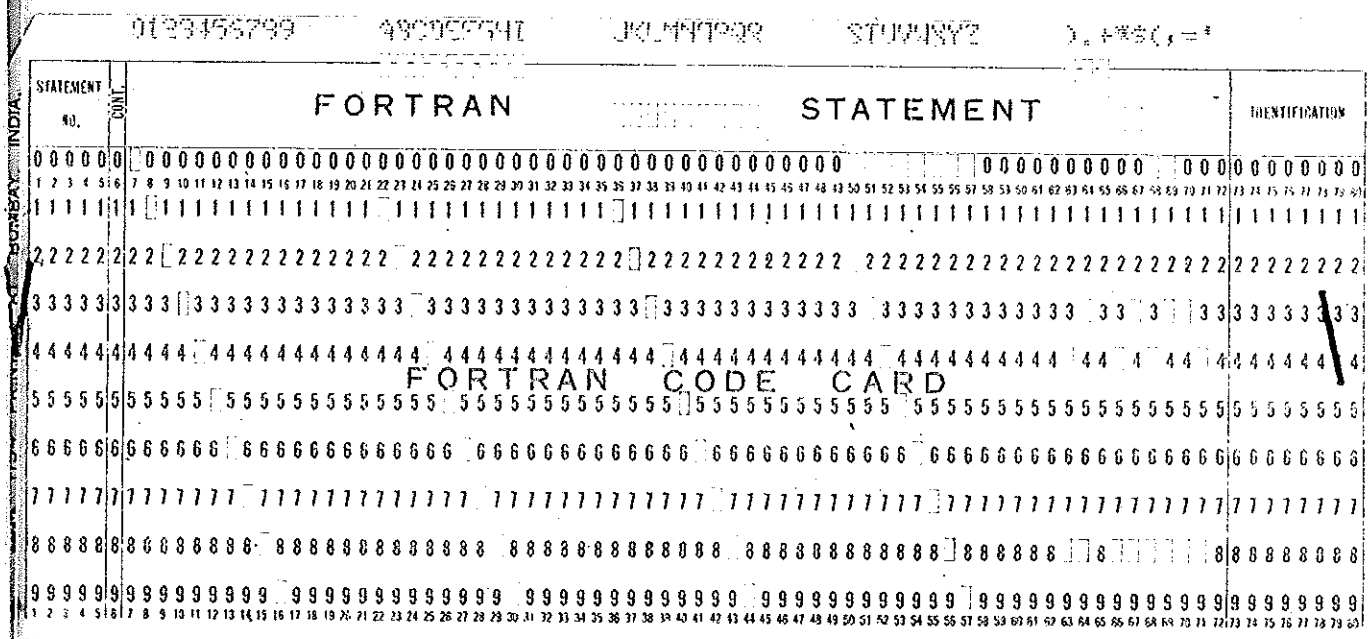
labels for the statements. Further, no two statements should be assigned the same statement number.

Column 6: This is called the continuation column. If a statement cannot be punched on a single card, it can be continued on to the next card by punching any character other than zero in the sixth column. A maximum of nine continuation cards are allowed for a statement. It is a good practice to punch 1,2,..., in column 6 of the first, second,..., continuation card. Note that a statement number should not appear on any of the continuation cards, even if the initial statement card has a statement number. It is not advisable to use too many continuation cards, unless they are necessary. If a statement can be punched entirely on one card, then column 6 should be left blank.

Columns 7 to 72: These columns are available for the Fortran statement proper. Blanks, if any, are ignored. The statement need not necessarily be started in column 7 itself. Blanks may be used to improve the readability of the statement on the card.

Columns 73 to 80: These are called identification columns and they will be ignored by the Fortran compiler. These columns may be used to punch the name of the program (e.g. EIGEN) followed by a running number to identify the position of the card in the program. (e.g. EIGEN 001, EIGEN 002,....,EIGEN 098,...).

A punched card illustrating the combination of punches for various characters is shown below:



Let us now discuss the various types of Fortran statements which are allowed in a source program.

FORTRAN STATEMENTS

Comment Statement: If the letter C is punched in the first column of a card, then the Fortran compiler will ignore the instruction or statement on this card. Such a card is called a Comment Card and is used by the programmer for identification purposes. Columns 4 through 72 on IBM 1620 and 2 through 80 on CDC 3600 are available for comments. If one card is not sufficient, more cards can be used but every comment card must have the letter C in the first column. Comment cards are used to provide

headings for the programs. Though the Fortran compiler does not process the information on the card, it prints the information while "listing" the source program. Comments are particularly helpful to the programmer when he returns to a program after a long time. Any number of comment cards can be used in a program.

Arithmetic Statements: Statements which specify the manner in which variables should be computed are called Arithmetic Statements. An arithmetic statement in the Fortran language has the form but not the meaning of a mathematical equation. e.g. A = 2.0*B + 4.0 * C**2 + D. The left side of the equality symbol in an arithmetic statement is the name of a variable which may be an unsigned fixed-point or floating-point variable. The right side of all arithmetic statements are expressions. The arithmetic statement instructs the computer to evaluate the expression on the right side of the equals sign and place the resultant value in the location for the variable whose name is given on the left side of the equals symbol. In other words, the equality sign in Fortran literally means, "to evaluate the expression on the right side and assign the result to the variable whose symbolic name is on the left side". For example, the arithmetic statement

M = M+N

instructs the computer to increase the current value of M by the current value of N. The arithmetic statement

A = B

causes the variable A to assume the value that the variable B has at the time this statement is executed. The two sides of an arithmetic statement need not both be fixed-point or floating-point quantities. The statement

```
A = K-2
```

causes the present value of A to be replaced by the floating-point mode of the present value of K-2. But the statement

```
Y = (K-2)**X**(K-3)
```

is not admissible because it violates the rule against the occurrence of mixed expressions. It must be replaced by the two statements

```
A = K-2
Y = A**X**(K-3)
```

The arithmetic statement

```
L = A + B
```

causes the present value of the fixed-point variable L to be replaced by the integral part of the present value of the floating-point quantity A+B, the integral part of a number being defined as the positive or negative integer that is closest to the number without exceeding it in absolute value. For example, the integral parts of 3.8 and -5.9 are 3 and -5, respectively.

Consider the quadratic equation:

$$ax^2 + bx + c = 0$$

whose roots are: $x = \frac{-b \pm \sqrt{b^2-4ac}}{2a}$. If a=5, b=6.3 and c =-18.9 the Fortran program that describes the computation is as follows:

```
A = 5.0     (or A = 5)
B = 6.3
C = -18.9
RΦΦT 1 = (-B + (B**2 - 4.0*A*C))/(2.0*A)
RΦΦT 2 = (-B - (B**2 - 4.0*A*C)/(2.0*A)
```

Of course, the program could also have been written as:

```
RΦΦT 1 = (-6.3 + (6.3**2 - 4.0*5.0*(-18.9)))/(2.0*5.0)
RΦΦT 2 = (-6.3 - (6.3**2 - 4.0*5.0*(-18.9)))/(2.0*5.0)
```

Input/output statements:

If a problem is to be done only once, all data can be incorporated, in the program itself, in the form of constant statements. But, programs are normally written to carry out the computation for as many sets of data as desired. Hence, programs are usually set up to read the problem's data from cards at the time the program is executed, and constants are used for quantities which are really constant throughout the computation for all the data cards. One of the most important topics, yet difficult, in Fortran programming is the handling of a class of input/output statements. The problem becomes complex when an array or matrix is involved. Let us first consider the input/output statements applicable for single variables.

Of the many possible Input/Output statements in the Fortran language let us consider here only the essential Input/output statements applicable to IBM 1620, IBM 1130 and CDC 3600 computers.

The Input statement is a READ statement, which contains a statement number and a List of variables. The List of variables can contain any number of fixed-and/or floating-point variables, each being separated from the other by a comma. The statement number in the READ statement corresponds to a FORMAT statement which specifies the mode and form in which the list of variables are read. Table 1. below gives the general form and some examples of the READ statement as applicable to the three computers mentioned above.

Table 1.

IBM 1620 CDC 3600	IBM 1130
General Form: READ n, List of variables where n is the statement number of the FORMAT statement.	General Form: READ (2,n) List of variables; where 2 in parenthesis is the designation of the input unit that reads the data card and n the statement number of the FORMAT statement.
Examples: READ 2, A, B, C, D, E, F READ 444, ALPHA, BETA, GAMMA READ 45, ASS, FOOL, KING	Examples: READ (2,2) A,B,C,D,E,F READ (2,444) ALPHA, BETA, GAMMA READ (2, 45) ASS, FOOL, KING

The READ statement instructs the computer to read numerical data (or other material) from one or more cards.

The output statement is a PUNCH, WRITE or a PRINT statement, followed by a statement number and a List of variables. The statement number corresponds to a FØRMAT statement which specifies the mode and form in which the list of variables are to be punched, written or printed. Table 2. gives the general form of the output statement and some examples, as applicable to the three computers mentioned above.

Table 2.

IBM 1620	IBM 1130	CDC 3600
General Form: PUNCH n, List of variables; where n is the statement number of the FØRMAT statement.	WRITE (3,n) List of variables; where 3 in parenthesis refers to the output unit onto which the variable values are to be written and n is the statement number of the FØRMAT statement.	PRINT n, List of variables; where n is the statement number of the FØRMAT statement.
Examples:		
PUNCH 45,A,B,C,D, E,F	WRITE(3,45)A,B,C,D,E,F	PRINT 45,A,B,C,D,E,F
PUNCH 555,ALP,CRØS, RESLT	WRITE(3,555)ALP,CRØS,RESLT	PRINT 555,ALPS,CRØS, RESLT

In the IBM 1620 installation, at the Fundamental Engineering Research Establishment, Guindy, the results of the computation come out as a punched deck of cards, which are later listed on an IBM Listing machine. In the IBM 1130 installation, at the Advanced Research Centre for Physics, Guindy, the Computer is

coupled to an on-line printer so that the output is in typewritten form. In the CDC 3600 installation, at the Tata Institute of Fundamental Research, Bombay, the results are first obtained on a Tape which is then converted into a typewritten form by a Listing machine.

FORMAT statement and field specifications:

The numerical values of the Input/Output variables can be specified in one of three field specifications. The labels I,F and E are used to distinguish these three notations. The use of plus sign is optional in these three notations. I specification is used, if the value of the variable is a fixed-point constant (or integer). F specification is used if the value of the variable is a floating-point constant without exponent. E specification is used if the value of the variable is a floating-point constant with exponent. These specifications when used in FORMAT statements have the following general forms:

Iw

Fw.d

Ew.d

where w represents the width including sign (if any) and decimal point (if any) of the data and d represents the number of places to the right of the decimal point. Both w and d are fixed-point constants.

When the variable is read in one of the three specifications mentioned above, the computer converts the external

notation into an internal notation with the help of special subroutines. The computer works with the internal notation only.

Iw specification: When used as an Input specification, the computer's subroutine examines w columns for the input quantity. If a positive or negative sign is present, it is included in the count. The integer must be right-justified in the input field.

Examples:

Input data	field specification in FORMAT statement
^81	I3
1421	I4
-1421	I5
10	I2
5	I1
^^42	I4
+67	I3

(^ is used to indicate a blank space)

When used as an output specification, w places are reserved for the number. The space for the sign must also be included in the count w of the output field. The number is right-justified in the output field.

Examples:

value of output variable	Field specification in FORMAT statement	Print-out
613	I3	613
12	I3	^12
9	I3	^^9
+13	I7	^^^^+13
8666	I3	666
-10	I2	-0

The last two examples, give a wrong print-out since the field specification is insufficient.

Fw.d specification: When used as an input specification, w specifies the number of places reserved for the input variable: including sign (if any) and decimal point (if any). Because, the number of decimal places are specified in the d part of the specification, the actual decimal point need not be punched on the input data card. The input quantity must be right-justified in the input-field.

Example:

Input data	Field specification in FORMAT statement
123.456	F 7.3
-123.456	F 8.3
811.12123	F 12.7
3987	F 4.2 (39.87)
1	F 1.0 (1.0)
315	F 3.2 (3.15)

When used as an output specification, w places are reserved for the number. The spaces for the decimal point and sign (if any) must also be included in the count w. If the number of places reserved for the integer portion of the quantity is insufficient, the F specification is ignored and the number is placed in the F 14.8 specification.

Examples:

value of output variable	Field specification in FORMAT statement	Print-out
32.1	F 8.4	32.1000
-0.9	F 5.2	-.90
-8.0	F 5.1	-8.0
-397.221	F 8.3	-397.221
41.6745	F 5.2	0.41674500E+02

The last statement is an example of an insufficient field specification (F 5.2) for the value of the variable to be printed. The computer will point this out as an error while executing, but it will also print-out the result in F14.8 specification.

Ew.d specification: When used as an input specification, w specifies the field width including sign (if any), decimal point (if any) and exponent. The d specifies the number of decimal places. Because of this, the actual decimal point need not be punched.

Examples:

Input data	Field specification in FORMAT statement
200.674E+13	F 11.3
-2.98E-16	F 9.2
+100.648E-16	F 12.3
98.E+15	F 7.0
101E+14 (10.1×10^{14})	F 7.1

When used as an output specification, the field width w includes four places for the exponent, one space for the decimal point and one space for the sign. Therefore, the difference between the integers w and d must be, at least, six. If the field specification is insufficient, then the computer automatically prints the result in E 14.8 specification.

Examples:

Value of output variable	Field specification in FØRMAT statement	Print-out
-67.3211	E 13.7	-.6732110E+02
982.	E 10.3	9.820E+02
.00000132	E 10.3	1.320E-06
-642.0068	E 11.4	-6.4200E+02
12345678.0	E 10.0	1234.E+04

The general form of a FØRMAT statement is given below:

n FØRMAT (specification for list of variables in I/O)

where n is the statement number referenced by an Input/Output (I/O) statement. An example of a READ--FØRMAT combination may be as below:

READ 123, J,K,X,Y

123 FØRMAT (I6,I7,F6.3,E14.8)

The statement number (123) of the FØRMAT statement also referenced in the READ statement, may be replaced by any other integer of up to five digits that is not used anywhere else in the program. Assuming that the variables are READ from a card input, the above combination would instruct the computer to do the following:- The quantity in columns 1 to 6 would be analyzed according to I6 specification and assigned to the variable J; the quantity in columns 7 through 13 would be analyzed according to I7 specification and assigned to the variable K; the quantity in columns 14 through 19 would be analyzed according to F6.3

specification and assigned to the variable X; and finally the quantity in card columns 20-through 33 would be analyzed according to E 14.8 specification and assigned to the variable Y.

