Programmer's Primer for

Fortran

AUTOMATIC CODING SYSTEM

FOR THE IBM 704

Programmer's Primer for FORTRAN Automatic Coding System for the IBM 704

TABLE OF CONTENTS

Page

GENERAL INTRODUCTION 1

SECTION I - PUNCHED CARD INPUT, ARITHMETIC OPERATIONS, STANDARD FUNCTIONS, PRINTED OUTPUT

Introduction 5
Arithmetic Statements 6
READ and PRINT Statements 8
IF, Unconditional GO TO, and STOP Statements 14
Additional Examples 16
Check List 19

SECTION II - DEFINITION OF FUNCTIONS, MANIPULATION OF SINGLE-SUBSCRIPTED VARIABLES, MAGNETIC TAPE INPUT AND OUTPUT

Introduction 23
Integer Constants and Variables 26
DIMENSION Statements 28
DO Statements 28
Function Statements 30
The Meaning of a List 31
FORMAT Statements 32
Magnetic Tape Input and Output 35
READ INPUT TAPE and WRITE OUTPUT TAPE Statements 35
General Information about the Use of Tapes 36
Additional Examples 37
Check List 41

SECTION III - MANIPULATION OF TWO- AND THREE-DIMENSIONAL ARRAYS

Introduction 43
Subscripts for Two- and Three-Dimensional Arrays 45
DO Nests 45
Lists for Two- and Three-Dimensional Arrays 47
Assigned GO TO Statements 48
Computed GO TO Statements 49
FORMAT Statements 51
Scale Factors 51
Hollerith Fields 52
Multiple-line Format 52
Example Problem and Program 53
Debugging 58
Storage 60
Master Check List 61
Summary of FORTRAN Statements 63

GENERAL INTRODUCTION

Every type of electronic computer is designed to respond to a special code, called a "machine language," which differs for different types of computers. A program, or set of instructions, telling a computer what steps to perform to solve a problem must ultimately be given to the computer in its own language. However, the FORTRAN System makes it unnecessary for the 704 programmer to learn the 704 machine language.

The FORTRAN System has been developed to enable the programmer to state in a relatively simple language, resembling familiar usage, the steps of a procedure to be carried out by the 704 computer, and to obtain automatically from the 704 an efficient machine language program for this procedure. The FORTRAN System has two parts: the FORTRAN language, and the FORTRAN translator or executive routine.

The FORTRAN language consists of 32 types of statements, which may be grouped into four classifications: arithmetic statements, control statements, input/output statements, and specification statements. The FORTRAN programmer uses this language to state the steps ultimately to be carried out by the 704. The FORTRAN translator is a large set of machine language instructions which causes the 704 to translate a FORTRAN program into an efficient (or "optimized") machine language program. The 704 does not respond directly to a FORTRAN program, but to the machine language program produced by means of the FORTRAN translator. The name FORTRAN comes from "FORmula TRANslation" and was chosen because many of the statements which this system translates look like algebraic formulas.

The purpose of the FORTRAN Primer is to introduce the reader to the FORTRAN language, which has been designed as a concise, convenient means of stating the steps to be carried out by the 704 in the solution of many types of problems, such as frequently occur in engineering, physics, and other scientific and technical fields. No prerequisite knowledge of computing technology or techniques on the part of the reader is assumed.

In order to clarify what the 704 can do and what it cannot do, consider, for example, the problem of finding the roots of a quadratic equation. The 704 cannot be given an equation of the form

$$3x^2 + 1.7x - 31.92 = 0$$

and be directed to find its roots. The 704 can, however, be directed to compute the value of

$$(-1.7 + \sqrt{(1.7)^2 - 4(3)(-31.92)})/2(3)$$

which gives one of the roots of the preceding equation. That is, the 704 must be told how to find the answer. It will do the work.

Once the 704 has been told how to solve a problem, it can take in data from punched cards at a rate exceeding 1000 numbers per minute, perform arithmetic steps at an approximate rate of 10,000 per second, and print results at a rate of about 750 per minute. Thus the 704 can do in minutes calculations which would require weeks or months to do manually.

Virtually any numerical procedure may be expressed in the FORTRAN language. The FORTRAN System is intended to reduce substantially the time required to produce an efficient machine language program for the numerical solution of a problem, and to relieve the programmer of a considerable amount of manual "clerical" work, minimizing the possibility of human error by relegating the mechanics of coding and optimization to the 704.

In this primer, as an aid to efficient study, the FORTRAN language is approached cumulatively through three stages, Section I, Section II, and Section III. The division into three sections is convenient for the description of successively more complex problem-solving procedures.

By using only the types of statements presented in Section I, it is possible to direct the 704 to take individual numbers from a card reader; combine them according to formulas involving arithmetic operations and standard functions such as sine, square root, log, etc.; make tests and follow different directions depending on the outcome of the tests; and finally print the results.

Section II presents additional types of statements which provide for the definition and use of functions peculiar to the problem to be solved; the iterative manipulation of subscripted variables (the elements of vectors or lists of numbers); the use of magnetic tape for input and output information; and greater flexibility in the format of input and output information. When magnetic tape is used for input and output information, the 704 can read or write more than 900 numbers per second, a much greater rate than is possible when the 704 reads cards and prints results directly. Since all information to be used or processed by the 704, other than magnetic tape output from a previous computer operation, must initially be recorded on punched cards, and since it is often desirable to maintain permanent records on punched cards or in printed form, the use of magnetic tape for input and output requires a means of transferring information from cards to tapes, tapes to cards, and tapes to printed form; this transfer may be effected by means of separate peripheral equipment.

Section III adds to these facilities the ability to handle matrices and three-dimensional arrays of numbers, to perform more complex iterative procedures, and to direct the flow of control within a program more flexibly.

The three sections of this primer do not include all the facilities offered by the complete FORTRAN language. Full information about the FORTRAN language is given in the manual Programmer's Reference Manual for FORTRAN, Form 32-7026, which can be obtained from IBM Stationery Stores, Endicott, N.Y., in the usual manner.

This primer describes the writing of FORTRAN programs for the 704; it does not explain the mechanics of tape loading, card feeding, and other related operations needed to get the 704 working on a problem, since it has been assumed that the 704 will be operated by a computing center. At the computing center, the FORTRAN program statements are first transcribed onto punched cards exactly as they have been written, each letter, digit, or punctuation mark resulting in one or more holes in a single column of a card. These cards are then placed in the card reader for the 704. By means of the FORTRAN translator or executive routine, the 704 is directed to start reading the FORTRAN statements from the cards; to translate the FORTRAN statements into machine language, one FORTRAN statement usually resulting in a sequence of several machine language instructions; and to record the machine language instructions as output on punched cards. The new set of cards contains a machine language program, which can be entered into the 704 any number of times, each time causing the 704 to carry out the procedure specified by the original set of FORTRAN statements.

Before the presentation of FORTRAN statements, a short description of the 704 itself may be of interest. The 704 consists of a control unit; an arithmetic and logical unit; a magnetic core storage unit capable of storing 4,096, 8,192, or 32,768 numbers; magnetic tape units and magnetic drum units for holding information exceeding the instantaneous storage capacity of the magnetic core storage unit; a card reader; a card punch; and a printer. The control unit directs the flow of information between the other units in accordance with the machine language instructions currently in the magnetic core storage unit.

Most calculations performed by the 704 are carried out in "floating point" form. Numbers are represented in the machine in this form, and the results produced by the arithmetic unit are usually in this form.

For example: calculation of the product

$$5 \times 0.0037$$

would be carried out by the 704 in a form analogous to

$$(.5000\ 0000 \times 10^{1}) \times (.3700\ 0000 \times 10^{-2}) = (.1850\ 0000 \times 10^{-1}).$$

This would be the case even though the numbers were entered as 5.0 and 0.0037 and the result were printed as 0.0185. All floating point numbers in the 704 are carried to about 8 significant decimal digits. Numbers outside the range 10^{-38} to 10^{38} (other than zero) cannot normally be accommodated.

Since FORTRAN language statements have to be translated by a machine, the 704, before a problem is ready to be run, FORTRAN statements must be written in exactly the proper form. The machine has no ability to understand what was meant; it can only translate what was written. Therefore, the omission of a single decimal point or operation symbol will make a FORTRAN statement incapable of being translated correctly. For this reason, the rules for writing FORTRAN statements must be carefully followed. In devising the FORTRAN System, considerable effort was devoted to making these rules consistent and having them conform to familiar usage wherever possible. A number of examples have been provided to illustrate these rules without having to include specific statements of them in the text. At the end of each section, a check list of things not to write is given, which should answer any remaining questions. Some of the rules in the check lists are not explicitly stated elsewhere.

SECTION I - PUNCHED CARD INPUT, ARITHMETIC OPERATIONS, STANDARD FUNCTIONS, PRINTED OUTPUT

Introduction

Consider the quadratic equation example previously presented

$$3x^2 + 1.7x - 31.92 = 0$$

The algebraic representation for one of the two roots of the equation could be written

$$\text{root} = \frac{-B + \sqrt{B^2 - 4AC}}{2A}$$

where $A = +3$
$B = +1.7$
$C = -31.92$

The complete FORTRAN program which describes this calculation and provides for printing the result may be written in six separate statements as follows:

C ← FOR COMMENT / STATEMENT NUMBER (1–5)	CONTINUATION (6)	FORTRAN STATEMENT (7–72)
		A = 3.
		B = 1.7
		C = -31.92
		ROOT = (-B + SQRTF(B**2. - 4.*A*C)) / (2.*A)
		PRINT 1, ROOT
		STOP

The first statement means: "Assign the value 3. to the variable A." The next two statements have a similar meaning. The fourth statement means: "Evaluate the expression on the right side and assign the result to the variable ROOT." The fifth statement prints the computed value of ROOT (see pages 8 ff.). The last statement instructs the computer to stop.

Notice the sequential nature of the program. The computer executes instructions in the same order as the order of the statements. For example, if the fourth statement were to be moved up and made the first statement, then the computer would evaluate ROOT before obtaining the desired values of A, B, and C. ROOT would therefore be evaluated using some arbitrary, unknown values for these variables.

The above example was written to illustrate the use of variables.

However, the same result could be obtained by writing

C = FOR COMMENT / STATEMENT NUMBER (1–5)	CONTINUATION (6)	FORTRAN STATEMENT (7–72)
		ROOT = (-1.7 + SQRTF (1.7**2. - 4.*3.* (-31.92))) / (2.*3.)
		PRINT 1, ROOT
		STOP

in which the actual numerical values appear in the statement describing the evaluation of ROOT.

The program on page 5 illustrates the use of three types of FORTRAN statements. In all, six types of statements will be presented in this section:

1. Arithmetic statements (e.g., first four statements in the program).
2. READ statements.
3. PRINT statements (e.g., fifth statement in the program).
4. IF statements.
5. Unconditional GO TO statements.
6. STOP statements (e.g., last statement in the program).

Arithmetic Statements

The first four statements in the above example are called arithmetic statements. An arithmetic statement looks like a simple statement of equality. The discussion of arithmetic statements in this section will be restricted to those in which the left side of the equality is a simple variable. In Section II, additional facilities will be presented for handling arithmetic statements in which the left side is a function of one or more variables. The right side of all arithmetic statements is an expression which may involve parentheses, operation symbols, constants, variables, and functions, combined in accordance with a set of rules much like that of ordinary algebra.

The fourth statement in the example illustrates the use of the five basic operations in the FORTRAN language. The symbols + and - are used in the usual way for addition and subtraction. The symbol * is used for multiplication, and the symbol / is used for division. The fifth basic operation, exponentiation, is represented by the symbol **. A**B is used to represent A to the exponent B (i.e., A^B).

If only the types of statements presented in this section are used, all calculations will be carried out by the 704 in floating point arithmetic, and the programmer must so instruct the 704 by writing all constants with a decimal point. All numbers (and only numbers) are considered constants with the exception of statement numbers. The FORTRAN arithmetic expression

A**B*C + D**E/F - G

will be interpreted to mean

$$A^{B}C + \frac{D^{E}}{F} - G$$

That is, if parentheses are not used to specify the order of operations, the order is assumed to be:

1. exponentiation
2. multiplication and division
3. addition and subtraction

Parentheses are used in the usual way to specify order. For example

$$(A(B + C))^{D}$$

is written in FORTRAN as

```
(A*(B + C)) **D
```

There are just three exceptions to the ordinary rules of mathematical notation. These are:

1. In ordinary notation AB means A·B or A times B. However, AB never means A*B in FORTRAN. The multiplication symbol cannot be omitted.
2. In ordinary usage, expressions like A/B·C and A/B/C are considered ambiguous. However, such expressions are allowed in FORTRAN and are interpreted as follows:

   A/B*C means (A/B)*C
   A*B/C means (A*B)/C
   A/B/C means (A/B)/C

   Thus, for example, A/B/C*D*E/F means ((((A/B)/C)*D)*E)/F That is, the order of operations is simply taken from left to right, in the same way that

   $$A + B - C + D - E$$

   means

   $$(((A + B) - C) + D) - E$$

3. The expression $A^{B^{C}}$ is often considered meaningful. However, the corresponding expression using FORTRAN notation, 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.

Besides the ability to indicate constants (like 3.57 and 2.), simple variables (like A and ROOT), and operations (like - and *), it is also possible to use functions. In the example on page 5, SQRTF( ) indicates the square root of the expression in parentheses.