Abbreviations such as (2I10,3E15.8) for (I10,I10,E15.8, E15.8,E15.8) are allowed in FORMAT statements. For example,

```
READ 60,  K,L,R,S,T.E
60 FORMAT (2I5, 4E15.8)
```

A FORMAT statement may set aside more space than is required by the list of variables in the READ statement. For example,

```
READ 44, A1, A2, A3
44 FORMAT (5F14.8)
```

This feature often makes it possible to combine the FORMAT statements for several input variables, as in:

```
READ 555, I, K, X, Y
READ 555, J, L, P, Q, R, S
555 FORMAT (2I5, 4E14.8)
```

FORMAT statements can make use of three more specifications in addition to those mentioned above. These are denoted by the labels X, H and /.

X specification:

The X-type specification is used either to skip certain number of columns while reading-in the input (from a card) or to leave blank spaces while printing the output. The general form is

where w is an unsigned fixed-point constant (integer) indicating the width of the field, or, the number of blanks to be provided or the number of columns to be skipped. When X-type specification is used, it need not be followed by a comma.

Example:1.

```
READ 2, ALPHA, BETA, GAMMA
2 FORMAT (F6.3, 2XF6.3, 4 X E12.6)
```

The corresponding data card from which the values of ALPHA, BETA and GAMMA are to be read should have the value of ALPHA punched in columns 1 to 6 in F6.3 specification; while the values of BETA in F6.3 specification and GAMMA in E12.6 specification should be punched only in card columns 9 to 14 and 19 to 30, respectively. Even if columns 7 to 8 and columns 15 to 18 contain characters, they will be skipped (ignored) by the computer.

Example:2.

```
PRINT 36, I1, I2, J1, J2
36 FORMAT (I8,5XI3,5XI5,5XI3),
```

According to these PRINT-FORMAT statements, the computer will print the four fixed point integers in their respective Iw specifications with five blanks between each one of them.

H specification:

H specification or Hollerith text is useful for giving headings to the output to be printed. The general form of the specification is

wH followed by w characters

in a FORMAT statement. If blanks are desired they are included in the count. The field width w is an unsigned fixed point constant (integer). The Hollerith statement is also used to introduce spacing between printed lines. The spacing specification consists of three characters immediately following the opening parenthesis of a FORMAT statement. The first two of these three characters are 1H and these two characters are followed by a blank space, 0,1,2,3,4,5 or 6. Explicitly,

1H$\wedge$ will produce single spacing before printing,
1H0 will produce double spacing before printing,
1H1 will cause skipping to the head of the next page,
1H2 will cause skipping of a half page,
1H3 will cause skipping of a third of a page,
1H4 will cause skipping of a fourth of a page,
1H5 will cause skipping of a fifth of a page,
1H6 will cause skipping of a sixth of a page.

Thus, the first character after H will serve as an indication of the spacing that is to occur before the rest of the line is to be printed and this first character will not be printed.

The following is an example of the use of the Hollerith text to provide a heading to the output.

Example:1.

```
      PRINT 11
   11 FORMAT (24H ^RESULTS ^OF ^FIRST^ TRIAL.)
```

This PRINT-FORMAT combination will yield the printed message:

RESULTS OF FIRST TRIAL.

The PRINT statement which calls exclusively for the printing of the Hollerith text contains only a reference to the statement number of the following Format specification card. If a blank were not left after 24H in the Format statement (i.e. if the Hollerith text is punched immediately after 24H in the Format statement) then the output message will be:

ESULTS OF FIRST TRIAL.

Therefore, while specifying the field width for the Hollerith text, the count for spacing indication and the count for the Hollerith text must be combined.

Using the H specification one can READ in a card and PRINT out the same information. This can be accomplished by the following three statements:

Example.2:

```
      READ 25
   25 FORMAT (30H<-----30 blank spaces----> )
      PRINT 25
```

Assume that the card to be read contained the following information in columns 1 to 29:

LEGENDRE POLYNOMIAL GENERATOR

Then this information would be printed in the output as such. This is called as "Echo print".

The H and X specifications can be suitably mixed to give the following printed output:

P = -15.213 Q = 256.867 R = 123.567 IJ=133

by using the following two statements: (Example.3)

Example.3:

```
PRINT 175, P,Q,R,IJ
175 FORMAT(1H02XP=F7.3,5X2HQ=F7.3,5X2HR=F7.3,5X3HIJ=I3)
```

Slash or Solidus specification:

The slash or solidus (/) may be used in the Format statement corresponding to an input or output statement. A comma need not follow the slash when it is used in a Format statement.

Example.1:

```
READ 2, A,B,I
2 FORMAT(F6.3/E14.8/I3)
```

This READ-FORMAT combination makes the computer read the value of A from the first data card, B from the second and C from the third.

In general, k slashes in a Format statement will cause k blank lines in the print-out if the slashes appear either before or after the last specification. But, k+1 slashes will cause only k blank lines in the print-out, if the slashes are between any two other forms of Format specifications.

Example.2:

```
PRINT 10, A,B,K,C
10 FORMAT (F10.3,E12.5/I6,F6.3)
```

This Format specification will cause the first two variables to be printed on the first line and the second two variables to be printed on the second line.

Example.3:

```
PRINT 36
36 FORMAT (18H SERIES SUMMATION.//)
```

This Print-Format statement combination will cause the Hollerith text to be printed and then two lines to be skipped.

If a set of variables are to be READ or PRINTed in the same specification, then we can make all the values be Read (Printed) from (on) the same card (line) or make each variable be Read (Printed) from (on) a different card (line).

Example.4:

```
READ 63, P,Q,R,S,T
63 FORMAT (5F6.3)
```

will make all the variables be read from the same card while

```
63 FORMAT (F6.3)
```

will make the computer read each variable from a different card.

Example.5:

```
PRINT 25, X,Y,Z,W
25 FORMAT (4E13.6)
```

will cause all the variables to be printed on the same line, whil

```
25 FORMAT (E13.6)
```

will cause each variable to be printed on different line .

Notice that in the above examples the right paremthesis, like a single slash or solidus, resets the printer for a new reco

CONTROL STATEMENTS:

PAUSE, STΦP and END statements are called control statements. Of these the PAUSE and STΦP statements take effect only when the object program is executed, i.e. they do not cause the termination of compilation. When the computer encounters these statements while executing the object program, it comes to a halt in the manual mode. If the computer comes to a halt in the manual mode due to a PAUSE statement, then the operator can by a manual operation on the computer consol(explicitly, by pressing the START switch) resume the execution of the program from where the computer left off. But, if the halt is due to a STΦP statement then the execution of the program cannot be resumed by a manual operation on the computer consol by the operator.

The END statement must be the last statement in the Fortran source program. This statement indicates to the compiler that there are no more Fortran statements to be compiled in the source program.

While there can be one or more PAUSE and STΦP statements in a source program, there must be one and only one END statement in a source program. After the END card, the data cards, if any, should follow.

With these basic Input/output and Format statements at our command, we can write simple computer programs.

Program.1:

Let us write a program to evaluate the roots of the quadratic equation: $ax^2 + bx + c = 0$ for a,b and c given in F6.3 notation (say).

```
    READ 2, A,B,C
2   FORMAT (3F6.3)
    STEP 1 = (B**2 - 4.0*A*C)**0.5
    DENOM = 2.0*A
    ROOT 1 = (-B + STEP 1)/DENOM
    ROOT 2 = (-B - STEP 1)/DENOM
    PRINT 8 , A,B,C, ROOT 1, ROOT 2
8   FORMAT (3H A=F6.3,2X2HB=F6.3,2X2HC=F6.3,2X6HROOT1=E13.6,
   1 2X6HROOT2=E13.6)
    STOP
    END
```

Exercise.3:

If A,B and C are punched in a card as follows:

card columns	variable name	Form
1-6	B	XXX.XX
10-14	C	XXX.X
20-24	A	XX.XX

What changes should be made in the above program?

Program.2:

Given a,b and c in F7.3 specification, write a program to evaluate

$$F = \frac{1 + a}{1 + \frac{b}{c+6}}$$

and print the result in E 20.8 field specification.

```
      READ 25, A,B,C
   25 FORMAT (3F7.3)
      ANR = 1.0 + A
      D1 = C+6.0
      D2 = B/D1
      DNR = 1.0 + D2
      F = ANR/DNR
      PRINT 33, A,B,C,F
   33 FORMAT (3H A=F7.3,2X2HB=F7.3,2X2HC=F7.3,
     1 2X2HF=E20.8
      STOP
      END
```

Exercise.4:

Write a program to evaluate the value of:

$$R = (13.6 - x)^{n+4}$$

given that x is a floating point variable with a width of 10 and that n is an integer with a field width 5. The output should contain x,n and R.

Exercise.5: Write a program for evaluating:

$$r = \frac{bc}{12}\left[6x^2\left(1-\frac{x}{a}\right)+b^2\left(1-\frac{x}{a}\right)^2+c^2\left(1-\frac{x^2}{b^2}\right)^{1/2}\right]$$

READ: a,b,c,x in F6.3 specification.

PRINT: a,b,c,x,r with r in Ew.d specification.

TRANSFER OF CONTROL STATEMENTS:

Normally, Fortran statements are executed by the computer in the sequence in which they occur in the source program, until the STØP statement is encountered. But, often we may wish to return to the beginning of the program to execute it again with different data. Or, we may wish to branch around a section of the program depending upon the values of intermediate computed results. Statements which make the computer execute source program statements in other than the normal one-after-the-other sequence, are called control or TRANSFER OF CONTROL or branch statements. There are two types of transfer of control, **viz. unconditional and conditional transfer of control statements. Transfer of control should be made to executable statements only. A transfer to a non-executable statement will result in a programming error.**

Unconditional Transfer of Control Statements:

The simplest statement which interrupts the normal sequential mode of execution is called the GØ TØ Statement. The general form of this statement is:

GØ TØ n

where n is a statement number. The GØ TØ n statement in a source program directs the computer to unconditionally go to execute the statement bearing the statement number n, instead of executing the next one in sequence in the source program. The statement (with statement number n) referred to by the GØ TØ

statement is allowed to be <u>any executable</u> statement in the source program, either before or after the GΦ TΦ statement itself. (An example of a non-executable statement, encountered so far, is the FΦRMAT statement).

<u>Example</u>.

```
    1 | READ 2, A,I,J,K,B
    2 | FΦRMAT (F6.3,3I2,F7.3)
      |  '     '
      |  '     '
      |  '     '
      |  '     '
      | GΦ TΦ 1
   45 | STOP
      | END
```

In the above example, the unconditional GΦ TΦ 1 statement before the STΦP statement makes the computer READ a new set of data (from data cards placed after the END card) and then re-execute the program with the new values of the variables A,I,J,K and B. Due to the unconditional nature of the GΦ TΦ 1 statement, the computer will never encounter the STΦP statement. But, the computer, after executing all the data cards will print an error statement (indicating the incorrect termination of the program) and then it will automatically come to a stop. Note that a statement.number, 45, has been given to the STΦP statement. This is essential in the case of some computers.(If the statement following the GΦ TΦ statement does not have a statement number, then this will be indicated as an error by the computer.)

Conditional Transfer of Control Statements:

We will consider here two of the conditional transfer of control statements which are essential in any typical program. These are the computed GO TO and arithmetic IF statements.

The general form of the computed GO TO statement is:

GO TO $(n_1, n_2, \ldots, n_k)$, i

where $n_1, n_2, \ldots, n_k$ are statement numbers and i is an unsigned fixed point (integer) variable with $1 \leq i \leq k$. The computed GO TO statement provides a many-way branch based on the value of an integer variable. If the value of the integer variable i, at the time of execution of the computed GO TO statement is j, then control is transferred to the statement with statement number n_j. For example, the statement

GO TO (15, 44, 23), I

means that if the value of the integer variable I is 1, then control will be transferred to statement number 15; if it is 2, to statement 44; and if it is 3, to statement 23.

Example 1:

We are required at a certain point in a program to compute the value of one of the first five Legendre polynomials, defined as:

$$P_0(x) = 1$$

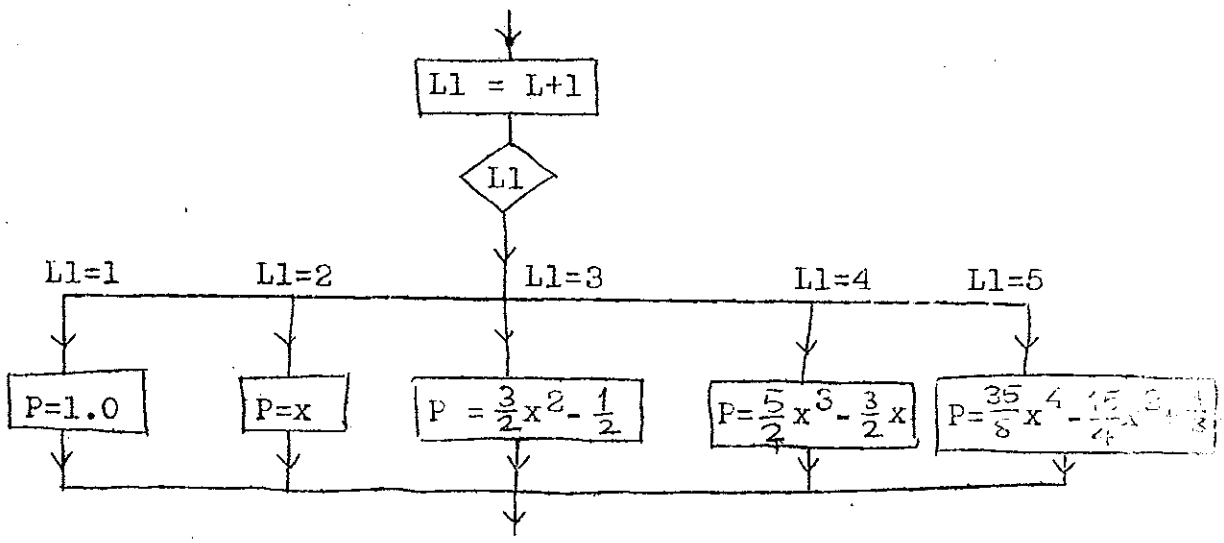
$$P_1(x) = x$$

$$P_2(x) = \frac{3}{2} x^2 - \frac{1}{2}$$

$$P_3(x) = \frac{5}{2} x^3 - \frac{3}{2} x$$

$$P_4(x) = \frac{35}{8} x^4 - \frac{15}{4} x^2 + \frac{3}{8}$$

Let us assume that x has been previously computed and that an integer variable L takes the values 0 to 4. It is the value of L which determines which of the five Legendre polynomials must be computed, i.e. if L=0, we are to compute $P_0(x)$; if L=1, we are to compute $P_1(x)$ and so on. We cannot use the computed Gϕ Tϕ statement straightaway, because of the restriction that the value of the integer variable must not be less than one. So, we must first define a new variable which takes the range 1 to 5 instead of 0 to 4. Let this new variable be L1. The flow chart for this problem is:



The sequence of Fortran statements which achieve this are:

```
    |  L1 = L+1
    |  G$ T$ (12,14,16,18,20),L1
 12 |  P = 1.0
    |  G$ T$ 21
 14 |  P = X
    |  G$ T$ 21
 16 |  P = 1.5*X**2 - 0.5
    |  G$ T$ 21
 18 |  P = 2.5*X**3 - 1.5*X
    |  G$ T$ 21
 20 |  P = 4.375*X**4 - 3.75*X**2 + 0.375
 21 |  .....
```

P is the value of whichever Legendre polynomial is computed.