Since the number of possible functions is very large, each 704 computing center will have its own list of available functions, with information about their use. Functions given in this list must be referred to exactly as indicated.

Some functions which might appear in a typical list are:

FORTRAN Symbol	Function
ABSF(X)	$\|X\|$ (absolute value of X)
SQRTF(X)	$\sqrt{X}$
SINF(X)	sin X
ARCTANF(X)	arctan X
EXPF(X)	e^X
LOGEF(X)	$\log_e X$
LOG10F(X)	$\log_{10} X$
INTF(X)	integral part of X
MAXF(X, Y)	maximum of X and Y

Notice, as in the last example, that a function may have more than one argument; as in general mathematical usage, multiple arguments are separated by commas. Section II will present facilities for defining functions peculiar to the problem at hand and not available from the computing center list.

READ and PRINT Statements

As stated in the General Introduction, this section presents FORTRAN statements which can direct the 704 to take numbers from a card reader and, after carrying out the desired calculations, print the results. Consider again the example of finding a root of a quadratic equation.

In many cases it will be desired to find ROOT for a number of sets of values of A, B, and C. To do this, the 704 would have to be directed to read a card in which values for A, B, and C have been punched, compute the value of ROOT, print ROOT (along with A, B, and C), read another card with different values for A, B, and C, compute and print the corresponding value of ROOT and so on. In this case, the FORTRAN program could be written:

C ← FOR COMMENT / STATEMENT NUMBER (1–5)	CONTINUATION (6)	FORTRAN STATEMENT (7–72)
10		READ 1, A, B, C
		ROOT = (-B + SQRTF (B**2. - 4.*A*C)) / (2.*A)
		PRINT 1, A, B, C, ROOT
		GO TO 10

The first statement (which has been given the number 10 for reference purposes) causes the 704 to read the first card from the deck in the card reader. Three numbers should be on this card, represented by three sets of punches. The value of the first number is named A; the value of the second number, B; and the value of the third number, C.

The 704 then proceeds to compute ROOT as before, after which it prints (on one line across the page) the values named A, B, C; and ROOT in that order. Upon reaching the last statement, the 704 is directed: "Go to the statement numbered 10 and do what it says." Thus the 704 reads the next card with three new values for A, B, and C, then computes ROOT, prints the current values of A, B, and C and the computed value of ROOT, and again returns to statement 10. This process continues as long as there are cards in the hopper of the card reader. When the cards are exhausted, the 704 stops upon its return to statement 10.

FORTRAN provides facilities for specifying the format of input data and of printed output in a great variety of ways. FORMAT statements are used to specify the desired arrangement for input cards and for printing. Two particular input/output formats are presented in this section. The general description of FORMAT statements appears in Section II, beginning on page 32.

One or both of the two FORMAT statements which specify the input/output formats presented in this section will be added to the completed program by the computing center, provided the computing center has been requested to do so; these FORMAT statements are always numbered 1 and 2. To use these two statements one must know what arrangement of data each specifies. The input data must be prepared on punched cards in conformity to the data arrangement specified by one of the FORMAT statements used; similarly, the output data will be printed in conformity to one of these specified arrangements. If the input data conforms to FORMAT statement 1 or 2, the READ statement is given with a 1 or 2 following, respectively; similarly, either PRINT 1 or PRINT 2 is used to print results.

If FORMAT 1 is used, statement number 1 must be reserved for it; if FORMAT 2 is used, statement number 2 must be reserved for it. The statement number reserved for a FORMAT statement must not be used as the number of any other FORTRAN statement.

READ and PRINT statements are similar to each other in nature and in appearance. Consider first the READ statement in the preceding example:

```
READ 1, A, B, C
```

The first item after READ is the number of the FORMAT statement which describes the arrangement of data on the cards to be read. Then

there follows an ordered list of the variable names which are to be associated with the values to be read in (A, B, and C). Such a list can be as long as necessary.

In any case, when a READ statement is executed, a stream of numbers comes in from the card reader, and each variable name in the list part of the statement is associated with the value of the corresponding number, the first number corresponding to the first variable, the second number to the second variable, etc. The card reader continues to read in numbers until the last variable in the list has received a value, unless the cards are exhausted first. Both FORMAT statements 1 and 2 specify five numbers on a card; therefore, the stream of numbers which is brought in by giving either READ 1 or READ 2 consists of five numbers from the first card, five from the second, and so on until the list is exhausted. In the example, only three variables appear in the list. Thus, the list is completed before the end of the card is reached, and only the first three numbers on a card are read. Since each new execution of a READ begins with a new card, the values of A, B, and C must be placed in the first three of the five fields on each card. The arrangement of the numbers might appear as:

Field	1	2	3	4	5
card 1	3.	1.7	-31.92		
card 2	1.5794	-17.3	+0.00023		
card 3	-180.	-.001	4.20		
etc.					

As has been mentioned, PRINT statements are very similar to READ statements. PRINT 1 and PRINT 2 both specify that up to five numbers are to be printed on each line until all the values associated with the variables in the list given in the PRINT statement have been printed. The PRINT statement in the above example

PRINT 1, A, B, C, ROOT

would cause lines like the following to be printed each time the statement is executed:

+3.00000	+1.70000	-31.92000	(value of ROOT)
+1.57940	-17.30000	+0.00023	(value of ROOT)
etc.			

The difference between the two standard FORMAT statements is that FORMAT 1 calls for fixed point input or output whereas FORMAT 2 calls for floating point input or output.

Fixed Point Input:

If fixed point input conforming to FORMAT statement 1 is desired, data must be arranged in five fields. Each line of data is punched on a single card. Each number may have a sign and must have a decimal point; unsigned numbers are interpreted as positive. A maximum of ten digits per number is permitted.
Several examples follow.

Example 1:

Statement:

READ 1, A, B, C, D, E, F

Data Sheet

Case 1 A, B, C, D, E	1.0	+50001.	-.0007	160.	-0.0615
F	14.2				
Case 2 A, B, C, D, E	4.7	-1763.	+.0589	87.	-0.0023
F	-3.0				
etc.					

Example 2:

Statements (appearing together in a program):

READ 1, A, B, C

READ 1, D, E, F, G

Data Sheet (for both READ statements)

Case 1 A, B, C	150579.1	10000000.	15.1007		
D, E, F, G	-1005.7	-.00000005	+1.0003	14.	
Case 2 A, B, C	2704.3	100000.	23.0823		
D, E, F, G	-99.5	-.087654	+0.3879	7.	

Floating Point Input:

The data sheet for floating point input conforming to FORMAT statement 2 must be arranged with five major fields, each field having a right-hand sub-field wide enough for the exponent (a sign and two digits). The same examples could be written for floating point input as shown on the following page.

Example 1:

Statement:

READ 2, A, B, C, D, E, F

Data Sheet

Case 1 A, B, C, D, E	1.0	+00	+50001.	+00	-.0007	+00	1.6	+02	-6.15	-02		
F	1.42	+01										
Case 2 A, B, C, D, E	4.7	+00	-17630.	-01	+.0589	+00	8.7	+01	-2.30	-03		
F	-3.00	+00										

Example 2:

Statements:

READ 2, A, B, C

READ 2, D, E, F, G

Data Sheet

Case 1 A, B, C	1.505791	+05	1.	+07	15.1007	+00						
D, E, F, G	-1.0057	+03	5.0	-08	+1.0003	+00	1.4	+01				
Case 2 A, B, C	2.7043	+03	1.	+05	23.0823	+00						
D, E, F, G	-9.95	+01	-8.7654	-02	+3.879	-01	7.	+00				

There must always be three characters in the exponent sub-field, a sign and two digits. As in the case of fixed point input, there is a limit of ten digits per field; the maximum number of digits for the number is therefore eight, two digits being required for the exponent.

Fixed Point Output:

For fixed point output, PRINT 1 causes numbers to be printed with five decimal places, five numbers per line. PRINT 1 should not be used if any result will exceed 999,999. If the input data of the preceding examples were to be printed as output in the fixed point form specified by FORMAT statement 1, the FORTRAN statements and printed sheets would appear as shown on the following page.

Example 1:

Statement:

```
PRINT 1, A, B, C, D, E, F
```

Printed Sheet

```
 +1.00000  +50001.00000      -0.00070     +160.00000      -0.06150
+14.20000
 +4.70000   -1763.00000      +0.05890      +87.00000      -0.00230
 -3.00000
```

Example 2:

Statements:

```
PRINT 1, A, B, C

PRINT 1, D, E, F, G
```

Printed Sheet

```
+150579.10000 (too large to print)  +15.10070
  -1005.70000       +0.00000        +1.00030      +14.00000
  +2704.30000  +100000.00000       +23.08230
    -99.50000       -0.08765        +0.38790       +7.00000
```

Floating Point Output:

The use of PRINT 2 causes numbers to be printed in floating point form, up to five numbers to a line. The numbers appear with the sign and one digit to the left of the decimal point and five digits to the right of the decimal point, followed immediately by the letter E and the sign and the two digits of the exponent. The FORTRAN statements necessary to print the numbers from the preceding examples in floating point form appear on the following page along with examples of the printed sheets.

Example 1:

```
Statement:

   PRINT 2, A, B, C, D, E, F

Printed Sheet

+1.00000E+00  +5.00010E+04  -7.00000E-04  +1.60000E+02  -6.15000E-02
+1.42000E+01
+4.70000E+00  -1.76300E+03  +5.89000E-02  +8.70000E+01  -2.30000E-03
-3.00000E+00
```

Example 2:

```
Statements:

   PRINT 2, A, B, C

   PRINT 2, D, E, F, G

Printed Sheet

+1.50579E+05  +1.00000E+07  +1.51007E+01
-1.00570E+03  +5.00000E-08  +1.00030E+00  +1.40000E+01
+2.70430E+03  +1.00000E+05  +2.30823E+01
-9.95000E+01  -8.76540E-02  +3.87900E-01  +7.00000E+00
```

IF, Unconditional GO TO, and STOP Statements

Besides arithmetic, READ, and PRINT statements, three other types of statements are included in this section. Two of these have already appeared in the examples on pages 5 and 8: STOP and the unconditional GO TO. The latter is called "unconditional" to distinguish it from two other types of GO TO statements covered in Section III.

A STOP statement is used, as in the example on page 5, to tell the computer when the end of the calculation has been reached. It may be omitted in certain cases, such as that encountered in the example on page 8, where the absence of cards in the card reader causes an automatic halt.

The unconditional GO TO statement is used to specify at some point in a program that the next statement to be executed is not the one following, as it normally would be, but instead, the statement numbered n. The statement GO TO n transfers control to statement n, and execution proceeds from there.

As an introduction to a third type of control statement, consider the following problem:

> Given values a, b, c, and d punched on a card, and a set of values for the variable x punched one per card, evaluate the function defined by

$$f(x) = \begin{cases} ax^2 + bx + c & \text{if } x < d \\ 0 & \text{if } x = d \\ -ax^2 + bx - c & \text{if } x > d \end{cases}$$

> for each value of x, and print x and f(x).

The FORTRAN program for this problem could be written as follows:

C ← FOR COMMENT	STATEMENT NUMBER	CONTINUATION	FORTRAN STATEMENT
	10		READ 1, A, B, C, D
	11		READ 1, X
	12		IF (X-D) 13, 15, 17
	13		FOFX = A*X**2. + B*X +C
	14		GO TO 18
	15		FOFX = 0.
	16		GO TO 18
	17		FOFX = -A*X**2. + B*X - C
	18		PRINT 1, X, FOFX
	19		GO TO 11

The values for A, B, C, and D are read from the first card (statement 10), and the first value of X is read from the next card (statement 11). Statement 12 is then executed. Statement 12 means: "If the quantity (X-D) is negative, go to statement 13; if it is zero, go to statement 15; and if it is positive, go to statement 17." Hence, if X<D, the value of FOFX is calculated by means of the proper formula, and the execution of statement 14 transfers control to statement 18, which is executed next. Similarly, if X = D, control goes from the IF statement to the proper formula (statement 15), and then from statement 16 to statement 18. If X>D, the IF statement selects statement 17, after which statement 18 is automatically taken next. Thus, in all three cases, control eventually reaches statement 18, the PRINT statement, which prints the values of X and FOFX. Statement 19 then returns control to the READ statement, which reads in the next value of X. The whole pattern repeats until all of the X-cards have been processed. The 704 will automatically halt when it attempts to execute the READ with no more cards in the card reader.

As has been illustrated, the IF statement is a kind of conditional, three-way GO TO statement. It often happens, as in the above problem, that the computer must choose one of alternative paths depending on whether the current value of an expression is negative, positive, or zero. This is done, as in the above program, by writing

$$\text{IF (E) } n_1,\ n_2,\ n_3$$

where (E) is an arithmetic expression, and the number of the statement to which control is to be transferred is n_1 if (E) is negative; n_2, if (E) is zero; and n_3, if (E) is positive.

It is not necessary to number every FORTRAN statement in a program. The only statements which must be numbered are those to which reference is made (as in GO TO or IF statements). Any numbers between 1 and 32,000 may be used, provided two different statements are never given the same number. As has been mentioned earlier, statement numbers 1 and 2 are reserved for the two standard FORMAT statements when the data arrangements specified by these statements are desired.