The general form of the Arithmetic IF statement is:

$$IF(A)\ n_1,n_2,n_3$$

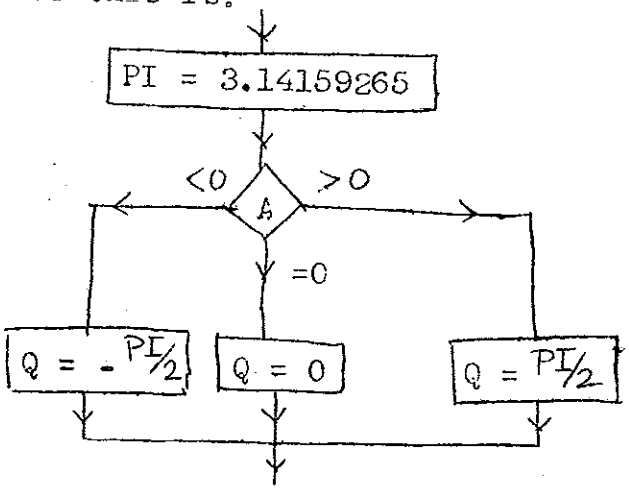
where A is any fixed or floating-point variable or expression and n_1,n_2,n_3 are three statement numbers. The Arithmetic IF statement is a conditional transfer of control statement causing transfer to statement numbered n_1,n_2 or n_3 depending on whether the sign of the discriminant of A is negative, zero or positive, respectively.

Example.2:

In a problem it is necessary to select the value of Q for A great than, equal to, or less than zero:

$$Q = -\pi/2 \quad \text{for} \quad A < 0$$
$$= 0 \quad \text{for} \quad A = 0$$
$$= \pi/2 \quad \text{for} \quad A > 0$$

The flow chart for this is:



The following are the Fortran statements in sequence for doing this:

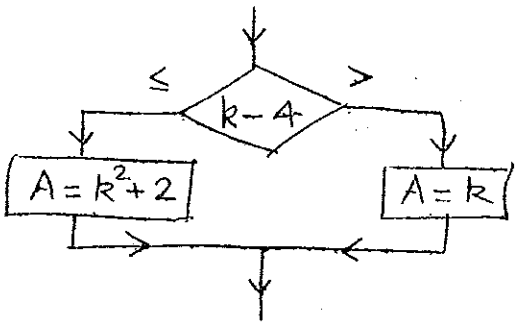
```
      PI = 3.14159265
      IF(A) 4,6,8
    4 Q = -PI/2.0
      GΦ TΦ 9
    6 Q = 0.0
      GΦ TΦ 9
    8 Q = PI/2.0
    9 ........
```

The above is an example for a three-way transfer of control using an IF statement. The following example illustrates the use of an IF statement as a two-way branch statement.

Example.3:

Let us draw a flow chart and a program segment for computing

$$A = k^2 + 2 \quad \text{for} \quad k \leq 4$$
$$= k \quad \text{for} \quad k > 4$$




```
      IF(K-4) 15,15,17
   15 A = K**2 + 2
      GΦ TΦ 18
   17 A = K
   18 .......
```

Occasionally use is made of the following statement:

IF (SENSE SWITCH i) n_1,n_2

where n_1 and n_2 are statement numbers and i is 1,2,3 or 4 in the case of IBM 1620. This statement causes control transferred to statement n_1 or n_2 depending on whether the console program switch i is on or off. For example,

IF(SENSE SWITCH 3) 65,25
(on) (off)

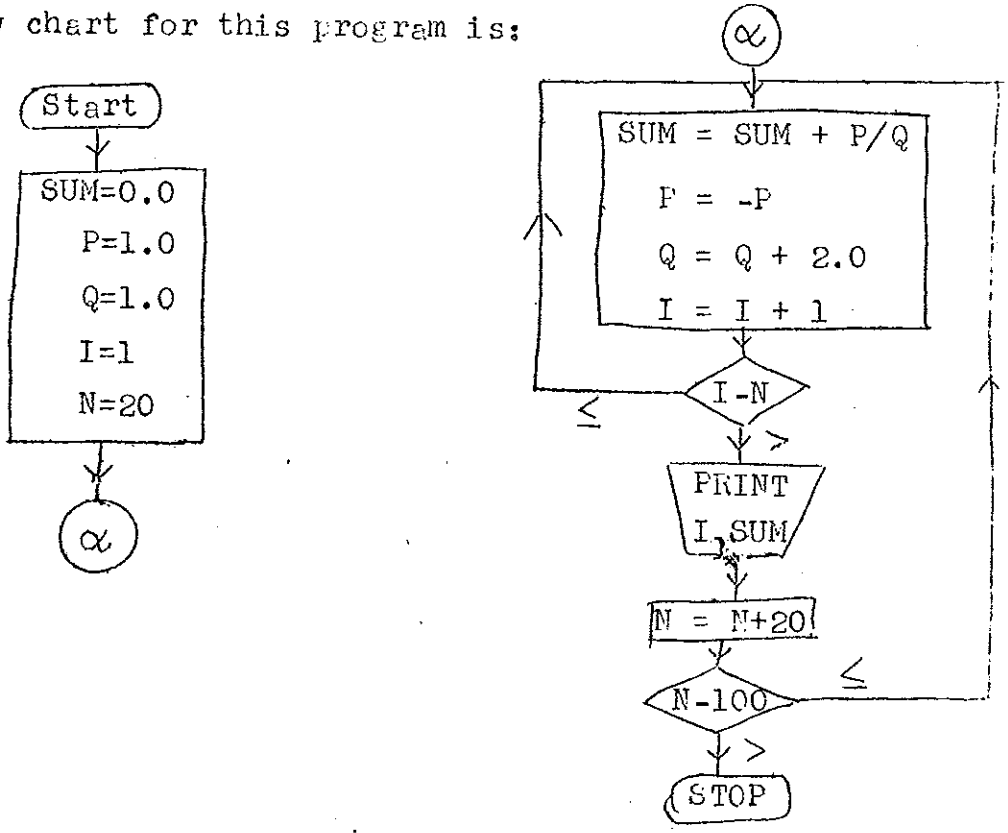
With the transfer of control statements at our disposal we can start writing realistic programs.

Program.3:

Let us write a program to compute the sums of the first 20,40,60,80 and 100 terms of the infinite series:

$$S = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \ldots.$$

The flow chart for this program is:



The program for this problem can be written with or without a READ statement. Let us write both the programs.

```
C       PROGRAM 3 USING READ STATEMENT
      1 READ 2,N
      2 FORMAT (I3)
        SUM = 0.0
        P = 1.0
        Q = 1.0
        I = 1
      7 SUM = SUM + P/Q
        P = -P
        Q = Q+2.0
        I = I+1
        IF(I-N)7,7,12
     12 PRINT 13, N,SUM
     13 FORMAT (2X2HN=I5,2X4HSUM=E20.8)
        GO TO 1
     15 STOP
        END
```

The values of N which are read are 20, 40, 60, 80 and 100 punched on five different data cards, the number being right justffied in the first three columns of the card.

```
C       PROGRAM 3 WRITTEN WITHOUT READ STATEMENT.
        N = 20
      1 SUM = 0.0
        P = 1.0
        Q = 1.0
        I = 1
      5 SUM = SUM + P/Q
        P = -P
        Q = Q+2.0
        I = I+1
        IF(I-N)5,5,10
     10 PRINT 11, I, SUM
     11 FORMAT (2X2HN=I5,2X4HSUM=E20.8)
        N=N+20
        IF(N-100)5,5,13
     13 STOP
        END
```

Notice that the former program is more general than the latter since in the case of the former program we can get the sum of any number of terms in the series by having suitable data cards while in the case of the latter many Fortran statements will have to be changed if the required results are not for sums of 20, 40, 60, 80 and 100 terms of the series.

Exercise.6: Write a program for

$$x = \sum_{k=0}^{500} C_k y^k$$

where $C_k = k$ for $0 \leq k \leq 250$

and $C_k = 3k^2$ for $251 \leq k \leq 500$, which can be computed for various values of y given in F10.5 format.

Exercise.7: Write a program for finding the sum of the first 25, 50, 75 and 100 terms of the series:

$$S = 1 - \frac{2}{3^2} + \frac{2^2}{5^2} - \frac{2^3}{7^2} + \ldots$$

without using a READ statement.

Program.4: To illustrate the use of the computed GO TO statement let us write a program for computing the following:

$$SUM1 = \sum_{k=1}^{10} k^3, \quad SUM2 = \sum_{k=1}^{10} (2k)^2, \quad SUM3 = \sum_{k=1}^{10} (3k)^4.$$

```
C       START OF PROGRAM 4A. NOTE THAT AK IS SUMMATION INDEX AND
        SUM1= 0.0                                          NOT K.
        SUM 2 = 0.0
        SUM 3 = 0.0
        K = 0
    5   AK = K+1
        SUM 1 = SUM 1 + AK**3
        SUM 2 = SUM 2 + (2.0 *AK)**2
        SUM 3 = SUM 3 + (3.0 *AK)**4
        K = K+1
        GO TO (5,5,5,5,5,5,5,5,5,11),K
   11   PRINT 12, K, SUM 1, SUM 2, SUM 3
   12   FORMAT (3H K=I5,2X5HSUM1=E13.6,2X5HSUM2=E13.6,
     1  2X5HSUM3=E13.6)13.6)
        STOP
        END
```

The following is an alternative way of writing the above program.

```
C       START OF PROGRAM 4B. ALTERNATIVE FORM OF PROGRAM 4A.
C       NOTE THAT K IS SUMMATION. INDEX HERE. AND NOTE THE
C       POSITION OF THE GO TO STATEMENT AS COMPARED TO THE
C       ABOVE VERSION.

        ISUM1 = 0
        ISUM2 = 0
        ISUM3 = 0
        K = 1
    5   GO TO (6,6,6,6,6,6,6,6,6,6,12),K
    6   ISUM1 = ISUM1 + K**3
        ISUM2 = ISUM2 + (2*K)**3
        ISUM3 = ISUM3 + (3*K)**4
        K = K+1
        GO TO 5
   12   PRINT 13, K, ISUM1, ISUM2, ISUM3
   13   FORMAT (3H K=I5,2X5HSUM1=E13.6,2X5HSUM2=E13.6,
     1  2X5HSUM3=E13.6)13.6)
        STOP
        END
```

It is obvious from the above example that use of the computed GΦ TΦ statement for indices which take large number of values (say 100) will be cumbersome. Further, computed GΦ TΦ is not always used to perform a loop in the program. For execution of a loop, use is made of DΦ and CΦNTINUE statements. Let us consider these statements in detail.

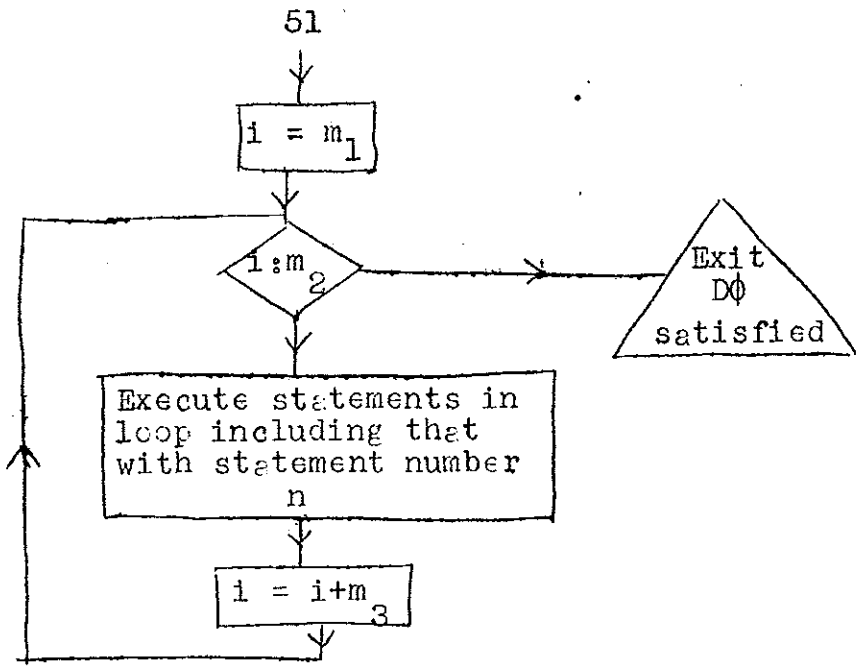
DΦ and CONTINUE Statements:

The DΦ statement marks the beginning of a loop and the CΦNTINUE statement usually marks the end of the loop. The general form of the DΦ statement is:

$$D\Phi \; n \; i = m_1, \; m_2, \; m_3$$

where n is the statement number of the statement ending the DΦ loop (usually it is the statement number of the CΦNTINUE statement); i is an unsigned fixed-point variable and m_1, m_2, m_3 are each either unsigned fixed-point constants or fixed-point variables.

The DΦ statement is a command to repeatedly execute the statements immediately following the DΦ statement upto and including the statement with statement number n, for values of the fixed point variable i starting from m_1 to m_2 in steps of m_3. If m_3 is not specified, it is taken to be 1 by the computer. Also, $0 < m_1 \leq m_2$. The DΦ loop will not be executed if initially either $m_1 > m_2$ or $m_1 = 0$. The flow chart for execution of a DΦ loop by the compiler is as shown below.



Examples of DO statements are:

(i)

```
      DO  15  K = 1, 10
      '       '
      '       '
      '       '
15    CONTINUE
```

(ii)

```
      DO  25  IDEAL = MIN, MAX, ISTEP
      '           '
      '           '
      '           '
25    - - - - - - - - - - - - - -
```

Now let us rewrite program 4, using the DO statement:

```
C     PROGRAM 4A REWRITTEN USING DO LOOP
      SUM1 = 0
      SUM2 = 0
      SUM3 = 0
      DO 11 K = 1,10
      AK = K
      SUM1 = SUM1 + AK**3
      SUM2 = SUM2 + (2.0 * AK)**2
      SUM3 = SUM3 + (3.0 * AK)**4
11    CONTINUE
      PRINT 13, SUM1, SUM2, SUM3
13    FORMAT (6HΔSUM1=E13.6,2X5HSUM2=E13.6,2X5HSUM3=E13.6)
      STOP
      END
```

The set of statements immediately following a DO statement upto and including the corresponding CONTINUE statement is called the range of the DO. When the index of a DO has assumed all the specified values, the DO is said to be satisfied and the next statement to be executed is the first statement following the range of the DO. This kind of exit from a DO loop is called normal exit. When a normal exit occurs the main value of the index (i) is lost and it may not be used anywhere else in the source program. However, the looping process specified by the DO may be curtailed by a transfer out of the range of the DO before it is satisfied. The exit now is called exit by transfer. If the exit occurs by a transfer which is in the range of several DO statements, the current values of all the indices controlled by them are preserved for any subsequent use.

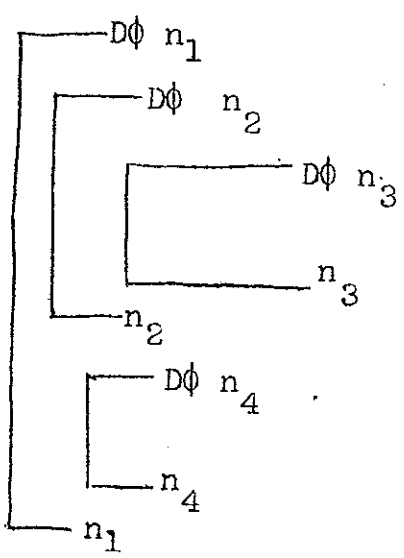
Nested DO loops:

When a DO loop contains another DO loop, the grouping is called a DO nest. The last statement of a nested DO loop must either be the same as the last statement of the outer DO loop or occur before it. The allowed nests are:

Example.1:

```
      DO  1  I = 1,20,2
      .   .
      .   .
      DO  2  J = 1,5
      .   .
      .   .
      DO  3  K = 2,8
      .   .     .
      .   .     .
3     CONTINUE  .
      .   .     .
2     CONTINUE  .
      DO  4  L = 1,3
      .   .     .
      .   .     .
4     CONTINUE  .
      .   .     .
1     CONTINUE
```

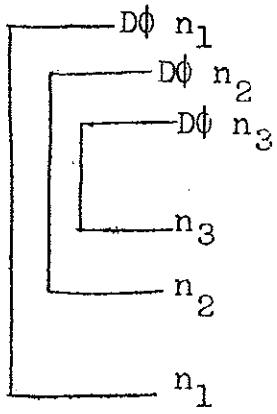
Schematically:



Example 2:

```
      DO  100  I = MIN, MAX
      DO  105  J = 1,5
      DO  110  K = 5,25,5
110   CONTINUE
105   CONTINUE
100   CONTINUE
```

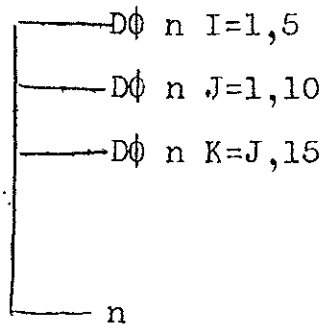
Schematically:



Example 3:

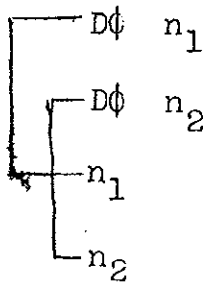
```
      DO  54  I = 1,5
      DO  54  J = 1,10
      DO  54  K = J,15
54    CONTINUE
```

Schematically:



It is important to note that a nest of DO loops may have a common ending point (as in example 3) but that the inner loop cannot end after the outer loops. The following example shows incorrectly ended loops:

Incorrect loops:



The CΦNTINUE statement is a dummy statement, most frequently used to terminate the DΦ loop. If CΦNTINUE is used anywhere else in the source program, it acts as a do-nothing instruction and control passes on to the next sequential statement in the program.

Rules governing the use of the DΦ statement:

1. The first statement in the range of the DΦ statement must be an executable statement i.e. the first statement after the DΦ statement must not be a CΦNTINUE, FΦRMAT, STΦP, END, (EQUIVALENCE, CΦMMΦN or DIMENSIΦN) statement.

2. If the range of a DΦ statement contains another DΦ statement, then the entire range of the inner DΦ statement must be part of the range of the outer DΦ statement. In no case should the range of an inner DΦ extend past the range of an outer DΦ.

3. The last statement in the range of a DΦ must not be a control or transfer of control statement. In other words, the last statement in the range of a DΦ must not be a STΦP, PAUSE, END, GΦ TΦ, IF or a DΦ statement.

4. The range of a DΦ must be entered only through the DΦ statement. Transfer into the range of a DΦ by an IF or GΦ TΦ statement outside this range is not permitted. In the case of nested DΦ loops, this rule prohibits a transfer from the range of an outer DΦ into the range of an inner DΦ, but it does

not prohibit a transfer from the range of an inner DO into the range of an outer DO. This statement is illustrated in Fig.1, where 2, 3 and 4 denote allowed transfers while 1,5 and 6 denote transfers which are prohibited.

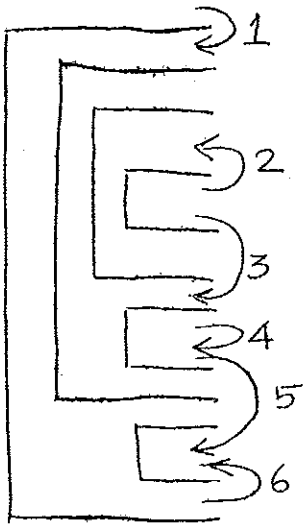


Fig.2

An exception to this rule is that of transferring out of a nested DO loop to execute some statements (outside the range of the nested DO loops) and then returning back to the nest. In Fig.2 'Extr.' represents a portion of the program outside the DO nest, and since the range of i includes the range of j, a transfer out of j and then a return into the range of either i or j is permitted.

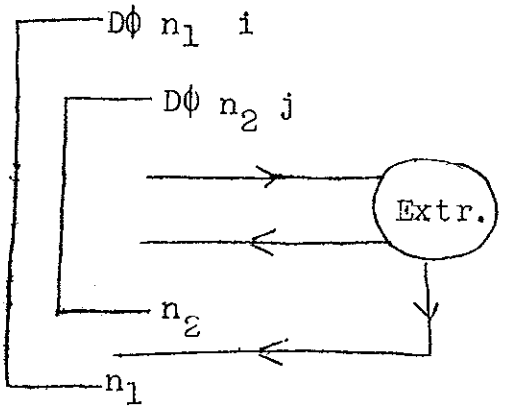


Fig.2

5. The range of a DO must not contain statements that change the values of the index or the value of the increment or the value of the upper bound. In other words, within the range of a DO loop, e.g. DO n i = m_1,m_2,m_3, change of the

value of i, m_1, m_2 or m_3 is not permitted.

6. The current value of the index of a DO is available for use in statements outside the range of the DO if and only if there has been an exit by transfer out of the loop.

Program.5:

We can rewrite program .3 using DO loops as follows:

```
      SUM = 0.0
      P = 1.0
      Q = 1.0
      DO 12 N = 20,100,20
      DO 9 I = 1,20
      SUM = SUM + P/Q
      P = -P
      Q = Q + 2.0
    9 CONTINUE
      PRINT 11, I, SUM
   12 CONTINUE
   11 FORMAT (2X2HN=I5, 2X4HSUM=E15.7)
      STOP
      END
```

Program.6: The following is another example of the use of DO loops. Compute

$$\sigma = \sum_{i=0}^{10} \sum_{j=0}^{M} \sum_{k=1}^{N} \left[kC + (B-i) + A^j \right]$$

where A,B,C are constants and N is specified.

```
      READ 2, A,B,C,N
    2 FORMAT (3F10.4,I2)
      SUMI = 0.0
      SUMJ = 0.0
      SUMK = 0.0
      DO 5 I = 1,11
      FI = I-1
    5 SUMI = SUMI - FI
      NP1 = N+1
      DO 10 J = 1, NP1
      FJ = J-1
   10 SUMJ = SUMJ + A**FJ
      DO 15 K = 1,N
      FK = K
   15 SUMK = SUMK + C*FK
      SIGMA = SUMK + B - SUMI + SUMJ
      PRINT 16, A,B,C,N, SIGMA
   16 FORMAT (3(F10.4, 2x), I2, E20.8)
      STOP
      END
```

SUBSCRIPTED VARIABLES:

In many mathematical problems we find ourselves making use of arrays or matrices. In mathematical notation the elements of the array are subscripted for ease of notation; e.g: a_i, b_{jk}, etc. Fortran provides for subscripting of variables, which makes it possible to refer to a complete array of data by one geometric name. Subscripted variables are useful in themselves but they have an added power when used in conjunction with the DO statement.

Any variable (fixed or floating-point) with a subscript in parenthesis after the name is called a subscripted

variable. An unsigned fixed-point constant which is different from zero, or an unsigned unsubscripted fixed-point variable are the most common forms of subscripts. The complete set of quantities is called an array and the individual quantities are called elements. A subscripted variable in FØRTRAN may have one, two or three subscripts, and it then represents a one-, two-, or three - dimensional array. (Note that here dimension refers to the number of subscripts only). Some versions of Fortran allow more than three subscripts, seven being the number permitted in advanced computers like IBM 7090, IBM 7094, IBM 360 H-level, Univac 1107, etc.

The one-dimensional array $x_1, x_2, x_3, \ldots, x_{20}$, is written in Fortran subscript notation as:

X(1), X(2), X(3),...,X(20)

The elements must always be numerical consecutively, starting with 1. An array of two rows and three columns might be written in mathematical notation as $a_{11}, a_{12}, a_{13}, a_{21}, a_{22}, a_{23}$. In Fortran subscript notation the elements would be written as:

A(1,1), A(1,2), A(1,3), A(2,1), A(2,2), A(2,3)

We note that the subscripts are separated by commas.

The most general admissible form of a subscript is the sum of two terms, the first is the product of an unsigned fixed-point constant by an unsigned unsubscripted fixed-point variable, while the second is a positive or negative fixed-point constant.

The following are examples of allowed subscript variable forms:

X(20), A(2,3), RADV(1,2,3);

ALPHA(NU), BETA(I), GAMA(MΦN), BETA(MU,NU);

I(K+4), KAPPA(5*I+3), B(7*K), DELTA(5*L-8),
Q(2*M+3,N-1)

Note that the use of subscripts such as 5*L-8 is only legitimate if the value of the subscript is positive for all values that L assumes in the course of computation. This restriction to positive subscripts in Fortran often requires all the subscripts in a formula to be increased by the same positive integer before it is programmed.

Note that quantities such as:

0, -3, -5 L, M(N), K*3+2, M+N, 4-J, 0.5, 2/3, R

cannot be used as subscripts in Fortran. So that, $a = bC_{l+m}$ must be evaluated only in two steps, as:

N = L+M

A = B*C(N)

A subscripted variable cannot be represented by a symbol that is used for a non-subscripted variable in the same program. Thus, while $x-x_j$ is unobjectionable in a formula, the Fortran compiler will not correctly interpret the expression X-X(J).

Example.1:

Suppose we have two points in space, represented in coordinate form by (x_1,x_2,x_3) and (y_1,y_2,y_3) and we are required to compute the distance between them, which is given by:

$$d = \left[(x_1-y_1)^2 + (x_2-y_2)^2 + (x_3-y_3)^2\right]^{1/2}$$

To program this, we set up a one-dimensional array called X, the three elements of which are the coordinates of X, and another similarly for Y. Then

```
D = ((X(1) - Y(1))**2 + (X(2) - Y(2))**2+(X(3)-Y(3))**2)**0.5
                                                    **0.5
```

is the Fortran statement that computes the distance between the two points.

Example.2:

$$SUMSQ = \sum_{i=1}^{20} x_i^2$$

We set up the 20 numbers, x_1 to x_{20}, as the elements of a one-dimensional array X. Now, any of the 20 numbers can be referred by the name X(I), where I takes the values from 1 to 20.

```
      SUMSQ = 0.0
      DO 25 I = 1,20
      SUMSQ = SUMSQ + X(I)**2
   25 CONTINUE
```

If, instead of using the array, we had given 20 different names as X1, X2,...,X20, then we could not have made use of the DO loop to find the sum of their squares. This example clearly reveals the power of the use of the subscripted variable in conjunction with the DO statement.

THE DIMENSION STATEMENT:

When subscripted variables are used in a program, the following information about them must be supplied to the Fortran compiler:

1. Which variable is subscripted?
2. How many subscripts are there for each subscripted variable?
3. What is the maximum size of each subscript?

These questions are answered by the DIMENSION statement. Every subscripted variable in a program must be mentioned in a DIMENSION statement, and this statement must appear before the first occurrence of the variable in the program. A common practice is to declare the DIMENSION information for all subscripted variables at the beginning of the program. The general form of this statement is:

DIMENSION $v_1, v_2, v_3, \ldots$

where $v_1, v_2, v_3, \ldots$ stand for fixed or floating-point variable names followed by parenthesis enclosing one, two, or three unsigned integer constants that give the maximum size of each subscript. When the Fortran compiler processes a DIMENSION statement, it sets aside enough storage locations to contain arrays specified by the information in the statement.