Additional Examples

Example 1: Find the approximate numerical solution of the ordinary differential equation

$$dy / dx = xy + 1$$

in the interval $0 \leq x \leq 1$, given that

$$y = 0 \text{ when } x = 0$$

A method which can be used to approximate the solution of this equation is as follows:

Assume that a point $(x_0,\ y_0)$ of the solution function is known. In this example, the point $x_0 = 0$, $y_0 = 0$ is known. It is then known from the differential equation that dy / dx, the rate of change of y with respect to x, at this point is $x_0 y_0 + 1$. Hence, an increment in x of Δ x would produce an approximate change in y of

$$\Delta\ y = \Delta\ x(x_0 y_0 + 1)$$

Let Δ x be the interval between successive x_i terms ($i = 0,\ 1,\ 2,\ \ldots$). Then $y(\text{at } x_1) = y(\text{at } x_0 + \Delta x) = y_0 + \Delta\ y = y_0 + \Delta\ x(x_0 y_0 + 1)$.

After the point (x_1, y_1) has been obtained, it can be used to find (x_2, y_2) in a similar way:

$$y_2(\text{at } x_2) = y_2(\text{at } x_1 + \Delta x) = y_1 + \Delta x(x_1 y_1 + 1).$$

In this example, the procedure is continued until the point $x = 1$ is reached, the upper bound of the interval for which the solution is being found. In general, the equation for stepping forward is

$$y_{i+1} = y_i + \Delta x(x_i y_i + 1).$$

Since it depends on the mesh size, Δx, the error of approximation is left as a parameter in the program. The solution for various values of Δx can be compared to give an empirical idea of the error. To print the value of every point obtained would be unnecessary and costly, since Δx must be quite small, so the program has been arranged to print only at intervals of 0.01. The program for this example, with an explanation of some of the statements, follows.

C ← FOR COMMENT	STATEMENT NUMBER (1–5)	CONTINUATION (6)	FORTRAN STATEMENT (7–72)
			READ 1, DELTAX
			PRINT 1, DELTAX
			XPRINT = 0.01
			X = 0.0
			Y = 0.0
	3		Y = Y + DELTAX*(X*Y + 1.0)
			X = X + DELTAX
			IF (X - XPRINT) 3, 4, 4
	4		PRINT 2, X, Y
			XPRINT = XPRINT + 0.01
			IF (X - 1.0) 3, 5, 5
	5		STOP

The only input to the program is the value of DELTAX punched in fixed point form on a card. The first statement causes the 704 to read DELTAX from the card. The second statement causes the 704 to print DELTAX to head the answer sheet. The third statement initializes XPRINT to 0.01; the first value of X which equals or exceeds XPRINT will be the next value printed. The next two statements assign the proper initial values to X and Y. Statement 3 is the basic equation for finding the next value of Y. Notice that the previous value of Y is used in the calculation and then is replaced by the result of the calculation to give the new value of Y. The next statement calculates the new value of X. This value of X is then compared with the value of XPRINT; if it is less than this value, control goes back to calculate the next point. As soon as X

equals or exceeds XPRINT, the calculation is interrupted to allow the current values of X and Y to be printed according to statement 4 (in floating point form). The value of XPRINT is increased by 0.01 for the next value to be printed. Then a test is made to determine whether the value of X has reached 1.0. If X equals or exceeds 1.0, the problem is finished and the computer stops (statement 5); if not, control returns to statement 3 to calculate the next point. (Note: the time required by the computer to calculate a point is about 0.9 milliseconds. Hence, for the case $\Delta X = 10^{-5}$, i.e., 100,000 points, the calculation time would be about 1.5 minutes.)

Example 2: Determine the current in an alternating current circuit consisting of resistance, inductance, and capacitance in series, given a number of sets of values of resistance, inductance, and frequency. The current is to be determined for a number of equally spaced values of the capacitance (which lie between specified limits) for voltages of 1.0, 1.5, 2.0, 2.5, and 3.0 volts.

The equation for determining the current flowing through such a circuit is

$$i = \frac{E}{\sqrt{R^2 + \left(2\pi fL - \frac{1}{2\pi fC}\right)^2}}$$

where i = current, amperes
E = voltage, volts
R = resistance, ohms
L = inductance, henrys
C = capacitance, farads
f = frequency, cycles per second
π = 3.1416

The FORTRAN program could be written as follows:

C ← FOR COMMENT / STATEMENT NUMBER (1–5)	CONTINUATION (6)	FORTRAN STATEMENT (7–72)
10		READ 1, OHM, FREQ, HENRY
11		READ 2, FRD1, FRDFIN
12		PRINT 1, OHM, FREQ, HENRY
13		VOLT = 1.0
14		PRINT 1, VOLT
15		FARAD = FRD 1
16		AMP = VOLT/SQRTF(OHM**2. + (6.2832*FREQ*HENRY
	X	-1./(6.2832*FREQ*FARAD))**2.)
17		PRINT 2, FARAD, AMP
18		IF (FARAD - FRDFIN) 19, 21, 21
19		FARAD = FARAD + 0.000 000 01
20		GO TO 16
21		IF (VOLT - 3.0) 22, 10, 10
22		VOLT = VOLT + 0.5
23		GO TO 14

Statement 10 causes the values of the resistance, the frequency, and the inductance to be read, in that order, from the first card. Statement 11 causes the initial and final values of the capacitance to be read from the next card. The values of the resistance, frequency, and inductance are printed (statement 12) in fixed point form. The initial value of the voltage is introduced and printed (statements 13 and 14). Statement 15 initializes the current value of the capacitance (denoted by FARAD) to the first value to be used in calculation (denoted by FRD1). The actual calculation is specified by statement 16. The result of that calculation, together with the current value of the capacitance, is printed (statement 17).

The current value of the capacitance is compared with the final value to determine whether or not all values have been investigated (statement 18). If not, the expression (FARAD -FRD 1) is negative and the program proceeds to statement 19, which causes the current value of the capacitance to be increased by the given increment. This is followed by a transfer (statement 20) to statement 16 which causes the calculation to be repeated for the new value of the capacitance. If the expression in statement 18 is zero or positive, all values of the capacitance have been investigated and the program transfers to statement 21.

At this point the value of the voltage is compared with the upper bound to determine whether or not all specified values of the voltage have been used. If not, the expression in statement 21 (VOLT - 3.0) is negative and the program proceeds to statement 22, which causes the value of the voltage to be increased. Following this, a transfer (statement 23) is made to statement 14, which causes the new value of the voltage to be printed. The program proceeds to statement 15, and the entire process of investigating all values of the capacitance is begun again.

If all values of the voltage have been used (the expression in statement 21 is zero or positive), the calculations for the current set of values of resistance, frequency, and inductance are finished. The program is returned to statement 10 so that the two cards defining the next case may be read and the program repeated. This process is repeated until all of the cases have been considered; i.e., all of the cards have been read.

Check List

In the preceding descriptions of six types of instructions no attempt was made to cover in detail all of the information necessary or helpful in writing a program using these types. The following list of items, together with what has already been presented, supplies this information.

1. The basic characters which may be used in writing a FORTRAN statement are:
   a. A, B, C, ..., Z (26 alphabetic characters)
   b. 0, 1, 2, ..., 9 (10 numerical characters)

c. + (plus), - (minus), * (asterisk), / (slash), ((left parenthesis),) (right parenthesis), , (comma), = (equal sign), and . (decimal point).

2. Upper and lower case alphabetic characters are not distinguished on a punched card; e.g., D and d are represented by the same punches.
3. The digits 1 and 0 must be carefully distinguished from the alphabetic characters I and O.
4. If calculations involving a constant (i.e., any number except a statement number) are to be carried out in floating point arithmetic, as is always the case if only the types of instructions presented in this section are used, the constant must be written with a decimal point.
5. A variable symbol can consist of six or fewer characters. It must satisfy the following conditions:
   a. The first character must be alphabetic.
   b. The first character cannot be I, J, K, L, M, or N, which are set aside to denote integer variables, as discussed in Section II.
   c. Any character following the first may be alphabetic or numerical, but not one of the special characters.
   d. The names of all functions appearing on the computing center list, as well as these names with the terminal F removed, must not be used as variable symbols. For example, if SINF is used as the name of a function, neither SINF nor SIN can be used as a variable symbol.
6. If a function appearing on the computing center list is used, the name of the function, as written by the programmer, must agree exactly with the name as it appears on the list.
7. The argument of a function is enclosed in parentheses; e.g., SINF (X).
8. If a function has more than one argument, the arguments are separated by commas; e.g., SINF (X, Y, Z).
9. The left side of an arithmetic statement must never be a constant. In the type of arithmetic statement covered in this section, the left side is always a simple variable; e.g., A. In Section II, arithmetic statements will be extended to include function statements, in which the left side is a function of one or more variables.
10. Never omit the intended operation symbol between two quantities; e.g., do not write AB for A*B.
11. Never write two operation symbols in a row; e.g., do not write A* -B for A*(-B). There are no exceptions. The exponentiation symbol ** may seem to be an exception, but it is regarded as a single symbol.

12. Blank spaces can be used or not used as desired, since blanks are ignored in the translation. For example

```
A=0.1
```

could be written as

```
A = 0.1
```

and

```
GO TO 25
```

could be written as

```
GOTO25
```

13. The prescribed form for the various non-arithmetic statements must be followed exactly, except for the arbitrary use of blank spaces. For example, the statements

```
READ 1 A, B

IF A-B, 5, 6, 7
```

are incorrectly written. They should be written

```
READ 1, A, B

IF (A-B) 5, 6, 7
```

with the punctuation marks appearing exactly as shown.

14. The magnitude of every non-zero quantity must lie between 10^{-38} and 10^{38}. By "quantity" is meant any constant or any value assumed by a variable or function in the course of the calculation.
15. Numbers to be read by means of a READ 1 statement must not exceed 10 digits.
16. Numbers to be read by means of a READ 2 statement must not exceed 8 digits. The exponent must have two digits and a sign, making a total maximum of ten digits to a field.
17. Numbers to be printed by means of a PRINT 1 statement should not exceed 999,999.99999.
18. The program statement which is written last should be a STOP statement or a statement which causes a transfer to some other statement in the program (a GO TO or an IF statement).

SECTION II - DEFINITION OF FUNCTIONS, MANIPULATION OF SINGLE-SUBSCRIPTED VARIABLES, MAGNETIC TAPE INPUT AND OUTPUT

Introduction

The part of the FORTRAN language presented in Section I can be used to direct the operation of the 704 in the solution of certain problems. However, it is difficult or impossible to program the solution of some problems using only the six types of statements described in Section I. These six types of statements, grouped into classifications, were:

Arithmetic statements

Input-output statements { READ, PRINT

Control statements { IF, Unconditional GO TO, STOP

In this section arithmetic statements will be extended to include function statements, and additional types of statements will be introduced which make it possible to direct the 704 in the solution of problems more complex than those dealt with in Section I.

If programming were done using only the six types of statements presented in Section I, laborious programming would be necessary to carry out relatively simple iterative calculations or logical steps such as are encountered in the addition of two vectors or the selection of a certain number from a list of numbers. However, it is possible, using the additional types to be presented in this section, to employ the subscript notation of mathematics to make the programming of such problems easier.

A mathematician would denote that c_i is the sum of the vectors (a_1, a_2, a_3) and (b_1, b_2, b_3) by writing

$$c_i = a_i + b_i \qquad i = 1, 2, 3$$

Notice that the first part of the statement

$$c_i = a_i + b_i$$

is a general statement which, in effect, becomes three specific statements

$$c_1 = a_1 + b_1$$
$$c_2 = a_2 + b_2$$
$$c_3 = a_3 + b_3$$

by assigning the values 1, 2, and 3 to i.

By using the FORTRAN language, it is possible to make general statements like $c_i = a_i + b_i$, and to make other statements which assign the desired values to i. When a general statement is executed it is always executed in one of its specific senses. For example, if the variable I has the value 3 when the FORTRAN equivalent of $c_i = a_i + b_i$

$$C(I) = A(I) + B(I)$$

is executed, the values denoted by A(3) and B(3) are added and the sum is assigned as the value of C(3). Thus, to compute the sum vector

$$(C(1),\ C(2),\ C(3))$$

it is necessary to execute the general statement 3 times, each time with I having one of the values 1, 2, 3. Therefore, in addition to providing for arithmetic statements with subscripted variables, it is necessary to provide for a method of stating that a given set of such statements should be executed repetitively for certain values of the subscript. The FORTRAN statement which provides this ability is called a DO statement. An example of a DO statement, followed by an explanation, appears below.

```
DO  20 I = 1,  250
```

This statement instructs the 704: "Execute all statements which immediately follow, up to and including the statement numbered 20, 250 times (the first time for I = 1, the second time for I = 2, and so on, and the last time for I = 250), and then go on to the statement following statement 20." Thus, to return to the example of vector addition, the FORTRAN statements necessary to add A(I) and B(I) are

C ← FOR COMMENT	CONTINUATION	FORTRAN STATEMENT
STATEMENT NUMBER 1–5	6	7–72
		DO 1 I = 1,3
1		C(I) = A(I) + B(I)
2		...

When the statement numbered 2 is encountered, the values of C(1), C(2), and C(3) will have been computed and stored.

Example: It is required to compute the following quantities

$$P_i = \sqrt{\sin^2 (A_i B_i + C_i) + \cos^2 (A_i B_i - C_i)}$$

$$Q_i = \sin^2 (A_i + C_i) + \cos^2 (A_i - C_i)$$

for i = 1, ..., 100. A possible FORTRAN program for this calculation follows.

C = FOR COMMENT	STATEMENT NUMBER	CONTINUATION	FORTRAN STATEMENT
	1		TRIGF(X,Y) = SINF (X+Y)**2+COSF(X-Y)**2
	2		DIMENSION A(100), B(100), C(100), P(100), Q(100)
	3		READ 8, A, B, C
	4		DO 6 I = 1,100
	5		P(I) = SQRTF(TRIGF(A(I)*B(I), C(I)))
	6		Q(I) = TRIGF(A(I), C(I))
	7		PRINT 8, (A(I), B(I), C(I), P(I), Q(I), I = 1,100)
	8		FORMAT (5F 10.4)
	9		STOP

Statement 1 defines the function TRIGF(X, Y) as equal to the expression ($\sin^2 (X+Y) + \cos^2 (X-Y)$). The DIMENSION statement indicates that the arrays A, B, C, P, and Q each have 100 elements. A, B, and C in the READ statement will cause all elements of A, then all elements of B, and then all elements of C to be read into the 704 from cards. Notice that the READ statement refers to a new type of statement (8), FORMAT. In this example, the FORMAT statement specifies the external arrangement for both input and output data. In this FORMAT statement, 5F10.4 means: "There are 5 Fixed point decimal fields per card or line, each field being 10 columns wide with 4 decimal places to the right of the decimal point." Hence, A, B, C, P, and Q will be read or printed in the form __±XX.XXXX, that is, two blanks, a sign, two digits, a decimal point, and four digits, a total of 10 columns. Statement 4 says: "DO the following statements through statement 6 for I=1, I=2, ..., I=100." Statements 5 and 6 compute P_i and Q_i. The PRINT statement says: "Print the arrays A, B, C, P, and Q for I=1, ..., 100 as specified by FORMAT statement 8." Statement 9 stops the computer.

The method of subscript notation and the use of the DO, FORMAT, DIMENSION, and function statements which have been introduced here will be further illustrated in the following pages of this section. In addition, several new statements for input and output and tape manipulation will be presented.

Integer Constants and Variables

In Section I, only floating point constants (which must have a decimal point) and floating point variables (which must not begin with I, J, K, L, M, or N) were considered. However, it should be clear that floating point numbers are neither desirable nor necessary for use as subscripts; i.e., $X_{1.3}$ is not generally a useful notation, and $X_{3.0}$ is redundant and wastes space. Integer constants and integer variables are more useful for this purpose. The two rules which follow describe the method of writing such numbers:

1. Integer constants are written without a decimal point.
2. Integer variables must begin with I, J, K, L, M, or N.

When used in FORTRAN statements, a subscripted variable is written as the name of the variable followed by the subscript (an integer constant or variable) in parentheses; e.g., A(3) is the FORTRAN representation of A_3 and X(I) is the FORTRAN representation of X_i.

Subscripts are not restricted to single quantities. They may take the general form

$$K * I \pm L$$

where I represents any integer variable and K and L represent any unsigned integer constants (L may also be zero, in which case the form reduces to K*I). Further examples appear below:

Y(M+1) means Y_{m+1}

P(3*K-5) means P_{3k-5}

If a floating point variable, for example, A, is used as a subscripted variable, it represents the collection of variables A(1), A(2), A(3), ... etc. and may not be used without a subscript, except in an input/output statement (like READ or PRINT) when it is desired to transfer the entire array, or in an arithmetic statement where A will be interpreted as A(1). Thus it is not possible to use B(J) and B in different statements and expect to have both a vector, B(J), and a non-subscripted variable, B.

Reference to a subscripted variable whose subscript is an integer variable, for example, X(N), is always interpreted in a specific sense determined by the value of N. Therefore, some statement which assigns a value to N, such as

```
N = I + J
   or
DO 10, N = 1, 20
   or
READ 6, N
```

should always be encountered before reaching a statement which refers to X(N).

Integer quantities are not permitted to appear in floating point expressions except as subscripts or as exponents. However, an expression containing integer quantities only (such as I + J) may be written; such expressions will be evaluated using truncated integer arithmetic rather than floating point arithmetic. Some examples of expressions which are and are not permitted appear below.

Expression	Permissible	Arithmetic Used
A*B*(C**2)	Yes	Floating
2*A	No (no decimal point)	-------
I + J	Yes	Integer
2.*A	Yes	Floating
A**(I+J)	Yes	Floating
2*I	Yes	Integer
I + A	No	-------

As long as the expression on the right side of an arithmetic statement is a legitimate one as described above, there are no further restrictions on arithmetic statements. There are, however, certain pitfalls which may be encountered if arithmetic statements are written having an integer expression on one side and a floating point expression on the other. For example, the formula

```
I = A+B**J
```

instructs the 704 to compute the value of $A+B^J$ using floating point arithmetic, truncate the result (i.e., drop any fractional part), and assign the integer so obtained as the value of I. This meaning results from the fact that the expression on the right is a floating point expression whereas the variable on the left is an integer variable. Conversely, the formula

```
A = JOB + N/3
```

instructs the 704 to compute the value of JOB+N/3 using integer arithmetic, put the resulting integer in floating point form, and assign this as the value of A. Note that integer arithmetic gives an integer result even for N/3. Thus, the value of 8/3 would be 2, the largest integer not exceeding 8/3, whereas the value of 8./3. in a floating point expression is 2.66666...

DIMENSION Statements

Whenever a subscripted variable appears in a FORTRAN program, it is necessary to include a statement which indicates the size of the array referred to by this variable. This type of statement is a DIMENSION statement. A DIMENSION statement causes the 704 to assign the proper number of storage locations to each subscripted variable.

A DIMENSION statement consists of the name of each subscripted variable followed by an integer in parentheses which represents the greatest number of elements which will ever be included in the array. The variables are separated by commas, and the whole group of names is preceded by the word DIMENSION.

If the subscripted variables ALPHA(I), GAMMA(J), and VECTOR(N) appear in a FORTRAN program, then a DIMENSION statement mentioning these variables must also be included. Assume that the number of elements in ALPHA(I) will never exceed 100, the number in GAMMA(J) will never exceed 25, and the number in VECTOR(N) will never exceed 12. The DIMENSION statement must then be written

```
DIMENSION  ALPHA(100), GAMMA(25), VECTOR(12)
```

DIMENSION statements are not actually executed. No instructions corresponding to this statement will appear in the translated machine language program. In the FORTRAN program, however, a DIMENSION statement giving the size of each array must precede the first executable statement mentioning that array. A single DIMENSION statement, including all subscripted variables mentioned in the program, may be used, or separate statements may be inserted prior to mentioning each new array.

DO Statements

An example of the use of a DO statement of the unconditional type appeared in the introduction to this section. The usefulness of such a statement for carrying out repetitive calculations was mentioned then. The standard form for an unconditional DO statement is

DO N I = m_1, m_2

where N is a statement number

I is an integer variable

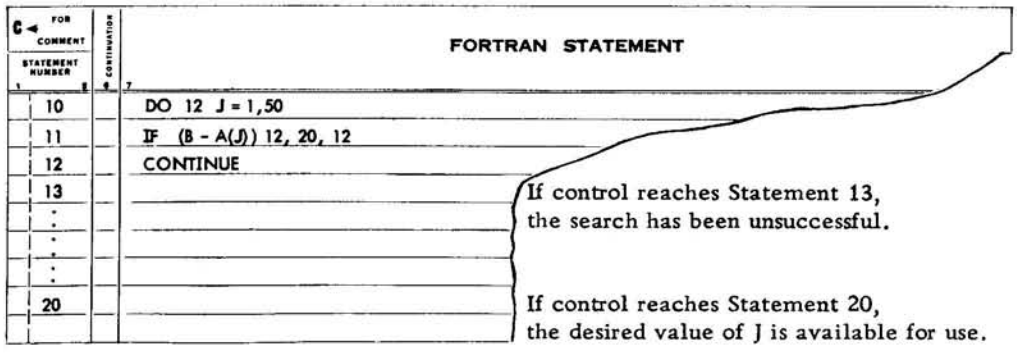
m_1 and m_2 are integer constants.

The meaning of the DO statement is: "Execute the statements immediately following this DO statement, up to and including the statement numbered N, first with I equal to m_1, then with I equal

to $m_1 + 1$, etc., and finally with $I = m_2$, and then go to the statement following statement N."

The set of statements immediately following the DO statement and extending through statement N is called the range of the DO statement. In Section III, the use of "nests" of DO statements, with one or more DO statements in the range of another, will be discussed. In the use of DO statements discussed in the present chapter, no DO statement contains another DO statement within its range. However, the range of a DO statement may contain GO TO or IF statements, and these may transfer control out of this range.

As a further illustration of the usefulness of the DO statement, consider a number B and a set of fifty numbers, A(J). The problem is to select the smallest of the values of J for which B = A(J), if there are one or more such values of J. A program to accomplish this could be written as follows:



Control passes to statement 20, out of the range of the DO statement, as soon as J, the index of the DO statement, reaches a value for which B-A(J) equals zero. Any reference which is now made to J will be interpreted for J equal to that specific value. Whenever B-A(J) is not equal to zero, control goes to the last statement in the range of the DO. This statement, CONTINUE, means "no operation." The reason for using it relates to the meaning of the DO statement. The DO statement causes the index, J in this example, to be increased by 1 each time the last statement in its range, statement 12 in this example, is reached, after which control goes to the first statement in its range. In this example, when B-A(J) is not zero, it is desired to increase J and begin the range again. To accomplish this, control must reach the last statement in the range (which cannot be the IF statement) even though no more work remains to be done with the current value of J. In this example, therefore, the last statement in the range of the DO statement must be CONTINUE, which means "do nothing."

Function Statements

Within the limits of the part of FORTRAN introduced in Section I, certain functions, specified by the computing center, were permitted in writing arithmetic expressions, such as square root, sine, log, etc. The functions were restricted to those appearing in the list furnished by the computing center.

It is also possible, however, to write expressions involving functions peculiar to the problem at hand. Each desired function is defined by means of a function statement. For example, suppose it is desired to use the function

$$G(X) = 1.3 + \sqrt{4.1X+X^2}$$

several times in a program. The function statement defining G(X) might be written as follows:

```
GXXF(X) = 1.3 + SQRTF(4.1*X+X**2)
```

A later arithmetic formula in the program, employing GXXF, might be

```
Y = 10.3*GXXF(ALPHA*BETA) + 14.7
```

In this use of GXXF, before the value of the function is computed, the quantity ALPHA*BETA will be substituted for X in the expression defining GXXF.

In general, function statements must obey the following rules:

1. All function statements in a program must be the first executable statements in that program.
2. The function name must have four to seven alphabetic or numerical characters; the first must be alphabetic, and the last must be F.
3. The name of the function is followed by parentheses enclosing the argument or arguments. Multiple arguments are separated by commas. Each argument must be a single non-subscripted variable.
4. Any argument which is a floating point variable in the definition of a function should be a floating point quantity in any subsequent use of the function. A similar rule applies to integer arguments.
5. The value of a function is a floating point quantity unless the name of the function begins with X, in which case the value is an integer quantity.

The following example illustrates some properties of function statements.

C ← FOR COMMENT / STATEMENT NUMBER (1–5)	CONTINUATION (6)	FORTRAN STATEMENT (7–72)
1		FIRSTF(X) = X**2 + A**2
2		SECONDF(R, S) = SQRTF(FIRSTF(R/(R+S)))
		.
		.
		.
		.
		.
		.
		.
15		Q(I) = FIRSTF(Y*B(I))
		.
		.
		.
		.
		.
		.
		.
27		P = SECONDF(1.7*DELTA, ALPHA)*PI

Notice that it is permissible to use a previously defined function in the definition of subsequent functions. Notice also that the variable A is involved in the definition of FIRSTF but is not an argument. A may be used in the same way as any other variable in the problem, and its current value is used each time FIRSTF is evaluated.

The Meaning of a List

Examples of lists have already appeared in READ and PRINT statements in this section, although they were not identified as such. A list is a set of items separated by commas; when a list appears in an input or output statement, the order of reading or writing is the order of the items as written.

For example, the statement

PRINT 20, A, B, C

has the list A, B, C; the quantities A, B, and C will be printed in that order. If any of the items A, B, or C have been specified in a DIMENSION statement as arrays, then the values of each element of the array will be printed. For example, if A and C are simple variables and B has been specified in a DIMENSION statement as a subscripted variable having 3 elements, then the quantities which the 704 would be instructed to print by means of the PRINT statement above are

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

If A and B were large arrays and one wished to specify the reading or writing of the quantities

A(1), B(1), A(2), B(2), ...A(100), B(100)

in that order, the list would consist of the single item

(A(I), B(I), I = 1, 100)

If one wished to specify the first seven elements of the array A, followed by the first five elements of the array B, the list would consist of the two items

(A(I), I = 1, 7), (B(I), I = 1, 5)

However, if A and B had dimensions seven and five respectively, the simpler list

A, B

would give the same results, but in the reverse order.

When, as above, an item in a list specifies part of an array or a mixture of arrays, the item must be enclosed in parentheses and the variables inside must be separated by commas as shown. The indexing information (e.g., I = 1, 100) is written exactly as in a DO statement.