Examples:

```
DIMENSION  X(20), A(3,20), K(2,2,5)
```

The compiler will assign 20 storage locations to the one-dimensional array named X; 60(=3 x 20) storage locations to the two-dimensional array A; and 20(=2 x 2 x 5) storage locations to the three-dimensional array K.

Rules to be observed: In the DIMENSION statement:

(i) subscripts must never be smaller than 1,

(ii) zero and negative subscripts are not allowed:

(iii) subscripts of variables must be unsigned fixed-point constants and not variables.

The DIMENSIϕN statement is said to be non-executable, i.e. it provides information only to the Fortran compiler and does not result in the creation of any instructions in the object program. Though a non-executable statement may occur almost anywhere in the source program, as already pointed out, the DIMENSIϕN statement must not be the first statement in the range of a Dϕ statement and must appear before the variable name is used in the source program.(+)

Example.1: Two one-dimensional arrays A and B each contain 30 elements. Compute $D = \sum_{i=1}^{30} (A_i - B_i)^2$.

Let us write the program segment only:

```
      DIMENSIϕN  A(30), B(30)
      D = 0.0
      Dϕ 5  I = 1,30
      TERM(I) = (A(I) - B(I))**2
      D = D + TERM(I)
5     CϕNTINUE
```

Example.2: Given two (2 x 2) matrices. A and B write statements to compute the elements of the product matrix AB = C.

(+)Often the DIMENSIϕN statement is required to be the first Fortran statement in a program.

We know that

$$C_{11} = a_{11}\, b_{11} + a_{12}\, b_{21}$$

$$C_{12} = a_{11}\, b_{12} + a_{12}\, b_{22}$$

$$C_{21} = a_{21}\, b_{11} + a_{22}\, b_{21}$$

$$C_{22} = a_{21}\, b_{12} + a_{21}\, b_{22}$$

Hence the program segment is:

```
DIMENSION A(2,2), B(2,2), C(2,2)
C(1,1) = A(1,1)*B(1,1) + A(1,2)*B(2,1)
C(1,2) = A(1,1)*B(1,2) + A(1,2)*B(2,2)
C(2,1) = A(2,1)*B(1,1) + A(2,2)*B(2,1)
C(2,2) = A(2,1)*B(2,2) + A(2,2)*B(2,2)
```

Input/Output statements for subscripted variables:

If subscripted variables are to be listed for input or output, the subscripts must be unsigned fixed-point constants or variables only. Thus A(2*J+5), while admissible in an arithmetic statement, cannot be used in a list. In a READ statement, the elements of arrays can be explicitly written. In such a case the order of the elements is not specified. All that has to be ensured is that the data be punched in the data cards in the same order in which they are Read.

Program 7: Consider the problem of solving two simultaneous equations in two unknowns:

$$C_{11}\, x_1 + C_{12}\, x_2 = b_1$$

$$C_{21}\, x_1 + C_{22}\, x_2 = b_2$$

The solution using Cramer's rule is:

$$x_1 = \frac{b_1\, C_{22} - b_2\, C_{12}}{C_{11}\, C_{22} - C_{12}\, C_{21}}$$

$$x_2 = \frac{b_2\, C_{11} - b_1\, C_{21}}{C_{11}\, C_{22} - C_{12}\, C_{21}}$$

The problem can be conveniently solved by setting up two one-dimensional arrays, of two elements each, for b and x, and a two-dimensional array of four elements for C. A simple program for this problem would be as follows:

```
      DIMENSION  B(2), X(2), C(2,2)
      READ 2, C(1,1), C(1,2), C(2,1), C(2,2), B(1), B(2)
    2 FORMAT (6F 10.4)
      DENOM = C(1,1)*C(2,2) - C(1,2)*C(2,1)
      X(1) = (B(1)*C(2,2) - B(2)*C(1,2))/DENOM
      X(2) = (B(2)*C(1,1) - B(1)*C(2,1))/DENOM
      PRINT 6, X(1), X(2)
    6 FORMAT (6H X(I)=E14.7,5X6HX(2)=E14.7)
      STOP
      END
```

The elements of an array need not always be listed individually as in the above simple example. They can always be specified both in Input and output statements by indexing. Examples below show the allowed forms of list specifications.

Example.1: The list $a_3, a_7, a_{11}, a_{15}, a_{19}, a_{23}$ can be specified by

(A(I), I = 3,23,4)

where the three integers following the equality sign indicate the initial value, final value and the constant difference between consecutive values, respectively, of the subscript I.

Example.2: The list $a_1, a_2, a_3, \ldots, a_{10}$ is specified by:

(A(J), J = 1,10)

where the third integer which indicates the constant difference between successive values of J is omitted, since it is 1(as in the case of the DO statement).

Example.3: The range of the subscript can be given in terms of unsigned fixed-point variables, provided the values of variables have been previously read or computed. Thus,

(B(K), K = MIN, MAX, ISTEP)

represents $B_{Min}, B_{Min + Istep}, \ldots, B_{Max}$ where MIN, MAX and ISTEP must have been read or computed before this list is encountered. Needless to say, the abbreviation is meaningful if and only if $0 < MIN \leq MAX$.

Example.4: The list $a_1, b_1, a_2, b_2, \ldots, a_{20}, b_{20}$ may be specified by:

(A(I), B(I), I = 1,20)

which indicates that a single indication of the range of a subscript is applicable to more than one variable with this

subscript.

Example.5: The list M_{11}, M_{31}, M_{51}, M_{12}, M_{32}, M_{52}, M_{13}, M_{33}, M_{53} is specified by:

((M(I,J), I = 1,5,2), J = 1,3)

Notice that for each value of the outer subscript, the complete range of the inner subscript is covered.

Let us now consider an example of Read and Format statements used in conjunction for a subscripted variable.

Example.6:

```
READ  28, K, (A(I),I = 1,K)
28 FΦRMAT (I5/(4E15.8))
```

This combination causes the value of K to be read from a first data card, and the values of A(1) through A(K) from as many subsequent data cards as necessary, each card containing four values with the exception of the last card, which may contain less than four values.

Similarly, an example for a Print-Format combination is the following.

Example.7:

```
PRINT 145, (I, A(I), I = 1,50)
145 FΦRMAT (1H  4(I5, E14.7))
```

which would yield a print out of twelve lines each containing four sets of values of I and the corresponding A(I) and a last line which contains the last two sets of values of I and A(I).

Example.8: If we require an (N x N) matrix to be printed out, (for $N \leq 8$), this can be achieved by the statements:

```
PRINT 45, ((MATRX (I,J), J=1,N), I=1,N)
45 FORMAT (8F10.4)
```

Or equivalently by

```
DO 44 I = 1,N
DO 44 J = 1,N
44 PRINT 45, MATRX (I,J)
45 FORMAT (8F10.4)
```

Program.8: The following is an example of a Fortran Program to do matrix multiplication for matrices of a maximum size of 15 x 15, assuming that the elements are punched (one per card) by rows.

Given a matrix A with dimensions (N x L) and the matrix B with dimension (L x M), the resultant product matrix C will be of dimension (N x M).

To compute any element C_{ij}, select the i^{th} row of A and the j^{th} column of B, and sum the products of their corresponding elements, using the general formula:

$$C_{ij} = \sum_{k=1}^{l} A_{ik} B_{kj}$$

The computer program is as follows:

```
C       MATRIX MULTIPLICATION
        DIMENSION A(15,15), B(15,15), C(15,15)
        READ 2, L, N,M
    2   FORMAT (3I2)
        DO 5  I = 1,N
        DO 5  J = 1,L
    5   READ 6, A(I,J)
    6   FORMAT (F8.2)
        DO  9  I = 1,L
        DO  9  J = 1,M
    9   READ 6, B(I,J)
        DO 15 I = 1,N
        DO 15 J = 1,M
        C(I,J) = 0.0
        DO 14 K=1,L
   14   C(I,J) = C(I,J) + A(I,K)*B(K,J)
   15   PRINT 16, I, J, C(I,J)
   16   FORMAT (2I4,2XF8.2)
        STOP
        END
```

Exercise.8: Given a one-dimensional array named Y with 50 elements, and numbers U and I, write a statement to compute:

$$S = y_i + u\,\frac{y_{i+1} - y_{i-1}}{2} + \frac{u^2}{2}(y_{i+1} - 2y_i + y_{i-1})$$

This is called Sterling's interpolation formula through second differences.

Exercise.9: Given two one-dimensional arrays A and B, of seven elements each, each element being in F8.4 specification, with the seven elements of A punched on one card and the seven elements of B punched on another card, write a program to Read the cards and then to compute

$$ANØRM = \left(\sum_{i=1}^{7} A_i \, B_i \right)^{2/3}$$

and print ANØRM in E20.8 field specification.

SUBPRØGRAMS

In a program, if the same group of statements are found repeated many times, we can write a Subprogram for that group of statements. There are two kinds of Subprograms:

1. FUNCTIØN subprogram
2. SUBRØUTINE subprogram

The main advantages of subprograms are three in number:

(a) Subprograms avoid unnecessary duplication of effort in first writing and then punching the same group of statements in the source program, as well as wastage of storage space in the computer memory.

(b) Since the same subprogram can be used in many different main programs, they avoid a duplication of effort.

(c) Subprograms can be separately checked for errors, since they can be compiled independently of the main program of which they are a part.

Whatever be the motivation for their use, subprograms are a powerful feature of the programming language.

FUNCTIONS:

Three types of functions are used in Fortran. They are:

(i) Library Functions

(ii) Arithmetic statement Functions

(iii) Function subprograms.

1(i). LIBRARY FUNCTIONS:

Frequently in programs it becomes necessary to calculate trigonometric functions, logarithms, absolute values and exponentiation (raise e to a power). Most Fortran systems provide these commonly required functions. The exact list of available library functions depend not only on the computer and the version of Fortran used but on the particular installation. In his own interest, each programmer should acquire a list of library functions available at his installation.

General Form:

The names of the library functions are established and hence the programmer must use them exactly as specified. The name of the function must be followed by the argument enclosed in parenthesis. The argument may be an expression and, if desired, may contain another function. The argument can be even a subscripted variable.

We will give here only the list of library functions available at the IBM 1620 installation, at the Fundamental Engineering Research Establishment, Guindy:

Name of FUNCTION	Explanation
SINF(X) or SIN(X)	Sine of x.(x in radians)
COSF(X) or COS(X)	Cosine of x.(x in radians)
ATANF(X) or ATAN(X)	Arctangent of x(angle in radians)
SQRTF(X) or SQRT(X)	Square root of x
LOGF(X) or LOG(X)	Natural logarithm of x
EXPF(X) or EXP(X)	Exponential of x,viz.e^x.
ABSF(X) or ABS(X)	Absolute value of x.

For each of these functions, there exists a subroutine within the Fortran system that computes the function of the argument enclosed in parenthesis. These subroutines will be compiled into the object program automatically, when the function is encountered in any Fortran statement. (See Appendix 1. for a list of library functions available at the IBM 1130 installation, A.C. College of Technology, Madras-25.)

Examples:

b^2-4ac can be written as SQRT (B**2 - 4.0*A*C)

e^{ax+b} can be written as EXP(A*X + B)

$Sin(b_1+2)$ can be written as SIN(B(I) + 2.0)

1(ii). ARITHMETIC STATEMENT FUNCTIONS:

If a simple computation recurs in the same program, a single arithmetic statement Function can be set up to carry it out. The name of the Function is chosen by the programmer

following the rules that apply to a variable name. The name, of course, must not be the same as that of a library function. The name of the function is followed by parenthesis enclosing the argument(s), which must be separated by commas, if there is more than one. Neither the argument nor the expression on the right hand side of the definition statement should involve a subscripted variable.

These are only definitions of the functions and they do not cause any computation to take place and hence these must appear before the first executable statement of the program.

Example.1: Suppose that in a certain program it is frequently necessary to compute the two roots of the quadratic equation, $ax^2 + bx + c = 0$, given the values of a,b and c. Two arithmetic statement functions can be defined to carry out this computation. They are:

```
ROOT 1(A,B,C) = (-B + SQRT(B**2 - 4.0*A*C))/(2.0*A)
ROOT 2(A,B,C) = (-B - SQRT(B**2 - 4.0*A*C))/(2.0*A)
```

Example.2: It is known that

$$\phi(x) = x^2 - 2x+7 \; ; \; \theta(y) = 3y^2-14$$

$$\text{and } Q(x,y,z) = 3\phi(x)\sqrt{\theta(y)} - 2.4z^2$$

The following arithmetic statements can be set up for this:

```
PHI(X) = X**2 - 2.0*X + 7.0
THETA(Y) = 3.0*Y**2 - 14.0
Q(X,Y,Z) = 3.0*PHI(X)*SQRT(THETA(Y)) - 2.4*Z**2
```

Note: Arithmetic statement functions should come only after the DIMENSION statements, if a program contains both statements.

The following rules are to be adhered to by the programmer while setting up arithmetic statement functions:

(a) The name may have one to six characters, the first character being alphameric. Functions to be computed as fixed-point numbers must begin with I,J,K,L,M or N.

(b) The arithmetic statement function must precede the first executable statement in the source program.

(c) The arguments in parenthesis must not be subscripted.

(d) Each function is defined as a single arithmetic statement, which can be continued on the following card. (Note that the number of continuation cards should not exceed nine).

1(iii).FUNCTIØN subprograms.

The limitations of the arithmetic statement function are that it allows only one statement and it can compute only one value. One of these limitations, viz. restriction to one Fortran statement, does not exist in the case of a Function subprogram. A Function subprogram can have many Fortran statements but it is limited to computing only one value. The computation desired in a FUNCTIØN subprogram is defined by writing the necessary statements in a segment. The first statement of the Function Subprogram has the general form:

FUNCTIØN name (arguments)

The name is one to six characters in length; the first character being alphameric and chosen according to the naming convention for variables. The name of the Function subprogram is followed

by parenthesis enclosing the arugument(s), which are separated by commas, if there is more than one. The name of the Function subprogram is associated with a value and therefore it must appear at least once in the subprogram as a variable on the left hand side of an assignment statement. The arguments in the subprogram definition are dummy variables and must be distinct unsubscripted variables or array names. Within the subprogram, however, subscripted variables may be freely used. If subscripted variables are used in the subprogram, they must be declared in a DIMENSIϕN statement. The Function subprogram must contain at least one RETURN statement. There must be one and only one END statement, to specify the end of Fortran statements which belong to the Function subprogram to the compiler.