FORMAT Statements

An input or output statement, such as READ or PRINT, specifies the variables which are to receive values or are to be printed. It also refers to the number of a FORMAT statement which specifies the arrangement of a line of input and/or output data. The FORMAT statement contains the specifications for each field in the line. There are three general forms for a field specification

Iw, Ew.d, Fw.d

where Iw indicates an Integer decimal number having a field width of w columns; Ew.d indicates a floating decimal point number (E), having a field width of w columns, and d places to the right of the decimal point; Fw.d indicates a Fixed decimal point number, having a field width of w columns, and d places to the right of the decimal point. For example, the statements

```
25    FORMAT (E10.4, F8.3, F7.5, E9.2, I3, F4.1)

      READ 25, A, B, C, D, I, E
```

might be used to instruct the 704 to read the following lines of input data from cards:

Field 1		Field 2	Field 3	Field 4		Field 5	Field 6
+.8765	E+06	+345.648	+.56872	-2.34	E+01	+81	-1.5
-.1223	E-02	+124.785	-.78963	-6.78	E+09	+15	+9.8
+.1034	E+05	-728.654	+.12345	+4.35	E-07	-28	-2.3

Note that the field width includes a column each for the sign, decimal point, and, in the case of floating decimal point numbers, the four characters of the exponent: the letter E, the sign of the exponent, and the two digits of the exponent. Floating decimal or fixed decimal point numbers may have any number of digits in the input field, depending on specifications in the FORMAT statement; however, only eight significant digits will be retained in the calculations. If the decimal point punched in the card does not agree with the specifications in the FORMAT statement, the decimal point overrides the specification; i.e., calculations will be performed with the decimal point as placed on the card. If no decimal point is given, the number is treated as if the decimal point were located according to the specification. No provision is made for handling decimal integers larger than 32767. A line of input data may have a maximum of 72 characters.

Prior to sending the input data sheets to the computing center for punching, the card columns for each of the fields must be specified. Columns 1-72 are available for use. For this example, the columns for field 1 should be specified as columns 1-10 (since 10 columns are specified by the FORMAT statement for the first field); field 2, columns 11-18; field 3, columns 19-25; field 4, columns 26-34; field 5, columns 35-37; and field 6, columns 38-41.

Specifying a field width larger than the number of characters in the field is particularly valuable for use with output statements. For example, the statements

```
      PRINT 10, A, B, C, D, I, E

10    FORMAT (E14.4, F11.3, F10.5, E13.3, I6, F7.1)
```

would cause the 704 to print the example data as follows:

```
___ 0.8765E 06___ 345.648___ .56872___-0.234E 02___ 81___-1.5

___-0.1223E-02___ 124.785___-.78963___-0.678E 10___ 15___ 9.8

___ 0.1034E 05___-728.654___ .12345___ 0.435E-06___-28___-2.3
```

Note the three-column separation between fields (represented by dashes) provided for by the FORMAT statement (10). In the case of floating decimal point numbers, the field width includes the zero preceding the decimal point. Floating decimal point numbers are printed with the first significant digit immediately to the right of the decimal point; therefore, these numbers have as many significant digits as there are decimal places specified. However, no more than eight significant digits are possible. A maximum of 119 characters may be printed per line.

The FORMAT statement is not executed and may be placed anywhere in the program. The field specifications are enclosed in parentheses with commas between the specifications for successive fields. Successive fields having the same format may be specified by inserting a coefficient (indicating the number of identical fields) before the code letter E, F, or I. Thus

```
FORMAT (I3, 2E12.3, F8.4)
```

is equivalent to

```
FORMAT (I3, E12.3, E12.3, F8.4)
```

It may be of interest to consider now the FORMAT statements which were referred to as 1 and 2 in Section I. FORMAT statement 1, which referred to fixed decimal point input and output, is written as

```
1  FORMAT (5F14.5)
```

The code letter F is preceded by the number 5, which indicates how many times this specification is to be repeated per line. The field width of 14 allows for six digits and a sign to the left of the decimal point (7 spaces), the decimal point (1 space), and five digits to the right of the decimal point (5 spaces), plus one additional space for field separation.

FORMAT statement 2, which referred to floating decimal point input and output, is written as

```
2  FORMAT (1P5E14.5)
```

The scale factor (1P) shifts the decimal point so that there is one significant digit to the left of the decimal point. (See Section III for scale factors.) The field width of 14 allows for one digit and a sign to the left of the decimal point, the decimal point, five digits to the right of the decimal point, the four-character exponent field, plus two additional spaces for field separation.

Magnetic Tape Input and Output

Thus far, the only methods that have been mentioned for transferring decimal data into and out of the magnetic core storage of the 704 are use of the on-line card reader for input and use of the on-line printer for output. The operation of these units is controlled by the READ and PRINT statements introduced in Section I. However, there is another method for transferring information into and out of magnetic core storage, the use of magnetic tapes.

1. READ INPUT TAPE and WRITE OUTPUT TAPE Statements:

Just as a READ statement directs the 704 to read data from cards, a READ INPUT TAPE statement directs the 704 to read data into core storage from magnetic tape. Similarly, just as the use of a PRINT statement causes on-line printing, a WRITE OUTPUT TAPE statement causes the output information to be written on magnetic tape.

Most computer installations have two machines available which are not connected to the computer. One of these machines can read information from punched cards and write this information on magnetic tape. The other machine can read information written on magnetic tape and print this information. By means of the card-to-tape machine, the information contained in a deck of data cards can be written on magnetic tape, which the 704 can subsequently be instructed to read by means of a READ INPUT TAPE statement. The output information written on magnetic tape by a WRITE OUTPUT TAPE statement can subsequently be printed by using the tape-to-printer machine.

The advantages of using magnetic tape for input and output lies in the fact that the computer reads from and writes on magnetic tape much more rapidly than it reads cards and prints. This means that a great deal of computer time which would otherwise be required for card reading and printing can be relegated to the relatively inexpensive card-to-tape and tape-to-printer equipment.

The 704 computer may have up to ten attached tape units which, in the FORTRAN language, are referred to by the numbers 1, 2, ..., 10. The general form of the two input-output statements mentioned above is

READ INPUT TAPE I, N, List

WRITE OUTPUT TAPE I, N, List

where I is the number of the tape unit (an integer between 1 and 10 or an integer variable), N is the number of a FORMAT statement, and "List" denotes a list of names of quantities to be read or written.

2. General Information about the Use of Tapes:

Information is recorded linearly on magnetic tape in blocks called records, which may be of various lengths. Each record is separated from the next by a gap of blank tape called the end-of-record gap.

In order to indicate that the last record of information has been written on a tape, the statement

END FILE I

where I is the number of the tape unit, is used. This causes an end-of-file mark to be written on the specified tape. Later, during reading, the end-of-file mark is recognized by the tape reading mechanism as a direction to stop tape reading at that point.

When a WRITE OUTPUT TAPE statement is executed, the tape mechanism writes the current values of the quantities in the list as one record or successive records, with the division into records determined by the associated FORMAT statement. For example, assuming that the list refers to five single quantities, the statement

WRITE OUTPUT TAPE 6, 1, A, B, C, D, E

causes the five numbers which are the current values of A, B, C, D, and E to be written, in accordance with standard FORMAT statement 1, as a single record, in the order A, B, C, D, E, on the tape mounted on tape unit 6. Physically, the tape is moved forward over the stationary tape read-write head, which magnetically records the five numbers and then erases a short segment of tape as the end-of-record gap. At the end of this operation, the tape is in position for the writing of the next record. No end-of-file mark is written.

Again assuming that the list refers to five single quantities, the effect of the statement

READ INPUT TAPE 6, 1, A, B, C, D, E

is to move the tape mounted on tape unit 6 forward over the read-write head, causing the 704 to start reading the next five numbers (which must constitute a record conforming to FORMAT 1), assigning the first number as the value of A, the second as the value of B, etc. After the five numbers called for by the list have been stored, control passes to the next executable statement, leaving the tape positioned for the reading of the next record.

The above examples are simple cases, intended to give an idea of the movement of the tape in response to the program, and of the way information is transmitted between core storage and tape.

A tape can be read or written only in the forward direction. However, there are two statements which can be used to move the

tape backward: REWIND I and BACKSPACE I. REWIND I moves tape I back to the physical starting point regardless of its current position. BACKSPACE I moves tape I back to the beginning of the preceding record. If the tape is in a rewound position, a BACKSPACE statement has no effect.

In order to move a tape forward one record without reading any information into storage, the statement

READ INPUT TAPE I, n

may be used, where n is the number of the FORMAT statement describing the record to be skipped. Note the omission of the list.

By means of REWIND I, BACKSPACE I, and READ INPUT TAPE I, n (without a list), a tape can be positioned for reading or writing at the beginning of any record desired. However, because of the nature of the tape read-write mechanism, writing a new record on tape makes it impossible to read any of the old information physically following this new record on the tape. Since the tape can be positioned only at the beginning of a record, it is not possible to begin reading or writing in the middle of a record.

There are two other types of statements for programming tape operations: READ TAPE and WRITE TAPE statements. For information about these types, see the Programmer's Reference Manual for FORTRAN, Form 32-7026.

Additional Examples

Several examples which illustrate the use of many of the statements introduced in this section appear below.

Example 1: It is required to calculate the amount of heat necessary to raise the temperature of a mixture of ten gases from a given base temperature, T_1, to a series of higher temperatures. These temperatures are 25 degrees apart and range from T_1 up to a maximum of T_2.

The heat required may be calculated by multiplying the heat capacity of the gas mixture by the temperature difference. However, the heat capacity is dependent upon the temperature. The mean heat capacity over a given range may be estimated by using the equation

$$C_p = a + \frac{b}{2}(T + T_0) + \frac{c}{3}(T^2 + TT_0 + T_0{}^2)$$

where C_p = the mean heat capacity

T = the upper temperature, degrees Kelvin

T_0 = the lower temperature, degrees Kelvin

a, b, c = empirical constants, different for each gas

(degrees Kelvin = degrees Centigrade + 273.1)

Input data to the program must therefore include the amount of each gas present, the three empirical constants, (a, b, c) for each gas, the base temperature (T_1, in ^{o}C), and the maximum temperature (T_2, in ^{o}C).

A possible FORTRAN porgram to carry out this calculation appears below. It has been written to provide the individual heat capacities in each range as well as the total heat requirement. The program incidentally illustrates the fact that statements need not be numbered sequentially.

C = FOR COMMENT / STATEMENT NUMBER	CONTINUATION	FORTRAN STATEMENT
9		DIMENSION X(10), A(10), B(10), C(10), CP(10)
10		FORMAT (10F8.3)
11		FORMAT (10E11.3)
12		READ INPUT TAPE 4, 10, X, A, T1, T2
13		READ INPUT TAPE 4, 11, B, C
14		T1K = T1 + 273.1
15		TK = T1K
16		TK = TK + 25.0
17		SUM = 0.0
18		IF ((TK - 273.1)-T2) 19, 19, 27
19		DO 21 I = 1, 10
20		CP(I) = A(I)+B(I)*(TK+T1K)/2.0+C(I)*(TK**2+TK*T1K+T1K**2)/3.0
21		SUM = X(I)*CP(I)+SUM
22		HEAT = SUM*(TK-T1K)
23		T = TK - 273.1
24		WRITE OUTPUT TAPE 5, 31, T1, T, HEAT
25		WRITE OUTPUT TAPE 5, 32, X
36		WRITE OUTPUT TAPE 5, 34, CP
26		GO TO 16
27		IF (T2 - 2500.) 12, 28, 28
28		END FILE 5
29		REWIND 4
30		REWIND 5
31		FORMAT (2F10.1, E15.5)
32		FORMAT (10F8.3)
34		FORMAT (5E14.5)
33		STOP

The DIMENSION statement sets aside storage locations for the constants and results. Statements 10 and 11 describe the

arrangement of the input data as follows:

X (fractional amount of each gas) = 0.xxx

A = ±x.xx

B = ±xx.xxxE±ee

C = ±x.xxxE±ee

T1, T2 = ±xxxx.x

Statements 32 and 34 describe the arrangement of the output data as follows:

X = 0.xxx

CP = 0.xxxxxE±ee

T1, T = ±xxxx.x

HEAT = ±0.xxxxxE±ee

Statements 12 and 13 cause the data for a case to be transferred into the 704 from tape unit 4. The calculation of the absolute temperature in degrees Kelvin from the base temperature is carried out by statement 14. Statement 15 sets the initial value of the temperature range to T_1 °K. Statement 16 causes the range to be increased by the specified increment. Statement 17 sets the location designated as SUM to zero. The upper limit of the range is compared to the maximum temperature specified for this case. If the maximum has not been reached, control reaches the DO statement (statement 19). The statements in the range of the DO (statements 20 and 21) cause the specific heat of each component to be calculated and weighted according to the fraction of that component in the mixture. The actual calculation of the heat requirement is described by statement 22. Statement 23 causes the upper limit of the range to be expressed in degrees Centigrade. Writing of the results, along with the fractions of each component, on tape unit 5 is accomplished by statements 24, 25, and 36. A transfer to begin the calculation for the next range is effected by statement 26.

If the comparison at statement 18 indicates that the maximum temperature for the given case has been exceeded, control reaches statement 27. At this point, the maximum temperature is examined to determine whether it exceeds 2500°C (which is the indication that the problem has been completed). If the problem has been completed, control reaches statement 28, an end-of-file mark is written, the

tapes used in the program are rewound (statements 28, 29, and 30), and the 704 stops. If the problem has not been completed, control is transferred to statement 12, which causes data for a new case to be read from the input tape.