Therefore, a Function subprogram invariably contains the following statements:

```
FUNCTIϕN name (arguments)
DIMENSIϕN v1,v2,...
  .   .   .
name = .... .
RETURN
  .   .
  .   .
END
```

To <u>use</u> a Function subprogram, it is necessary only to write the name of the function where its value is desired, with suitable arguments.

The mechanism of the operation of the object program is as follows: the Function subprogram is compiled as a set of

machine instructions in one place in storage, and whenever the name of the function appears in the source program, a transfer to the subprogram is set up in the object program. When the computation of the subprogram has been completed, a transfer is made to the section of the program that brought the subprogram into action. The RETURN statement(s) in a subprogram signifies the conclusion of a computation and a transfer of the result of the computation (viz. the value of the function) back to the place in the main program from which the subprogram was referred.

Program.5: A Function subprogram which evaluates the value of $(-1)^n$ is given here:

```
      |FUNCTIΦN  PHASE(N)
C     |CΦMPUTES THE VALUE ΦF (-1)**N.
      |M = (N/2)*2
      |IF(N-M) 5, 3, 5
    3 |PHASE = 1.0
      |RETURN
    5 |PHASE = -1.0
      |RETURN
      |END
```

2. SUBRΦUTINE SUBPROGRAMS.

The Function subprograms which we have discussed so far, are, in a sense, smaller versions of SUBRΦUTINE subprograms. A SUBRΦUTINE subprogram can be used to compute as many results as desired. The general form of a subroutine subprogram is:

```
SUBRΦUTINE  name (arguments)
Fortran statements
RETURN
END
```

where 'name' is the name of the subprogram and the arguments used must be non-subscripted variable name, array name, or subprogram name (except the name of an arithmetic statement function).

Since the SUBRØUTINE is a separate program, the variables and statement numbers do not relate to any other program (except the dummy argument variables). The subroutine subprogram may use one or more of its arguments to return values to the main source program which calls it. Any arguments so used must appear on the left side of a Fortran statement or in an input list within the subprogram. When the argument is an array name, an appropriate DIMENSIØN statement must appear in the subroutine subprogram.

A SUBRØUTINE can be CALLed by a main program or another subprogram. The arguments in a SUBRØUTINE may be considered as dummy variable names which are replaced at the time of execution by the actual arguments supplied in the CALL statement. The actual arguments in the CALL statement must correspond in number, order and mode to the dummy arguments of the SUBRØUTINE subprogram.

The general form of the CALL statement is:

CALL name (arguments)

where 'name' is the name of the Subroutine subprogram and the arguments are the actual arguments being supplied to the SUBRØUTINE subprogram.

The CALL statement is a transfer of control statement. It transfers control to the SUBRØUTINE subprogram and replaces the dummy variables with the values of the actual arguments that appear in the CALL statement. The arguments in a CALL statement can be any of the following: any type of constant, any type of subscripted or unsubscripted variable, an arithmetic expression, or a subprogram name (except that they may not be names of arithmetic statement functions). The arguments in a CALL statement must agree in number, order, mode and array size with the corresponding arguments in a SUBRØUTINE subprogram.

A SUBRØUTINE cannot CALL itself, i.e. If Subroutine A calls subroutine B, then Subroutine B should not call subroutine A.

PRØGRAM.10:

To evaluate the roots of the Quadratic equation $ax^2+bx+c = 0$, we can write the following SUBRØUTINE subprogram:

```
      SUBRØUTINE QUADR(A,B,C, X1, X2)
      DISCR = B*B - 4.0*A*C
      IF (DISCR) 4, 6, 8
4     PRINT 5
5     FØRMAT (21H  RØØTS  ARE  IMAGINARY.)
      RETURN
6     X1 = -B/(2.0*A)
      X2 = X1
      RETURN
8     X1 = (-B + SQRT(DISCR))/(2.0*A)
      X2 = (-B - SQRT(DISCR))/(2.0*A)
      RETURN
      END
```

The following is a valid CALL statement to the subroutine:

```
CALL  QUADR(P,Q,R, RΦΦT 1, RΦΦT 2)
```

The following are invalid CALL statements to the subroutine:

```
CALL  QUADR(I,J,K,X1,X2)
CALL  QUADR(P,Q,R,X1,X2,Z)
```

because, the former contains arguments which do not agree in mode with those of the SUBRΦUTINE arguments and the latter contains six arguments while the SUBRΦUTINE has only five arguments.

PRΦGRAM.11: To illustrate the different natures of FUNCTIΦN and SUBRΦUTINE subprograms, let us write programs to evaluate:

$$\alpha = \left[\sum_{i=1}^{17} C_i D_i + 1 \right]^{1/2} + \sum_{j=1}^{15} E_j F_j$$

where C_i, D_i, E_j, F_j are one-dimensional arrays, using a subprogram to compute $\sum_{i=1}^{k} A_i B_i$.

Case (1) Using a FUNCTIΦN Subprogram:

```
      FUNCTIΦN  SUM (A,B,K)
      DIMENSIΦN A(20), B(20)
      SUM = 0.0
      DΦ 4  I=1,K
4     SUM = SUM + A(I)*B(I)
      RETURN
      END
```

The main program will then be:

```
C     MAIN PRØGRAM FØR CASE (1)
      DIMENSIØN C(20), D(20), E(15), F(15)
      READ 2, (C(I), D(I), I=1,17)
      READ 2, (E(I), F(I), I = 1,15)
    2 FØRMAT (10F7.3)
      ALPHA = SQRT (1.0 + SUM(C,D,17)) + SUM(E,F,15)
      PRINT 5, ALPHA
    5 FØRMAT (7H ALPHA = E20.8)
      STØP
      END
```

According to the first READ statement, the 34 values of C_i and D_i will be read in the sequence $C_1, D_1, C_2, D_2, \ldots, C_5, D_5$ from the first data card; then $C_6, D_6, C_7, D_7, \ldots, C_{10}, D_{10}$ will be read from the second data card; the value C_{11}, D_{11}, C_{12}, $D_{12}, \ldots, C_{15}, D_{15}$ will be read from the third data card and the last four values C_{16}, D_{16}, C_{17}, D_{17} will be read from the fourth data card. According to the second READ statement, the values of $E_1, F_1, \ldots, E_6, F_5$ will be read from the fifth data card; $E_6, F_6, \ldots, E_{10}, F_{10}$ from the sixth data card and the values $E_{11}, F_{11}, \ldots, E_{15}, F_{15}$ from the seventh and last data card. This is so, since each data card contains 10 numerical values in F7.3 specification.

Case(2). Using a SUBRØUTINE subprogram:

```
      SUBRØUTINE SUMM (A,B,K,SUM)
      DIMENSIØN A(20), B(20)
      SUM = 0.0
      DØ 4 I = 1,K
    4 SUM = SUM + A(I)*B(I)
      RETURN
      END
```

The corresponding main program will now be:

```
C       MAIN PRΦGRAM FΦR CASE(2).
        DIMENSIΦN C(20), D(20), E(20), F(20)
        READ 2, (C(I), D(I), I = 1,20)
        READ 2, (E(I), F(I), I = 1,15)
      2 FΦRMAT (10F7.3)
        CALL SUMM(C,D,17,SUM1)
        CALL SUMM(E,F,15,SUM2)
        ALPHA = SQRT(1.0 + SUM1) + SUM2
        PRINT 5, ALPHA
      5 FΦIMAT (7H ^ ALPHA = E20.8)
        STΦP
        END
```

In this example, if we had not written a subprogram, we would have written the DΦ loops for doing $\sum_{i=1}^{17} C_i D_i$ and for doing $\sum_{j=1}^{15} E_j F_j$, which repetition has now been avoided.

The essential features of the four types of subprograms, viz. library functions, arithmetic statement functions, Function subprograms and subroutine subprograms are summarized in Table.3 below:

Table.3

	Library Function	Arithmetic Statement Function	FUNCTIΦN subpro-gram	SUBRΦUTINE subprogram
Naming	Supplied with the compiler (by manu-facturer)	1 to 6 charac-ters, first being alpha-meric.	1 to 6 charac-ters, first being alpha-meric	1 to 6 charac-ters, first being alpha-meric.
Type	Implied by first character of the name	Implied by first charac-ter of the name	Implied by the first character of the name	No specifica-tion in the subrou-tine name.

Table.3(ctd.)

/	Library Function	Arithmetic Statement Function	FUNCTIΦN subprogram	SUBRΦUTINE subprogram
Definition	-	One arithmetic statement before first usage of function	Any number of statements after FUNCTIΦN statement.	Any number of statements after SUBRΦUTINE statement.
Calling	By writing name of function	By writing name of function	By writing name of function	By a CALL statement
Arguments	One or more as defined	One or more as defined	One or more as defined	Any number, including none, as defined
Output	One value associated with the name of the function.	One value associated with the name of the function	One value associated with the name of the function.	Any number, as specified in the arguments.

Exercise.10: Define a statement function to compute $x^2 + \sqrt{1+2x+3x^2}$ and then use the function to compute:

$$\alpha = \frac{6.9 + y}{y^2 + \sqrt{1+2y+3y^2}} \; ; \quad \beta = \frac{2.1Z + Z^4}{Z^2 + \sqrt{1+2Z+3Z^2}} \; ;$$

$$\gamma = \frac{\text{Sin } y}{y^4 + \sqrt{1+2y^2+3y^4}} \; ; \quad \delta = \frac{1}{\text{Sin}^2 y + \sqrt{1+2\text{sin}y+3\text{sin}^2 y}}$$

Exercise.11: Write a FUNCTIΦN subprogram to compute:

$$y(x) = \begin{cases} 1 + \sqrt{1+x^2}, & x < 0 \\ 0, & x = 0 \\ 1 - \sqrt{1-x^2}, & x > 0 \end{cases}$$

and then write statements to evaluate the following:

$$f = 2 + y(a+z)$$

$$G = \frac{y[x(k)] + Y[x(k+1)]}{2}$$

$$h = y(\text{Cos } 2\pi x) + \sqrt{1+y(2\pi x)}$$

EQUIVALENCE statement:

When a program contains several subscripted variables whose subscripts assume many values, a DIMENSIØN statement including all these variables may occupy a large part of the available memory space or storage locations, leaving inadequate space for the rest of the program, thereby preventing the machine to continue the execution due to overflow of memory. In such a case, the programmer may try to overcome the difficulty (of overflow of memory) by assigning the same storage locations to several variables which are not needed at the same time in the same program. Thus, instead of assigning unique storage locations for all the variables and arrays in a program, two or more of the variables or arrays of the same size can be made to share the same storage locations.

The general form of an EQUIVALENCE statement is:

EQUIVALENCE $(a_1,a_2,\ldots,a_n)$, $(b_1,b_2,\ldots,b_m)$,...

where $a_1,a_2,\ldots,a_n,b_1,b_2,\ldots,b_m$ are either subscripted or non-subscripted variables. This statement will cause the variables $a_1,a_2,\ldots,a_n$ to share one storage location and variables $b_1,b_2,\ldots,b_m$ to share another storage location. Each

pair of parenthesis in the EQUIVALENCE statement encloses two or more variable names that refer to the same storage location during the execution of the object program. Any number of variables may be listed in a single EQUIVALENCE statement. We give here some examples to explicitly illustrate the usage of the EQUIVALENCE statement:

Example.1: A programmer might realise after writing a long program that he has inadvertently (or, even wantonly!) named the same variable as X, X1 and RST. Instead of trying to correct his error, he can introduce the following statement:

EQUIVALENCE (X, X1, RST)

Example.2: If in a certain program variables NPL, IISC, MATS and IIT occur only once in different portions of the same program, then instead of allowing these to occupy four storage locations, we may assign all the four variables to one location by writing:

EQUIVALENCE (NPL, IISC, MATS, IIT)

Example.3: In a program, suppose that a set of coefficients $C_1, C_2, \ldots, C_{100}$ are read from cards and that once their values have entered the computation no further reference will be made to any but the first 20 of them. At a later stage, a set of intermediate results $R_1, R_2, \ldots, R_{60}$ may be generated, which in turn finally produce the results $S_1, S_2, \ldots, S_{30}$. By the time the first of these final results is obtained, if even the

coefficients $C_1, C_2, \ldots, C_{20}$ are no longer needed, then the DIMENSIϕN statement

DIMENSIϕN C(100), R(60), C(30)

may be supplemented by the EQUIVALENCE statement:

EQUIVALENCE (C(31), R(1)), (C(1),S(1))

which assigns (in the intermediate stage) to R_1 through R_{60} the same locations as to C_{31} through C_{90} and also assigns (in the final stage) to S_1 through S_{30} the same locations as C_1 through C_{30}. Note that in the EQUIVALENCE statement above (R(1), C(31)) and (S(1), C(1)) could have been used instead of (C(31),R(1)) and (C(1), S(1)). In this example, the locations assigned to C_{91} to C_{100} are not shared by any other variables. Thus, it is not necessary for the arrays made equivalent to have the same number of elements.

In writing an Equivalence statement, the programmer must note that even two-and three-dimensional arrays actually appear in storage as a one-dimensional sequence of core locations. We do know that arrays are stored in such a way that the first subscript varies most rapidly and that the last varies least rapidly. To determine in a two-or three-dimensional array which element is the n^{th}, a simple element successor or rule is employed. If A, B and C are maximum values of subscripts given by a DIMENSIϕN statement and a,b and c are values of the subscript expressions; then Table.4 below gives the needed information about conversion from multiple subscripts to a single subscript:

Table.4. Element Successor Rule

Dimensionality	Maximum subscript value(given in DIMENSIϕN statement)	Multiple subscript (in expression)	Corresponding single subscript value
1	(A)	(a)	a
2	(A,B)	(a,b)	a+A.(b-1)
3	(A,B,C)	(a,b,c)	a+A.(b-1)+A.B.(c-1)

Example.1:

For instance if the array X is declared in the DIMENSIϕN statement as X(3,12), then as per table above, A=3, and B=12. The 36 elements of array are arranged in a linear sequence. Where will X(2,9) occur in this one-dimensional sequence? The answer from the table (for a=2, b=9) is 2+3.(9-1)=26 Thus, if the element X(1,1) is in the location X(1), then the element X(2,9) is in the location X(26).