Example 2: Given X_i, Y_i, Z_j for i = 1, ..., 10 and j = 1, ..., 20, compute:

$$PROD = \left(\sum_{i=1}^{10} A_i \right) \left(\sum_{j=1}^{20} Z_j \right)$$

where

$$A_i = X_i^2 + Y_i \quad \text{if } |X_i| > |Y_i|$$

$$A_i = X_i + Y_i^2 \quad \text{if } |X_i| < |Y_i|$$

$$A_i = 0 \quad \text{if } |X_i| = |Y_i|$$

A possible FORTRAN program follows

C ← FOR COMMENT / STATEMENT NUMBER	CONTINUATION	FORTRAN STATEMENT
3		DIMENSION X(10), Y(10), Z(20)
4		FORMAT (5F14.4)
5		READ 4, X, Y, Z
6		SUM A = 0.0
7		DO 12 I = 1, 10
8		IF(ABSF(X(I)) - ABSF(Y(I))) 9, 12, 11
9		SUMA = SUMA+X(I)+Y(I)**2
10		GO TO 12
11		SUMA = SUMA+X(I)**2+Y(I)
12		CONTINUE
13		SUMZ = 0.0
14		DO 15 J = 1, 20
15		SUMZ = SUMZ+Z(J)
16		PROD = SUMA*SUMZ
17		PRINT 4, SUMA, SUMZ, PROD
18		STOP

The DIMENSION statement sets aside storage locations for the input data. Statement 4 specifies the input and output data as fixed point numbers having 4 decimal places. The READ statement reads the input data from cards into the 704. Statement 6 sets the quantity SUMA to zero. Statements 8-12, under control of the DO statement 7, compute $\sum_{i=1}^{10} A_i$. Statement 15 computes $\sum_{j=1}^{20} Z_j$ under the control of DO statement 14. The following statements compute and print

PROD. Statement 12, CONTINUE, serves as a common reference point; and since it is the last statement in the range of the DO, I is increased after its completion, and the next repetition is begun.

Check List

1. All subscripted variables must appear in a DIMENSION statement. This statement must appear in the program before reference is made to the variables.
2. Negative subscripts are not permitted.
3. Subscripting of subscripts is not permitted.
4. In a floating-point expression, integer variables and constants can be used only as subscripts and/or exponents.
5. Integer constants are written without a decimal point; integer variables must begin with I, J, K, L, M, or N.
6. The names of all functions defined in the program or appearing on the computing center list, as well as these names with the terminal F removed, must not be used as variable symbols. For example, if SINF is used as the name of a function, neither SINF nor SIN can be used as a variable symbol.
7. If a subscripted variable has 4 or more characters in its name, the last of these must not be an F. For example, SINF(I) cannot be used as a subscripted variable, regardless of whether SINF is used as the name of a function.
8. The last statement in the range of a DO must not be a transfer.
9. Decimal integers larger than 32767 are treated modulo 32768.
10. An end-of-file mark should always be written on output tapes.
11. Provision for rewinding tapes should be made in the program.
12. No constants may be given in a list for an input/output statement, only variables.
13. FORMAT statements for output must be written so that the first character of the first field is a blank.

(Refer to the end of Section I for additional points.)

SECTION III - MANIPULATION OF TWO- AND THREE-DIMENSIONAL ARRAYS

Introduction

The following is a list of the 15 types of statements, grouped into classifications, which have been presented in Sections I and II:

Arithmetic statements	
Control statements	IF Unconditional GO TO STOP DO CONTINUE
Input-output statements	READ PRINT FORMAT READ INPUT TAPE WRITE OUTPUT TAPE REWIND BACKSPACE END FILE
Specification statement	DIMENSION

Several of the statements introduced in Section II offered a convenient method for handling one-dimensional arrays in an iterative manner. However, no provision was made for handling two- and three-dimensional arrays. The present section will describe additional features. In this section, subscripting is extended to two- and three-dimensional arrays. This provision greatly facilitates the solving of many engineering and scientific problems which require matrix manipulations for their solution. Several new statements will also be introduced.

The following example of matrix multiplication will serve to illustrate DO nests and multiple subscripts. (A DO nest is a set of two or more DO statements, the range of one of which includes the ranges of the others.)

Given the matrix A with dimensions 10 x 15 and the matrix B with dimensions 15 x 12, compute the elements C_{ij} of the matrix C = AB. To compute any element C_{ij}, select the i row of A and the j column of B, and sum the products of their corresponding elements. The general formula for this computation is

$$C_{ij} = \sum_{k=1}^{15} A_{ik} B_{kj}$$

The following is a possible FORTRAN program for this matrix multiplication.

C ← FOR COMMENT / STATEMENT NUMBER (1–5)	CONTINUATION (6)	FORTRAN STATEMENT (7)	
		DIMENSION A(10, 15), B(15, 12), C(10, 12)	
3		FORMAT (5E14.5)	
		READ 3, A, B	
4		DO 30 I = 1, 10	Range of 1st DO
5		DO 30 J = 1, 12	Range of 2nd DO
6		C(I, J) = 0.0	
10		DO 20 K = 1, 15	Range of 3rd DO
20		C(I, J) = C(I, J) + A(I, K)*B(K, J)	
30		PRINT 50, I, J, C(I, J)	
50		FORMAT (2I5, E16.7)	
60		STOP	

The DIMENSION statement says: "Matrix A is of maximum size 10 x 15, matrix B is of maximum size 15 x 12, and matrix C is of maximum size 10 x 12." The READ statement reads all elements of the matrix A and then all elements of matrix B into the 704 from punched cards, the format of which is specified by statement 3. Since two-dimensional arrays are stored column-wise, the matrices A and B must be punched column-wise; i.e. all the A_i of column 1, followed by all the A_i of column 2, etc. (A_{11}, A_{21}, A_{31}, A_{41}, ..., $A_{10,\ 15}$), and similarly for matrix B. Notice that statements 6 through 30 constitute a program similar to programs considered in Section II. Whatever values I and J have at the moment, this program computes and prints C(I, J) along with I and J. Statement 5 says that this program is to be repeated 12 times, first for J = 1, then for J = 2, ..., J = 12. Notice that for each repetition of statements 6 through 30 statement 20 is executed 15 times, first for K = 1, then for K = 2, and so on. Thus, when the process called for by statement 5 is complete, the I row of the product matrix has been computed and printed. In a similar manner, statement 4 causes the program from statement 5 to statement 30 to be repeated for the appropriate values of I, thereby producing all of the rows of the product matrix.

This example illustrates the fact that one or more DO statements may appear in the range of a DO statement. This nesting of DO statements can result in a single statement being the last statement in the range of several DO statements. For example, statement 30 is the last one in the range of DO statements 4 and 5. Consequently, a more general rule is needed to describe the flow of control and the incrementing of indices following the last statement in the range

of a DO; the following rule holds for DO ranges which have the same last statement:

> Upon the completion of the last statement in the range of a DO, control passes to the first statement in the range of the nearest preceding DO which is not yet completed and which has the same last statement, and the index of that DO is incremented. The last statement in the range of a DO may not be a control statement (e.g., IF, GO TO, DO, etc.). If all DO ranges containing this last statement as the end of their range are completed, control passes to the next statement.

Subscripts for Two- and Three-Dimensional Arrays

In the preceding example of matrix multiplication, A, B, and C were two-dimensional arrays. As was noted, each variable had two subscripts which were separated by commas, and the set of two subscripts was enclosed in parentheses.
For example:

```
A(I, K)

B(K, J)

C(I, J)
```

Three-dimensional arrays are denoted by the use of three subscripts. For example:

```
X(M, N + 10, 5*L)
```

The same rules presented in Section II regarding the formation of subscripts apply to the two- and three-dimensional cases.

The DIMENSION statement is similarly extended to two- or three-dimensional arrays. For example, the statement

```
DIMENSION W(10, 10, 15), ALPHA(15, 5), V(20, 10)
```

causes 1500 locations in storage to be set aside for the three-dimensional array W, 75 locations for the two-dimensional array ALPHA, and 200 locations for the two-dimensional array V.

DO Nests

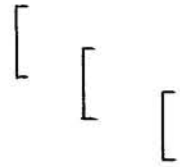
There are certain rules which must be observed when using DO statements within the range of another DO statement:

1. If the range of a DO statement includes another DO statement, all statements in the range of this second statement must also be in the range of the first DO

statement. The following diagram illustrates this rule.

Permitted

Violation of Rule 1



2. No transfer of control by IF or GO TO statements is permitted into the range of any DO statement from outside its range, since such transfers would not permit the DO loop to be properly indexed. The following diagram illustrates this rule.

Permitted

Violation of Rule 2

All of the DO statements so far presented were written in the form

$$\text{DO N I} = m_1, m_2$$

In these cases, the index, I, started at the specified value, m_1, and was increased by one each time the statements in the range of the DO were executed, until the value of I equaled m_2. It is possible, however, to achieve greater flexibility in the DO statements by adding a third fixed point number so that the general form is

$$\text{DO N I} = m_1, m_2, m_3$$

In this case the value of the index, I, starts at m_1 (as before), but it is increased by m_3 (which may be different from one) each time, until the value of I equals or exceeds m_2, at which point the DO is "satisfied." It is not necessary to include the increment, m_3, in the DO statement unless the increment is different from one; i.e., the statements

```
DO 20 I = 1, 10
```

and

```
DO 20 I = 1, 10, 1
```

are equivalent.

Every type of calculation is permitted in the range of a DO with one exception. No calculation which changes the value of the index or any of the indexing parameters (m_1, m_2, m_3) of the DO statement is permitted within the range of that DO statement. The indexing parameters (m_1, m_2, m_3) may be either integer constants or non-subscripted integer variables.

Lists for Two- and Three-Dimensional Arrays

The extension of the input-output statements to govern the transfer of two-and three-dimensional arrays to or from magnetic core storage requires only that the subscripting information given earlier be used when writing the list. If the list

```
JOBNO, CASE, RUN, K, (X(I), Y(I, K), I = 1, 4),
    ((Z(I, J), I = 1, 3), W(J, 3), J = 1, 3)
```

were used with an input statement, the successive words, as they were read into the 704, would be interpreted as the following sequence of variables and placed in the storage locations (previously assigned by FORTRAN) in that same order:

JOBNO, CASE, RUN, K, X(1), Y(1, K), X(2), Y(2, K),

X(3), Y(3, K), X(4), Y(4, K), Z(1, 1), Z(2, 1), Z(3, 1),

W(1, 3), Z(1, 2), Z(2, 2), Z(3, 2), W(2, 3), Z(1, 3), Z(2, 3),

Z(3, 3), W(3, 3)

Note that a variable subscript (K) was used at one point. This is permissible only if that variable has been previously assigned a value (in this case, a value would have been read in earlier).

To transfer a complete array, subscripting and index information is not necessary. Such information is provided, in this case, by the DIMENSION statement. Using the example above, the statements

```
DIMENSION ALPHA (15, 5)

READ 1, ALPHA
```

would cause the entire 75 word array

ALPHA(1, 1), ALPHA(2, 1), ALPHA(3, 1), ALPHA(4, 1)...,

ALPHA(15, 1), ALPHA(1, 2), ALPHA(2, 2), ALPHA(3, 2),

ALPHA(4, 2)..., ALPHA(15, 5)

to be transferred into magnetic core storage in the reverse of the above order.

Assigned GO TO Statements

One modification of the GO TO statement which allows greater freedom in directing the logical flow of a program is the assigned GO TO statement. The assigned GO TO statement requires a companion statement, an ASSIGN statement, which must be previously executed.

As an example of the use of the assigned GO TO statement, suppose it is desired to calculate several average values such as average temperature, pressure, and density. If the data is on cards, the following program might be used:

C ← FOR COMMENT	STATEMENT NUMBER	CONTINUATION	FORTRAN STATEMENT
			DIMENSION X(25)
	5		ASSIGN 30 TO N
	10		READ 2, X
			SUM = 0.0
	15		DO 20 I = 1, 25
	20		SUM = SUM + X(I)
	25		AVG = SUM/25.0
	26		GO TO N, (30, 40, 50)
	30		AVGTEM = AVG
	31		ASSIGN 40 TO N
			GO TO 10
	40		AVGPRE = AVG
	41		ASSIGN 50 TO N
			GO TO 10
	50		AVGDEN = AVG
			PRINT 60, AVGTEM, AVGPRE, AVGDEN
			STOP
	60		FORMAT (3E14.5)

In this example, statement 26 transfers control to one of the three statements referred to in the list, i.e., 30, 40 or 50, depending upon the value of N at the time of execution, which is determined by the last preceding ASSIGN statement. The first execution of statement 26 causes control to be transferred to statement 30, since statement 5, the last preceding ASSIGN statement, assigned the value of 30 to N. Statement 31 assigns the value of 40 to N; hence the second execution of statement 26 transfers control to statement 40. The third execution of statement 26 transfers control to statement 50, the value of 50 having been assigned to N by statement 41.

In general terms, the assigned GO TO statement is written

$$\text{GO TO N, } (n_1, n_2, \ldots, n_m)$$

where N is a non-subscripted integer variable appearing in a previously executed ASSIGN statement, and n_1, n_2, ..., n_m stand for statement numbers. These statement numbers are, in effect,

a list of values which may be assigned to N. Note the comma which is inserted between the variable and the left parenthesis; it must always be included.

The statement

ASSIGN 30 TO N

is not equivalent to the arithmetic formula

N = 30

A variable N which currently has a value is either an assigned variable or an ordinary variable, never both simultaneously. It is an assigned variable if its current value has been established by an ASSIGN statement (e.g., ASSIGN 30 TO N); it is an ordinary variable if its current value has been established by an arithmetic formula (e.g., N = 30). The current value is the one given by the last previous ASSIGN statement or arithmetic formula, whichever was most recently executed. A variable N which is currently an assigned variable is effective only in assigned GO TO N statements. An ordinary variable N is effective in all statements involving N except assigned GO TO N statements.

There is a restriction on the assigned GO TO statement when it lies in the range of a DO statement, in addition to the general restrictions on transfers on pages 45 and 46. This restriction requires that the statements to which the assigned GO TO statement may transfer must all lie in one single part of the nest which includes the range, or must all lie outside the nest. If this condition cannot be met, it may be possible by suitable programming changes to use a computed GO TO statement to accomplish the desired branching, since there is no such restriction on this type of statement.

Computed GO TO Statements

Computed GO TO statements are similar to assigned GO TO statements in that both types establish a many-way fork. They differ in that an assigned GO TO statement requires a companion statement (ASSIGN) to pre-set or assign a current value to the integer variable in the GO TO statement and thereby select the proper branch. The value of the integer variable in a computed GO TO statement may be arrived at by computation; no companion statement (comparable to ASSIGN) is necessary.
Example:

Given: A_i, B_i, N_i, X_i, Y_i for i = 1, ..., 10, where, for each i, N_i = 1 or 2, compute

$$Z_i = \sqrt{A_i X_i^2 + B_i Y_i} \qquad \text{for } N_i = 1$$

$$Z_i = \sqrt{A_i X_i^2 - B_i Y_i} \qquad \text{for } N_i = 2.$$

A possible FORTRAN program follows.

```
C ← FOR COMMENT
STATEMENT NUMBER   CONTINUATION   FORTRAN STATEMENT
1 ... 5            6              7 ... 72
      DIMENSION A(10), B(10), N(10), X(10), Y(10), Z(10)
      READ 3, (A(I), B(I), N(I), X(I), Y(I), I = 1, 10)
    3 FORMAT (2E13.5, I3, 2E13.5)
    5 DO 21 I = 1, 10
    6 J = N(I)
    7 GO TO (10,20), J
   10 Z(I) = SQRTF(A(I)*X(I)**2 + B(I)*Y(I))
   11 GO TO 21
   20 Z(I) = SQRTF(A(I)*X(I)**2 - B(I)*Y(I))
   21 PRINT 23, A(I), B(I), N(I), X(I), Y(I), Z(I)
   22 STOP
   23 FORMAT (2E13.5, I3, 3E13.5)
```

In this program, statement 7 transfers control to statement 10 if J = 1 or to statement 20 if J = 2. The ten values of N_i read into the program are each either 1 or 2. Since J is set equal to N_i by statement 6, the correct formula for Z_i is selected, depending on whether the current value of N_i is 1 or 2. Statement 6 is necessary since J cannot be a subscripted variable; subscripted variables are not allowed in computed GO TO statements.

As illustrated in the program for the example, computed GO TO statements have the form

$$\text{GO TO } (n_1, n_2, \ldots, n_m), \text{ I}$$

where the $n_1, n_2, \ldots, n_m$ stand for statement numbers, and I is a non-subscripted integer variable. Control is transferred to the first statement in the list (statement n_1) if, at the time of execution, the value of I is one; it is transferred to the second statement in the list (statement n_2) if the value of I is two, etc. Any number of statement numbers may appear in the list. The current value of I may be arrived at in any manner desired (e.g., in the program above, by an arithmetic formula modified by DO indexing), and its value at the time of execution of the computed GO TO statement determines which branch will be taken by the program. Note the comma which is inserted between the right parenthesis and the variable.

FORMAT Statements

In Section II the basic field specifications Iw, Ew.d, and Fw.d were introduced. In the present section, scale factors, Hollerith fields, and multiple-line formats will be discussed.

1. Scale Factors:

The use of scale factors allows greater flexibility in an output format. The specification

(2E14.4)

might print the following output line (dashes stand for blank spaces):

-0.4321E 04___0.5674E-06

If the specification were written as

(2P2E14.4)

the same output data would be printed with six significant digits, with the decimal point four places from the right. For example, the same output data as above might print as

-43.2147E 02___56.7439E-08

The scale factor 2P causes the floating point number to be multiplied by 10^2 and the exponent to be reduced by 2 prior to printing. Only a positive scale factor may be used with an E-type specification. However, positive or negative scale factors may be used with an F-type specification. For example, the specification

(-1PF10.3, 7PF12.3)

would print the following data

-4321.47 .0000005674

as

-432.147_____5.674

The scale factor is assumed to be zero if no other value has been given. Once a value has been given, however, it will hold for all subsequent E- and F-conversions within the same FORMAT statement until a new value is given. If it is desired to specify a scale factor of zero subsequent to another scale factor within the same FORMAT statement, 0P must be written. For example, the specification

(1PF10.1, F12.9)

would print the preceding data as

-43214.7__.000005674

The same data would be printed by the specification

(1PF10.1, 0PF12.9)

as

-43214.7__.000000567

The scale factor has no effect on I-conversion.

2. Hollerith Fields

English text may be printed by specifying a Hollerith field. Such fields are designated by the letter H preceded by a number designating the number of characters in the text; the field designation is followed by the desired English characters (including blanks). In order to print the factors X and Y along with their product, the FORMAT statement

10 FORMAT (2HX = F8.3, 4H__Y = F8.3, 5H__XY = F8.3)

could be used to print the output line

X = __10.723__Y = -_-12.561__XY = -134.692

Note that there is no comma after a Hollerith field specification (e.g., 4H__) in the FORMAT statement.

3. Multiple-line Format:

Within the limitations of FORMAT statements as presented in Section II, in order to print the following lines of output data,

_-67.8912E-03__106.23__-73

______732______82.976_6.25

two FORMAT statements would haye been necessary, for example

10 FORMAT (2PE13.4, 0PF8.2, I5)

11 FORMAT (I9, F12.3, F5.2)

However, with the introduction of multiple-line formats, only one FORMAT statement is required to print the above lines

12 FORMAT (2PE13.4, 0PF8.2, I5/I9, F12.3, F5.2)

The slash (/) separates the formats for the different lines of each set of lines. Thus, in this example, lines 1, 3, 5, ... have the format (2PE13.4, 0PF8.2, I5) and lines 2, 4, 6, ... have the format (I9, F12.3, F5.2); successive pairs of lines constitute the sets described by the FORMAT statement. Each line may have a maximum of 119 characters.

Example Problem and Program

The following example illustrates the use of many of the types of instructions presented in the three sections of this primer:

The n points (x_i, y_i) are given to fit by the least-squares method an m degree polynomial

$$y = a_0 + a_1 x + a_2 x^2 + \ldots + a_m x^m$$

In order to obtain the coefficients $a_0, a_1, \ldots\ldots, a_m$, it is necessary to solve the normal equations

(1) $$S_0 a_0 + S_1 a_1 + \ldots + S_m a_m = V_0$$

(2) $$S_1 a_0 + S_2 a_1 + \ldots + S_{m+1} a_m = V_1$$

. . . . .
. . . . .
. . . . .

(m + 1) $$S_m a_0 + S_{m+1} a_1 \ldots + S_{2m} a_m = V_m$$

where $S_0 = n$ $\qquad V_0 = \sum_{i=1}^{n} y_i$

$S_1 = \sum_{i=1}^{n} x_i$ $\qquad V_1 = \sum_{i=1}^{n} y_i x_i$

$S_2 = \sum_{i=1}^{n} x_i^2$ $\qquad V_2 = \sum_{i=1}^{n} y_i x_i^2$

. .
. .
. .

$V_m = \sum_{i=1}^{n} y_i x_i^m$

$S_{2m} = \sum_{i=1}^{n} x_i^{2m}$

After the S's and V's have been computed, the normal equations are solved by the method of elimination which is illustrated by the following solution of the normal equations for a second degree polynomial (m = 2).

$$(1)\quad S_0 a_0 + S_1 a_1 + S_2 a_2 = V_0$$

$$(2)\quad S_1 a_0 + S_2 a_1 + S_3 a_2 = V_1$$

$$(3)\quad S_2 a_0 + S_3 a_1 + S_4 a_2 = V_2$$

The forward solution is as follows:

1. Divide equation (1) by S_0.
2. Multiply the equation resulting from step 1 by S_1 and subtract from equation (2).
3. Multiply the equation resulting from step 1 by S_2 and subtract from equation (3).
   The resulting equations are

$$(4)\quad a_0 + b_{12} a_1 + b_{13} a_2 = b_{14}$$

$$(5)\quad b_{22} a_1 + b_{23} a_2 = b_{24}$$

$$(6)\quad b_{32} a_1 + b_{33} a_2 = b_{34}$$

where $b_{12} = S_1/S_0$, $b_{13} = S_2/S_0$, $b_{14} = V_0/S_0$

$$b_{22} = S_2 - b_{12} S_1,\quad b_{23} = S_3 - b_{13} S_1,\quad b_{24} = V_1 - b_{14} S_1$$

$$b_{32} = S_3 - b_{12} S_2,\quad b_{33} = S_4 - b_{13} S_2,\quad b_{34} = V_2 - b_{14} S_2$$

Steps 1 and 2 are repeated using equations (5) and (6), with b_{22} and b_{32} instead of S_0 and S_1. The resulting equations are

$$(7)\quad a_1 + c_{23} a_2 = c_{24}$$

$$(8)\quad c_{33} a_2 = c_{34}$$

where $c_{23} = b_{23}/b_{22}$, $c_{24} = b_{24}/b_{22}$

$$c_{33} = b_{33} - c_{23} b_{32}$$

$$c_{34} = b_{34} - c_{24} b_{32}$$

The backward solution is as follows:

$$(9)\quad a_2 = {}^{c}34/c_{33} \qquad \text{from equation (8)}$$

$$(10)\quad a_1 = c_{24} - c_{23}a_2 \qquad \text{from equation (7)}$$

$$(11)\quad a_0 = b_{14} - b_{12}a_1 - b_{13}a_2 \qquad \text{from equation (4)}$$

The following is a possible FORTRAN program for carrying out the calculations for the case: n = 100, $m \leq 10$. S_0, S_1, S_2, ..., S_{2m} are stored in SUM (1), SUM (2), SUM (3), ..., SUM (2M + 1), respectively. V_0, V_1, V_2, ..., V_m are stored in V (1), V (2), V (3), ..., V (M + 1), respectively.

C ← FOR COMMENT / STATEMENT NUMBER (1–5)	CONTINUATION (6)	FORTRAN STATEMENT (7–72)
		DIMENSION X(100), Y(100), SUM(21), V(11), A(11), B(11,12)
		READ 3, M, N
3		FORMAT (I2, I3)
		READ 4, (X(I), Y(I), I = 1, N)
4		FORMAT (4E14.7)
		LS = 2*M + 1
		LB = M + 2
		LV = M + 1
		DO 5 J = 2, LS
5		SUM(J) = 0.0
		SUM(1) = N
		DO 6 J = 1, LV
6		V(J) = 0.0
		DO 16 I = 1, N
		P = 1.0
		V(1) = V(1) + Y(I)
		DO 13 J = 2, LV
		P = X(I)*P
		SUM(J) = SUM(J) + P
13		V(J) = V(J) + Y(I)*P
		DO 16 J = LB, LS
		P = X(I)*P
16		SUM(J) = SUM(J) + P
17		DO 20 I = 1, LV
		DO 20 K = 1, LV
		J = K + I
20		B(K, I) = SUM(J-1)
		DO 22 K = 1, LV
22		B(K, LB) = V(K)
23		DO 31 L = 1, LV
		DIVB = B(L, L)
		DO 26 J = L, LB
26		B(L, J) = B(L, J)/DIVB
		I1 = L + 1
		IF (I1 - LB) 28, 33, 33
28		DO 31 I = I1, LV
		FMULTB = B(I, L)
		DO 31 J = L, LB

(continued on next page)

(continued from preceding page)

C ← FOR COMMENT	STATEMENT NUMBER	CONTINUATION	FORTRAN STATEMENT
	31		B(I, J) = B(I, J) - B(L, J)*FMULTB
	33		A(LV) = B(LV, LB)
			I = LV
	35		SIGMA = 0.0
			DO 37 J = I, LV
	37		SIGMA = SIGMA + B(I-1, J)*A(J)
			I = I-1
			A(I) = B(I, LB) - SIGMA
	40		IF(I-1) 41, 41, 35
	41		PRINT 42, (A(I), I = 1, LV)
	42		FORMAT (5E15.6)

The elements of the SUM and V arrays, except SUM(1), are set equal to zero. SUM(1) is set equal to N. For each value of I, X_I and Y_I are selected. The powers of X_I are computed and accumulated in the correct SUM counters. The powers of X_I are multiplied by Y_I and the products are accumulated in the correct V counters. In order to save machine time when the object program is being run, the previously computed power of X_I is used when computing the next power of X_I. Note the use of variables as index parameters. By the time control has passed to statement 17, the counters have been set as follows:

$$\text{SUM (1)} = N \qquad V(1) = \sum_{I=1}^{N} Y_I$$

$$\text{SUM (2)} = \sum_{I=1}^{N} X_I \qquad V(2) = \sum_{I=1}^{N} Y_I X_I$$

$$\text{SUM (3)} = \sum_{I=1}^{N} X_I^2 \qquad V(3) = \sum_{I=1}^{N} Y_I X_I^2$$

$$\vdots \qquad\qquad \vdots$$

$$V(M+1) = \sum_{I=1}^{N} Y_I X_I^M$$

$$\text{SUM (2M + 1)} = \sum_{I=1}^{N} X_I^{2M}$$

By the time control has passed to statement 23, the values of $S_0, S_1, \ldots, S_{2m}$ have been placed in the storage locations corresponding to columns 1 through M + 1, rows 1 through M + 1, of the B array, and the values of $V_0, V_1, \ldots, V_m$ have been stored in the locations corresponding to column M + 2, rows 1 through M + 1, of the B array. For example, for the illustrative problem beginning on page 54 (M = 2), columns 1 through 4, rows 1 through 3, of the B array would be set to the following computed values:

S_0	S_1	S_2	V_0
S_1	S_2	S_3	V_1
S_2	S_3	S_4	V_2

This matrix represents equations (1), (2), and (3) on page 54, the normal equations for M = 2.