Example.2:

If A is a three-dimensional variable declared as:

DIMENSIϕN A(2,3,4)

and further if A(1,1,1) is in the location A(1), then the element A(1,1,2) is in the location A(7).

Example.3:

In an array dimensioned A(3,3,3) the fourth element of the array can be referred as A(1,2,1) or as A(4).

Example.4: The statements

```
DIMENSIϕN  A(3,3), B(2,2), C(2)
EQUIVALENCE  (A(5), B(2),C(1))
```

sets up an equivalence among the elements of each array as follows:

A(1,1), A(2,1), A(3,1), A(1,2), A(2,2), A(3,2), A(1,3), A(2,3), A(3,3)
B(1,1), B(1,2), B(2,1), B(2,2)
C(1), C(2)

The possibility of redefining a two-dimensional array as a one-dimensional array by means of an EQUIVALENCE statement may enable a programmer to substitute a one-dimensional address computation for the slower one in two-dimensions.

Example.5: To find a principal diagonal element of a square array, A(K,K), with, say, DIMENSION A(40,40), Fortran uses the element successor rule to pick the element from the corresponding cell of the string of cells assigned to the array A. The statements:

```
      DIMENSION  A(40,40), B(1600)
      EQUIVALENCE  (A(1), B(1))
```

enables us to speed up the evaluation of the sum S of the elements along the principal diagonal of the two-dimensional array A, by replacing the program segment:

```
      S = 0.0
      DO 55 K=1,40
55    S = S + A(K,K)
```

by the faster program segment

```
      S = 0.0
      DO 55 L = 1,1600,41
55    S = S + B(L)
```

The following are the Rules to be adhered to while making use of the EQUIVALENCE statement:

(i) EQUIVALENCE is a non-executable statement and must precede the first executable statement in the program or subprogram.

(ii) If DIMENSIϕN and EQUIVALENCE appear together, the order among these is immaterial.

(iii) None of the dummy arguments may appear in an EQUIVALENCE statement in a Subroutine subprogram.

(iv) The variables may be with or without subscripts.

Exercise.12:

Sketch the storage layout that would result from the following:

```
(a)    DIMENSIϕN  A(3), B(4)
       EQUIVALENCE (A(2), B(1))
(b)    DIMENSIϕN C(2,3), D(3,2)
       EQUIVALENCE (C(3), D(1))
```

CϕMMϕN statement:

We have already noted that main and subprograms have their own variable names. Variables (or arrays) that appear in the main program and a subprogram, or in two subprograms, can be made to share the same storage locations by use of the CϕMMϕN statement (provided they are of the same length). In other words, the CϕMMϕN statement assigns variables in different subprograms or in a main program and a subprogram to the same

storage locations, while an EQUIVALENCE statement assigns variables within a main program or within a subprogram to the same storage location.

There are four kinds of C¢MM¢N statements which are referred to as:

(a) simple C¢MM¢N

(b) dimensioned C¢MM¢N

(c) labelled C¢MM¢N

(d) numbered C¢MM¢N

(i) Simple C¢MM¢N statement:

General form:

C¢MM¢N a,b,c,...,n

where a,b,c,...,n are fixed and/or floating-point variables.

Example.1: If one program contains the statement

C¢MM¢N TAB

and a second program contains the statement

C¢MM¢N T1

then the compiler will assign the two variables to the same storage location.

Example.2: Suppose the main program contains the statement

C¢MM¢N X,Y,I

and a subprogram contains the statement

C¢MM¢N A,B,J

Then X and A are assigned to one storage location, Y and B are assigned to another storage location, while I and J are assigned

to a third storage location.

Example.3: Dummy variables can be used in a CΦMMΦN statement to establish shared locations for variables that would otherwise occupy different locations. Thus while the main program has:

CΦMMΦN X,Y,Z

the subprogram can be:

CΦMMΦN Q,R,S

where Q and R are dummy names that are not used in the subprogram, while variable S used in the subprogram is assigned the same location as variable Z.

(ii) Dimensional CΦMMΦN statement:

General form:

CΦMMΦN $a(k_1),b(k_2),c(k_3),\ldots,n(k_n)$

where a,b,c,...,n are array names and

$k_1,k_2,k_3,\ldots,k_n$ are each composed of 1,2 or 3 unsigned integer constants that specify the dimension of the array. This form of the CΦMMΦN statement performs not only the function of the simple CΦMMΦN statement but also performs the additional function of specifying the size of the arrays. A variable that is written with subscripts in a CΦMMΦN statement must not be mentioned in a DIMENSIΦN statement. But, if a variable is dimensioned in a DIMENSIΦN statement, it is permissible to use the variable name alone in the CΦMMΦN statement, without subscripting information.

Example.1: A main program contains the following statement:

COMMϕN A(10), B(5,5,5), C(5,5,5)

while the subprogram contains:

COMMϕN X(40), Y(10,10), Z(120)

This means that the main program contains 10 elements of A, 125 elements of B and 125 elements of C. The same 260 locations would also contain the 40 elements of X, the 100 elements of Y and the 120 elements of Z. The overlap caused by the six arrays involved would cause the compiler no difficulty - indeed the compiler would never really consider the situation!

Example.2: A single CϕMMϕN statement may contain variable names, array names and dimensioned array names. Thus, the following are valid statements.

DIMENSIϕN B(5,15)

CϕMMϕN A,B,C(9,9,9)

Or, equivalently: DIMENSIϕN B(5,15), C(9,9,9)

EQUIVALENCE A,B,C

(iii) Labelled CϕMMϕN statements:

The simple and dimensioned CϕMMϕN statements mentioned above set apart only one "CϕMMϕN block" of storage locations and the compiler never knows at a time more than one CϕMMϕN statement. We can establish as many distinct blocks of CϕMMϕN storage as we please by labelling the CϕMMϕN statement. The CϕMMϕN statements (simple and dimensioned) which we discussed above are called also as Blank CϕMMϕN statements.

General form:

C¢MM¢N/Ident 1/List 1/Ident 2/List2...

where 'Ident 1', 'Ident 2' are the identifiers of the C¢MM¢N blocks of storage locations being set apart for the list of variables 'List 1', 'List 2' etc. The identifier name should appear between two slashes and like a variable name must start with an alphameric character.

The same name should be used in a subprogram, if a list of variables belonging to the subprogram are to be assigned to the same C¢MM¢N block of storage locations.

Example.1: The C¢MM¢N declaration in a main program is:

C¢MM¢N/BLK/A(10,10), B(10,10), C(10,10)

The number of locations for the C¢MM¢N block named BLK is 300. In other words the C¢MM¢N block is 300 "words" long.

If a subroutine, say, a matrix multiplication routine, can use the same common block to multiply two 10 x 10 matrices A and B and put the result in C, then the Subroutine including the C¢MM¢N declaration is as follows:

```
      SUBR¢UTINE MAT
      C¢MM¢N/BLK/X(10,10), Y(10,10), Z(10,10)
      D¢ 5 I = 1,10
      D¢ 5 J = 1,10
      Z(I,J) = 0.0
      D¢ 5 K = 1,10
    5 Z(I,J) = Z(I,J) + X(I,K)*Y(K,J)
      RETURN
      END
```

Example.2: If a single CϕMMϕN statement includes labelled and blank CϕMMϕN statements, the blank CϕMMϕN portion may either be written first without a name, as we have done so far, or the name may be omitted between slashes:

i.e. CϕMMϕN X,Y,Z/BLK/P,Q

and CϕMMϕN // X,Y,Z/BLK/P,Q

mean the same to the Compiler, viz. X,Y,Z belong to blank CϕMMϕN and P,Q belong to the CϕMMϕN block BLK.

Example.3: Suppose the statement in the main program is:

CϕMMϕN R,ST/B1/U,V/B2/F(20),G(2,5)

and the statement in the subprogram is:

CϕMMϕN R,ST/B1/U,V/B2/X(10),Y(10,2)

Here blank CϕMMϕN would assign A,B and C, in that order, in the main program to three locations and assign R,S and T in the subprogram to the same storage locations. The CϕMMϕN block labelled B1 would establish D and U in the same location and E and V to another location in the same block B1. The Common block labelled B2 contains in the main program 20 elements of F and 10 elements of G while in the subprogram the same 30 locations would contain the 10 elements of X and 20 elements of Y. As stated earlier, the overlap between the four arrays involved would cause the compiler no difficulty.

(iv) Numbered CϕMMϕN statement:

The name of the labelled CϕMMϕN statement can be a number. For a numbered CϕMMϕN statement, the first letter as

well as the other characters, if any, of the name must be numeric. Leading zeroes in numeric identifiers are ignored. Zero by itself is an acceptable number as a CMMN block identifier.

Example: CMMN/140/A,B/1226/C(10),D(10,10)

Rules to be followed while using CMMN statement:

1. CMMN is a non-executable statement and must precede the first executable statement in the program. Any number of CMMN statements may appear in a program section.
2. If DIMENSI$N, EQUIVALENCE and/or C$MM$N appear together, their order is immaterial.
3. Labelled CMMN block identifiers are used only for block identification within the compiler; they may be used elsewhere in the program. However, one may not choose to do so.
4. An identifier in one CMMN block may not appear in another CMMN block. If it does the identifier will be doubly identified.
5. The order of arrays in a CMMN block are determined by the CMMN statement. All items in CMMN are stored in descending storage locations.
6. Dummy arguments for SUBR$UTINE or FUNCTI$N statements cannot appear in CMMN statements.
7. A variable that is written with subscripting information in a CMMN statement must not be mentioned in a DIMENSI$N statement.

CϕMMϕN with EQUIVALENCE statement:

When an array is named in an EQUIVALENCE statement and in a CϕMMϕN statement, the EQUIVALENCE is established in the same general way as described earlier. However EQUIVALENCE statement may increase the size of the CϕMMϕN block of storage locations and thus change the correspondences between the CϕMMϕN block described and some other CϕMMϕN block in another program.

Example.1: Consider a program containing the following statements:

```
DIMENSI$\phi$N A(4), B(4)
C$\phi$MM$\phi$N A,C
EQUIVALENCE (A(3),B(1))
```

Without the EQUIVALENCE statement, the CϕMMϕN block would contain five storage locations in the sequence.

A(1), A(2), A(3), A(4),C

With the EQUIVALENCE statement, the array is brought into CϕMMϕN, so to speak, and requires the following sequence of storage locations:

A(1), A(2), A(3), A(4), C

(1), B(1), B(2), B(3), B(4)

so that the CϕMMϕN is now six storage locations long. CϕMMϕN may be lengthened this way.

Since arrays are stored in descending order of storage locations, a variable may not be made Equivalent to an element of an array in such a manner as to cause the array to extend beyond

the beginning of the CMMN area. In other words, the beginning of a CMMN block must never be shifted by an EQUIVALENCE statement. In example.1 considered above, the following Equivalence statement is illegal:

```
EQUIVALENCE (A(1),B(2))
```

For, in such a case, the storage assignment in CMMN would be:

```
A(1), A(2), A(3), A(4),C
B(1), B(2), B(3), B(4), B(5)
```

Since B is not mentioned in the CMMN statement, but is brought into CMMN by the Equivalence statement, the first element of B precedes the start of the CMMN block which is not permitted.

Example.2: Within an EQUIVALENCE list, there should be no more than one variable which has been previously placed in an EQUIVALENCE or CMMN statement. If this is not satisfied there will be a contradiction.

Suppose we had the combination of statements:

```
C$MM$N  A,B,C,D
EQUIVALENCE (A,B)
```

Then, the CMMN assigns the four variables A,B,C,D in separate locations in a special area of storage, while the EQUIVALENCE states that A and B are to be assigned to the same location. Hence, the net effect of the above two statements is a contradiction.

For the reason stated above, the following sequence of statements are invalid:

```
C$MM$N  D
EQUIVALENCE (A,B,C),(X,Y,Z),(A,Z),(D,X,P)
```

while the following sequence is valid:

CϕMMϕN D

EQUIVALENCE (D,X,P), (A,B,C,X), (X,Y,Z)

DATA statement:

The programmer may assign constant values to variables in the source program by using the DATA statement either by itself or with a DIMENSION statement. The DATA statement assigns values to statements at the time of compilation, not at the time of execution of the object program.

General Form:

(i) DATA List1/$d_1,d_2,\ldots,d_n$/, List 2/$d_1,d_2,\ldots,d_m$/,....

where 'List 1', 'List 2',.. contains the names of variables to receive the values and d's are the values. If d_3 is repeated, say, k times, then in the list this is indicated as $k * d_3$ where k is an integer.

Example.1: DATA A,B,C/14.7, 62.1, 1.5E-20/

This statement would assign the value 14.7 to A, 62.1 to B and 1.5×10^{-20} to C.

Example.2: The following statements have the same effect:

DATA A/67.87/, B/54.72/, C/5.0/

and DATA A,B,C/67.87, 54.72, 5.0/

The choice is a matter of personal preference.

Example.3: The following statement assigns the value 21.7 to all the six variables r,s,t,u,v and w:

DATA R,S,T,U,V,W/6*21.7/

In some installations (for e.g. CDC 3600-160A at T.I.F.R., Bombay) the following general form of the DATA statement is in vogue:

DATA (V_1 = List 1), (V_2 = List 2),...

where V_1, V_2,.. represent simple variable name, array name, or a variable with integer constant or integer variable subscripts and List 1, List 2,... represent constants only and these have the general form:

$$a_1,a_2,\ldots,k\ (b_1,b_2,\ldots),\ C_1,C_2,\ldots$$

where $a_1,a_2,\ldots,b_1,b_2,\ldots,C_1,C_2,\ldots$ are simple fixed or floating-point constants and k is a constant repetition factor that causes the parenthetical list following it to be repeated k times. k is always an integer, if it is used.

Rules to be followed while using the DATA statement:

1. DATA is a non-executable statement and must precede the first executable statement in any program or subprogram in which it appears.
2. When DATA appears with DIMENSIØN, EQUIVALENCE or CØMMØN statements, the order in which these statements appear, before the first executable statement, is immaterial.
3. DØ-loop-implying notation is permissible with the restriction that the third indexing parameter, m_3, cannot appear.

Example:

```
      DATA (GIB(I), I=1,10)/1.0,2.0,3.0,7*(4.32)/
or    DATA ((GIB(I), I=1,10) = 1.0, 2.0,3.0,7(4.32))
```

4. There must be a one-to-one correspondence between the list of variables and their values specified in a DATA statement.
5. A DATA statement may contain Hollerith text:

Example:1 DATA DØT, X, BLNAK/1H., 1HX, 1H /

or DATA (DØT = 1H.), (X = 1HX), (BLANK=1H) -

Example.2:

```
      DIMENSIØN MESGE(3)
    DATA (MESGE(I), I=1,3)/3HWHØ, 2HIS, 3HSHE/
or  DATA (MESGE = 3HWHØ, 2HIS, 3HSHE)
```

FØRGØ and FØR-TØ-GØ:

So far we have been discussing the various features of the programming language called FØRTRAN-II. In this section we would like to discuss two simplified versions of FØRTRAN - II, usually taught to beginners in FØRTRAN programming.

FØRGØ was written primarily to serve as an educational programming system to teach undergraduate engineering students the principles of programming and incidentally to 'debug' research programs before they are run on a computer larger than IBM1620. The system checks for almost all possible errors and all errors are referred back to the only language the students are expected to understand - FØRTRAN. The system is strictly card oriented, but a control digit will permit the typewriter to be used for demonstration purposes.

FØRGØ and FØR-TØ-GØ are very closely related programming languages. The major difference between them is that both the

compiler and Subroutines remain in memory at all times with FΦRGΦ, thus leaving less room (i.e. storage locations) for the compiled program, while the compiler is overwritten by the Subroutines in FΦR-TΦ-GΦ, thus permitting much larger room for compiled programs. Quantitatively this means the following:

The memory for IBM1620 is 40K(each word takes 10 core locations). The compiler (including subroutines) takes up 35K (or 35,000 core locations). Hence only 5K is available for the source program. In FΦR-TΦ-GΦ the Compiler is in two parts, each of 20K. Part A translates the source language into machine language. Then Part B, which contains library functions, replaces the core occupied earlier by part A. Hence 20K is available for the source program.

The following points are noteworthy for FΦRGΦ and FΦR-TΦ-GΦ in comparison with FΦRTRAN-II.

1. The first card of a program must be a card called Control Card. It has a C punched in columns 1 and 4 and any other information in columns 7 through 80. Each program has one and only one control card. (Note that a Comment Card has ' C' punched in Column 1 only and that there may be as many comment cards as desired).

2. Statement numbers may be punched in columns 2 to 5 of the source program cards.

3. FΦRMAT statements are optional. In the absence of a FΦRMAT statement for input variables, the values of the variables punched on data cards must be separated by commas.

4. The full Fortran expressions are allowed, except that in raising a floating-point quantity to a fixed-point power. The power may be either a single variable or a constant, but may not involve a fixed-point operation, i.e. (A+B)**I is permissible but not A**(I+J).

5. In FØRGØ operation, no continuation cards are allowed. Hence, column 6, must be left blank in all FØRGØ source statements.

In FØR-TØ-GØ one continuation card is allowed and the first card of the pair must have a blank in column 6 while the second card of the pair must have 1 punched in column 6.

6. Subscripted variables may be either one-or-two-dimensional and a subscript may be of the form I*J ± K

7. The following library functions and spellings are available:

SIN(X) , SINF(X)	:	Sine of x (x in radians)
CØS(X) , CØSF(X)	:	Cosine of x(x in radians)
ATAN(X) , ATANF(X) ATN(X) , ATNF(X)	:	Arctangent of x(angle in radians)
EXP(X) , EXPF(X)	:	Exponential of x
LØG(X) , LØGF(X)	:	Natural logarithm of x.
SQRT(X) , SQRTF(X) SQR(X) , SQRF(X)	:	Square root of x
ABS(X) , ABSF(X)	:	Absolute value of x.

Note that the last character is not required to be F.

8. The following statements are not allowed in both FRG and F$R-T$-G$:

IF(SENSE SWITCH i) n_1,n_2

PAUSE
PRINT
FUNCTI$N
SUBR$UTINE
CALL
RETURN
EQUIVALENCE
CMMN
DATA

F$RTRAN-I: In F$RTRAN-I the following restrictions exist:

1. The library function ABSF(X) is not available.
2. Multiple Format specifications, such as 5I8, is not allowed.
3. Maximum Hollerith specification is 49H.
4. Maximum X specification is 49X.
5. The following Fortran statements are not allowed:

SUBR$UTINE
FUNCTI$N
CALL
RETURN
EQUIVALENCE
CMMN
DATA.

References

"Basic Programming Concepts and the IBM 1620 Computer",
by D.N.Leeson and D.L.Dimitry, Holt, Rinehart and Winston, Inc.,(1962).

"Numerical Methods and Computers"
by S.S.Kuo, Addison-Wesley (1965).

"Introduction to Basic Fortran Programming and Numerical Methods",
by W.Prager, Blaisdell Publishing Co., (1965).

"A guide to FORTRAN IV Programming"
by D.D.McCracken, John Wiley and Sons, Inc., (1966)

"3600 Computer System Fortran",
Control Data Corporation (1964), preliminary reference manual.

"3600 FØRTRAN" (User's Manual)
by K.S.Kane and C.N.Kum, T.I.F.R. Report.

APPENDIX.1

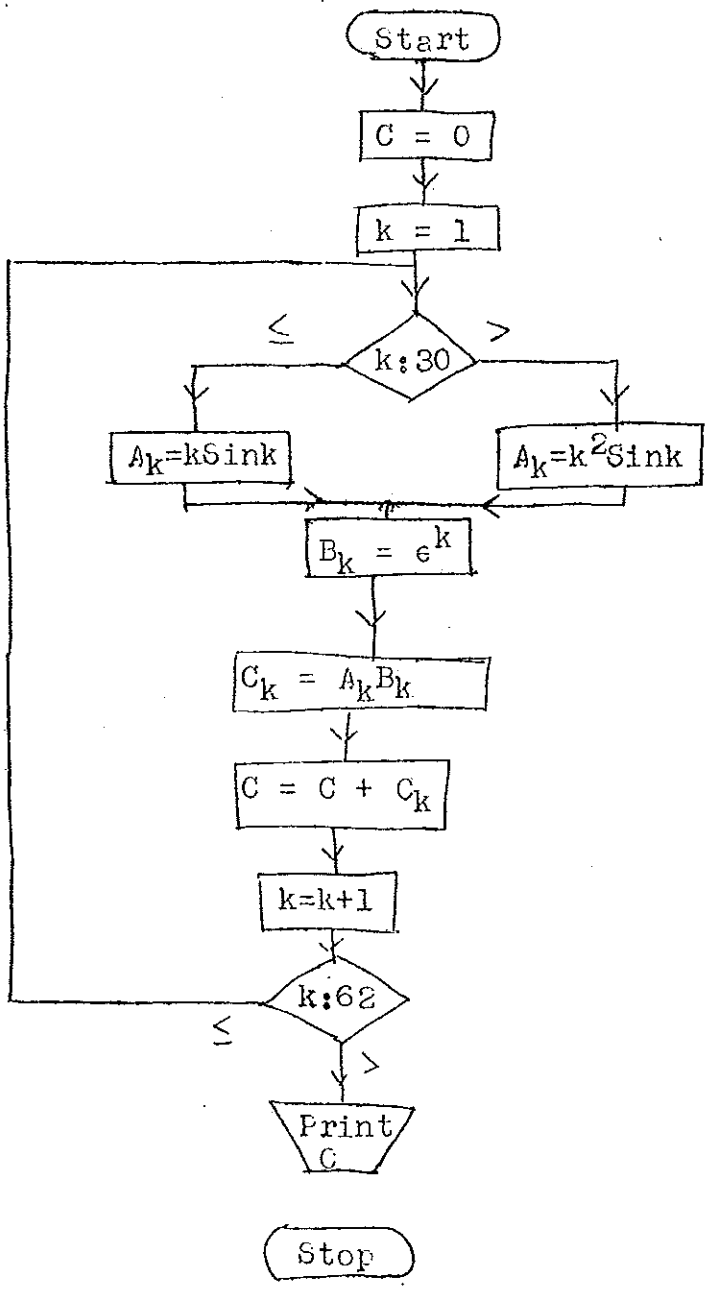
Library Functions available at IBM 1130 installation:

Name	Function performed	No.of arguments
SIN	Trigonometric Sine	1
CΦS	Trigonometric Cosine	1
ATAN	Arctangent	1
ATANH	Hyperbolic tangent	1
ALΦG	Natural logarithm	1
EXP	Argument power of e(i.e..e^X)	1
SQRT	Square root	1
ABS	Absolute value	1
IABS	Integer Absolute value	1
FLΦAT	Convert fixed point argument to floating point argument	1
IFIX	Convert floating point argument to fixed point argument	1
SIGN	Transfer of sign (Sign of Argument 2 times Argument 1)	2
ISIGN	Integer transfer of sign (Sign of Arg.2 times Argument1)	2

APPENDIX 2

WORKED OUT EXAMPLES

Ex:1



Ex:2

X**Y**2 ; B**(K+2) ; X**(A+B) ;

A + B/(C + E/F); P* Q/(R*S); A/B + N;

A*X**2 + B*X + C ; A**(X + 2.0);

(B**2 - 4.0*A*C)**0.5.

Ex:3 In Program 1, the READ-FORMAT combination should be changed to:

```
    1  READ 2, B, C, A
    2  FORMAT (F6.2, 3XF5.1, 5XF5.2)
```

Ex:4:

```
      READ 2, X, N
    2 FORMAT (F10.4, I5)
      N1 = N+4
      R = (13.6 - X)**N1
      PRINT 6, X,N,R
    6 FORMAT (3H X=F10.4,2X2HN=I5,2X2HR=E20.8)
      STOP
      END
```

Ex.5:

```
      READ 10, A,B,C,X
   10 FORMAT(4F6.3)
      A1 = 1.0 - X/A
      TERM1 = 6.0 * A1 * X * *2
      TERM2 = (B * A1)**2
      TERM3 = C **2 *A1 **0.5
      R = B*C*(TERM1 + TERM2 + TERM3)/12.0
      PRINT 15, A,B,C, X,R
   15 FORMAT (3H A=F6.3,2X2HB=F6.3,2X2HC=F6.3,2X2HX=F6.3,
     1 2X2HR=E20.8)
      STOP
      END
```

Ex.6: The flow chart for this exercise is similar to that drawn for Ex.1.

```
C     START OF PROGRAM FOR EXERCISE 6
   10 READ 12, Y
   12 FORMAT (F10.5)
      X = 0.0
      K = 0
   15 AK = K
      IF(K-250) 16,16,18
```

```
   16 |CK = AK**0.5
      |GO TO 19
   18 |CK = 3.0 *AK **2.0
   19 |X = X + CK* Y**K
      |K = K+1
      |IF(K-500)15,15,22
   22 |PRINT 23, Y, X
   23 |FORMAT (3H Y=F10.5,2X,2HX=E15.7)
      |GO TO 10
   25 |STOP
      |END
```

Ex.7:

```
    C |PROGRAM FOR EXERCISE 7
      |N = 25
      |I = 1
      |P = 1.0
      |Q = 1.0
      |S = 0.0
    5 |S = S + P/Q
      |P = -2.0 *P
      |Q = (Q + 2.0)**2
      |I = I+1
      |IF(I-N)5,5,10
   10 |PRINT 11, I, SUM
   11 |FORMAT (2X2HN=I5,2X2HS=E15.8)
      |N = N+25
      |IF(N-100)5,5,13
   13 |STOP
      |END
```

Ex.8:

```
    C |PROGRAM FOR EXERCISE 8. STERLING'S FORMULA.
      |DIMENSION Y(50)
      |READ 2, U, I
    2 |FORMAT (F10.4, I5)
      |  = A(=,Y(I)
      |    B = U *(Y(I+1) - Y(I-1))/2.0
      |    C = (U**2)*(Y(I+1) - 2.0 *Y(I)+Y(I-1))/2.0
```

```
      S = A+B+C
      PRINT 10, U,I,S
   10 FORMAT(3H U=F10.4,2HI=I5,2HS=E14.7)
      STOP
      END
```

Ex.9:

```
C     PROGRAM FOR EXERCISE 9.
      DIMENSION A(7), B(7)
      READ 2, (A(I), I=1,7)
    2 FORMAT(7F8.4)
      READ 2, (B(I), I=1,7)
      SUM = 0.0
      DO 5 I = 1,7
    5 SUM = SUM+A(I) * B(I)
      ANORM = SUM**(2.0/3.0)
      PRINT 8, ANORM
    8 FORMAT(7H ANORM=E20.8)
      STOP
      END
```

Ex.10:

```
C     PROGRAM FOR EXERCISE 10.
      AF(X) = X**2 + SQRT(1.0+2.0 * X + 3.0 *X **2)
      READ 2, Y,Z
    2 FORMAT(2F10.4)
      ALPHA = (6.9+Y)/AF(Y)
      BETA = (2.1 * Z + Z**4)/AF(Z)
      GAMA = SIN(Y)/AF(Y**2)
      DELTA = 1.0/AF(SIN(Y))
      PRINT 10, Y,Z,ALPHA, BETA, GAMA, DELTA
   10 FORMAT(3H Y=F10.4,2HZ=F10.4//6HALPHA=E13.6,
     1 5HBETA=E13.6,5HGAMA=E13.6,6HDELTA=E13.6)
      STOP
      END
```

Ex.11:

```
      FUNCTION Y(X)
      IF(X)2,4,6
    2 Y = 1.0 + SQRT(1.0 + X**2)
      RETURN
    4 Y = 0.0
      RETURN
    6 Y = 1.0 - SQRT (1.0 - X**2)
      RETURN
      END
```

```
C     MAIN PROGRAM FOR EXERCISE 11.
      F = 2.0*Y(A+Z)
      G = (Y(X(K)) + Y(X(K+1)))/2.0
      PI = 3.14159265
      H = Y(COS(2.0 * PI * X)) + SQRT(1.0 + Y(2.0* PI* X))
```

Ex.12:

(a) A(1), A(2), A(3)

B(1), B(2), B(3), B(4)

(b) C(1,1),C(1,2),C(1,3),C(2,1),C(2,2),C(2,3)
D(1,1),D(1,2),D(2,1),D(2,2),D(3,1),D(3,2)