The forward solution, which results in equations (4), (7), and (8) in the illustrative problem, is carried out by statements 23 through 31. By the time control has passed to statement 33, the coefficients of the A_I terms in the M + 1 equations which would be obtained in hand calculations have replaced the contents of the locations corresponding to columns 1 through M + 1, rows 1 through M + 1, of the B array, and the constants on the right-hand side of the equations have replaced the contents of the locations corresponding to column M + 2, rows 1 through M + 1, of the B array. For the illustrative problem, columns 1 through 4, rows 1 through 3, of the B array would be set to the following computed values:

1	b_{12}	b_{13}	b_{14}
0	1	c_{23}	c_{24}
0	0	c_{33}	c_{34}

This matrix represents equations (4), (7), and (8) on page 54.

The backward solution, which results in equations (9), (10), and (11) in the illustrative problem, is carried out by statements 33 through 40. By the time control has passed to statement 41, which prints the values of the A_I terms, the values of the M + 1 A_I terms have been stored in the M + 1 locations for the A array. For the

illustrative problem, the A array would contain the following computed values for a_2, a_1, and a_0, respectively:

Location	Contents
A(3)	c_{34}/c_{33}
A(2)	$c_{24} - c_{23}\, a_2$
A(1)	$b_{14} - b_{12}\, a_1 - b_{13}\, a_2$

The resulting values of the A_I terms are then printed according to the FORMAT specification in statement 42.

Debugging

In order to debug a FORTRAN program, it is recommended that extra PRINT statements under the control of a sense switch be used. The sense switches are located on the 704 console and may be used to control the program. The following statement is used in conjunction with the sense switches.

IF (SENSE SWITCH i) n_1, n_2

where i stands for the number of one of the six sense switches 1, 2, 3, 4, 5, or 6, and n_1 and n_2 are statement numbers. If sense switch i is in UP status, control is transferred to statement number n_2; if sense switch i is in DOWN status, control is transferred to statement number n_1. The following example illustrates the use of sense switches as an aid in debugging a program.
Example:

Given: a_i, b_i, and c_i for i = 1, ..., 10, compute and print

$$\text{RESULT} = \left(\sum_{i=1}^{10} (a_i c_i)^2\right)\left(\sum_{i=1}^{10} (b_i - c_i)\right) \Big/ \left(\sum_{i=1}^{10} (a_i b_i - c_i^2)\right)$$

Assume that the following FORTRAN program has been written and compiled (i.e., translated into machine language by the 704 by means of the FORTRAN system) and is to be tested:

C ← FOR COMMENT / STATEMENT NUMBER (1–5)	CONTINUATION (6)	FORTRAN STATEMENT (7–72)
		DIMENSION A(10), B(10), C(10)
		SUM1 = 0.0
		SUM2 = 0.0
		SUM3 = 0.0
		READ 1, (A(I), B(I), C(I), I = 1, 10)
		DO 10 I = 1, 10
		SUM 1 = SUM1 + (A(I)*C(I))**2
		SUM 2 = SUM2 + B(I) - C(I)
		SUM 3 = SUM3 + A(I)*B(I) - C(I)
		IF (SENSE SWITCH 1) 10, 5
5		PRINT 1, SUM 1, SUM 2, SUM 3
10		CONTINUE
		RESULT = SUM1*SUM2/SUM3
		PRINT 1, RESULT
		STOP

A test case is run using the compiled program. The 704 operator is instructed to run the test case with sense switch 1 in UP status (which causes the printing of intermediate results). Assume the test case has the following input data

$$a_1 = -.23456 \qquad b_1 = 12.34111 \qquad c_1 = 27.86523$$

Then the first line of output is

42.72019 -15.52412 -30.75996

Hand calculations for i = 1, using the original formula for RESULT, show that

SUM 1 = 42.72019

SUM 2 = -15.52412

SUM 3 = -779.36577

The hand-computed results for SUM 1 and SUM 2 agree with the output results; however, SUM 3 results do not agree. By looking at the FORTRAN statement which computes SUM 3, the error is located. The statement is changed from

SUM 3 = SUM3 + (A(I)*B(I) - C(I)

to

SUM 3 = SUM3 + (A(I)*B(I) - C(I)**2

After the indicated change is made, the FORTRAN program is again compiled and the test re-run. This time the machine results agree with the hand-computed results for all three sums. The 704 operator is instructed to run the program with sense switch 1 in DOWN status. With sense switch 1 DOWN the IF (SENSE SWITCH) statement transfers control to statement 10; therefore, no intermediate results are printed.

Storage

Many problems which are to be solved using the 704 will require the use of magnetic tapes and/or drums for additional storage. In order to determine whether additional storage is necessary or not, do the following:

1. Multiply the number of FORTRAN statements by 10; call this value A.
2. Add up the number of locations required by entries in DIMENSION statements; call this value B. For example, A(12, 6) requires 72 locations.
3. Use the rules given below to determine the number of storage locations needed for input/output routines; call this value C.
4. The list of available functions provided by the computing center should give the number of locations required for each function. Add up the number of locations required for all the functions used in the program; call this value D.
5. If (A + B + C + D) is much greater than the number of locations in the storage unit of the 704 to be used for running the program, the program will have to be rewritten, using tapes and drums for auxiliary storage of data. If (A + B + C + D) is nearly equal to the number of locations in the storage unit, the program should be compiled to find the precise number of locations required, since (A + B + C + D) is merely an estimate.

If it is necessary to use magnetic tapes and/or drums for intermediate storage, consult the Programmer's Reference Manual for FORTRAN, Form 32-7026, for information regarding the necessary statements. This manual contains additional control statements and covers particular situations in which some of the restrictions presented here may be relaxed. It also includes information regarding limitations on the size of a FORTRAN program (e.g., the number of variables, the size of DO nests, the number of transfer statements, etc.).

Rules for estimating storage required for input/output routines:

1. For each of the following types of statements which appear in the program add the corresponding number once. (If, for

example, several PRINT statements appear, add in 258 only once.)

PRINT	258
READ	137
READ INPUT TAPE	21
WRITE OUTPUT TAPE	12
PUNCH	90

2. Add to the above total:

If there is both decimal input and output,	945
If only decimal output,	484
If only decimal input,	461

Master Check List

1. The basic characters which may be used in writing a FORTRAN statement are
   a. A, B, C, ..., Z (26 alphabetic characters)
   b. 0, 1, 2, ..., 9 (10 numerical characters)
   c. + (plus), - (minus), * (asterisk), / (slash), ((left parenthesis),) (right parenthesis), , (comma), = (equal sign), and . (decimal point).
2. Upper and lower case alphabetic characters are indistinguishable on a punched card; e.g., D and d are represented by the same punches.
3. The digits 1 and 0 must be carefully distinguished from the alphabetic characters I and O.
4. A variable symbol can consist of six or less characters. It must satisfy the following conditions:
   a. The first character must be alphabetic.
   b. The first character cannot be I, J, K, L, M, or N, unless the symbol is an integer variable; if the symbol is an integer variable, the first character must be I, J, K, L, M, or N.
   c. Any character following the first may be alphabetic or numerical, but not one of the special characters.
   d. The names of all functions defined in the program or appearing on the computing center list, as well as these names with the terminal F removed, must not be used as variable symbols. For example, if SINF is

used as the name of a function, neither SINF nor SIN can be used as a variable symbol.

e. If a subscripted variable has four or more characters in its name, the last of these must not be an F. For example, SINF(I) cannot be used as a subscripted variable, regardless of whether SINF is used as the name of a function.

5. The name used for a function in programming must agree exactly with the name appearing in the list of functions.
6. The argument of a function is enclosed in parentheses; e.g., SINF (X).
7. If a function has more than one argument, the arguments are separated by commas; e.g., SINF (X, Y, Z).
8. The left side of an arithmetic formula must always be a variable or a function of one or more variables.
9. Never omit the operation symbol between two quantities; e.g., do not write AB for A*B.
10. Never have two operation symbols in a row; e.g., do not write A*-B for A*(-B). The exponentiation symbol ** may appear to be an exception, but it is regarded as a single symbol.
11. Blank spaces can be used or not used as desired, since blanks are ignored in the translation.
12. The prescribed form for the various non-arithmetic statements must be followed exactly except for the arbitrary use of blank spaces.
13. The magnitude of every non-zero quantity must lie between 10^{-38} and 10^{38}. By "quantity" is meant any constant or any value assumed by a variable or function in the course of the calculation.
14. Numbers to be read by means of a READ 1 statement must not exceed 10 digits.
15. Numbers to be read by means of a READ 2 statement must not exceed 8 digits. The exponent must have two digits and a sign.
16. Numbers to be printed by means of a PRINT 1 statement should not exceed 999,999.99999.
17. The physically last statement of a program should be a STOP statement or a statement (GO TO or IF) which causes a transfer to some other statement in the program.
18. All subscripted variables must appear in a DIMENSION statement, which must appear in the program before reference is made to the variables.
19. Negative subscripts are not permitted.
20. Subscripting of subscripts is not permitted.

21. Subscripts for two- and three-dimensional arrays should be separated by commas.
22. In a floating-point expression, integer variables and constants can be used only as subscripts and exponents.
23. Integer constants are written without a decimal point.
24. Decimal integers larger than 32767 are treated modulo 32768.
25. If the range of a DO includes another DO, then all statements in the range of this second DO must also lie within the range of the first DO.
26. Transfers into the range of any DO from outside its range are not permitted.
27. The last statement in the range of a DO must not be a transfer.
28. No calculation which changes the index or indexing parameters of a DO is permitted within the range of that DO.
29. Assigned GO TO statements have a comma between the variable and the left parenthesis.
30. Computed GO TO statements have a comma between the right parenthesis and the variable.
31. An ASSIGN statement must be encountered by the program prior to encountering an assigned GO TO statement.
32. The ASSIGN statement is not equivalent to an arithmetic formula.
33. When an assigned GO TO lies in the range of a DO, all statement numbers to which control may be transferred must lie in a single part of the DO nest which includes the range, or be completely outside the nest.
34. An end-of-file should always be written on output tapes.
35. Provision for rewinding tapes should be made in the program.
36. No constants may be given in a list in an input/output statement, only variables.
37. FORMAT statements for output must be so written so that the first character of the first field is a blank.

Summary of FORTRAN Statements

The complete FORTRAN language provides for 32 types of statements, which may be grouped as follows:

1. Arithmetic statements
2. Control statements (15 types)
3. Input/output statements (13 types)
4. Specification statements (3 types)

This manual has covered 19 types of statements:

1. Arithmetic statements
2. The following 9 types of control statements:
   a. Unconditional GO TO
   b. Assigned GO TO
   c. Computed GO TO

d. ASSIGN
e. IF
f. IF (SENSE SWITCH)
g. STOP
h. DO
i. CONTINUE

3. The following 8 types of input/output statements:
   a. FORMAT
   b. READ
   c. READ INPUT TAPE
   d. PRINT
   e. WRITE OUTPUT TAPE
   f. END FILE
   g. REWIND
   h. BACKSPACE
4. The following type of specification statements:
   DIMENSION

The types of FORTRAN statements which have not been covered in this manual are:

1. The following 6 types of control statements:
   a. SENSE LIGHT
   b. IF (SENSE LIGHT)
   c. IF ACCUMULATOR OVERFLOW
   d. IF QUOTIENT OVERFLOW
   e. IF DIVIDE CHECK
   f. PAUSE
2. The following 5 types of input/output statements:
   a. PUNCH
   b. READ TAPE
   c. READ DRUM
   d. WRITE TAPE
   e. WRITE DRUM
3. The following 2 types of specification statements:
   a. EQUIVALENCE
   b. FREQUENCY

Having approached the FORTRAN language cumulatively through the three stages presented in the sections of this primer, the reader should have little difficulty in extending his knowledge of FORTRAN to include the entire FORTRAN language as presented in the Programmer's Reference Manual for FORTRAN, Form 32-7026.

INTERNATIONAL BUSINESS MACHINES CORPORATION, 590 MADISON AVE., NEW YORK 22, NEW YORK

32-0306

Litho in U.S.A.