210

PRELIMINARY REPORT

Programming Research Group
Applied Science Division
International Business Machines Corporation

November 10, 1954

Specifications for

The IBM Mathematical FORmula TRANslating System,

FORTRAN

PRELIMINARY REPORT

Specifications for the IBM Mathematical FORmula TRANslating System, FORTRAN

The IBM Mathematical Formula Translating System or briefly, FORTRAN, will comprise a large set of programs to enable the IBM 704 to accept a concise formulation of a problem in terms of a mathematical notation and to produce automatically a high speed 704 program for the solution of the problem. The logic of the 704 is such that, for the first time, programming techniques have been devised which can be applied by an automatic coding system in such a way that an automatically coded problem, which has been concisely stated in a language which does not resemble a machine language, will be executed in about the same time that would be required had the problem been laboriously hand coded. Heretofore, systems which have sought to reduce the job of coding and debugging problems have offered the choice of easy coding and slow execution or laborious coding and fast execution.

It is felt that FORTRAN offers as convenient a language for stating problems for machine solution as is now known. Studies have indicated that a hand coded program for a problem will usually contain at least 5 times as many characters and sometimes 20 times as many characters as the problem statement in FORTRAN language. Furthermore, after an hour course in FORTRAN notation, the average programmer can fully understand the steps of a procedure stated in FORTRAN language without any additional comments.

Before considering the way in which a problem may be presented for automatic coding by the FORTRAN system, it might be well to consider some of the advantages of such a system. Basically, of course, the reason for the existence of high speed computers is the fact that they make possible the solution of problems in a much shorter time and at much less cost than would otherwise be required. The time and cost required for the solution of a problem on a high speed calculator fall roughly into 4 catagories:

1. Analysis and Programming
2. Coding
3. Debugging
4. Machine Solution

Faster and more capacious machines will considerably reduce the cost and time required for item 4 but so far the advent of new machines seems to have done little to reduce either the cost or time required for items 1, 2, and 3. It seems to be quite generally true that the personnel costs of a computing installation are at least as great as the machine cost. Furthermore, it is reasonable to assume that personnel cost for coding and debugging constitute considerably more than half the total personnel cost. Finally, at installations which have relatively few long term problems, as much as 1/2 of the machine cost is devoted to debugging. Therefore, in a crude fashion

one can say that out of every dollar spent to solve an average problem on a high speed computer, less than 25 cents is spent for analysis and programming, more than 25 cents is spent for personnel coding and debugging cost, about 25 cents for machine debugging cost, and about 25 cents for machine running cost.

Since FORTRAN should virtually eliminate coding and debugging, it should be possible to solve problems for less than half the cost that would be required without such a system. Furthermore, since it will be possible to devote nearly all usable machine time to problem solution instead of only half the usable machine time, the output of a given machine should be almost doubled. Also, of course, the total elapsed time for the solution of a problem should be a small fraction of the time required without FORTRAN since the time required for coding and debugging is usually more than 3/4 the total elapsed time. Not only does FORTRAN greatly reduce the initial investment in producing a program, but it will reduce even more the cost of reprogramming problems for future IBM calculators, since each such calculator should have a system similar to FORTRAN accompanying it which could translate the statement of the problem in a language very much like FORTRAN to its own code.

In addition to FORTRAN's great potentialities for economy, such a system will make experimental investigation of various mathematical models and numerical methods more feasible and convenient both in human and economic terms. Also, FORTRAN may apply complex, lengthy techniques in coding a problem which the human coder would have neither the time nor inclination to derive or apply. Thus, in many cases, FORTRAN may actually produce a better program than the normal human coder would be apt to produce.

Finally, the amount of knowledge necessary to utilize the 704 effectively by means of FORTRAN is far less than the knowledge required to make effective use of the 704 by direct coding. Information concerning how to use subprograms, what machine instructions are available, how to optimize a sequence of calculations, and concerning a large number of other coding techniques, is built into the FORTRAN system and it is not necessary for the programmer to be familiar with this information. In fact, a great deal of the information the programmer needs to know about the FORTRAN system is already embodied in his knowledge of mathematics. Thus it will be possible to make the full capabilities of the 704 available to a much wider range of people than would otherwise be possible without expensive and time consuming training programs.

In summary, then, a system such as FORTRAN has the following potentialities :

1. Great economy of time and money.
2. Feasibility of more mathematical experiments.
3. Ability to apply complex, lengthy techniques in coding a problem.

4. Ability to make the 704 avilable to more people with more convenience and less training.

Before beginning a description of the FORTRAN system, it should be noted that the following description is intended only to indicate present plans. Although the methods by which FORTRAN will operate are well understood, future developments in programming FORTRAN may necessitate certain minor changes in the system as it is presented below.

The following is a description of the admissible symbols and combinations of symbols in the FORTRAN language and how to use it:

1. CONSTANTS

A. FIXED POINT (INTEGERS)

i) General Form:

1 to 5 sequential decimal digits optionally preceded by a plus or minus sign

ii) Examples:

3
+1
-34500

B. FLOATING POINT

i) General Form:

Any sequence of decimal digits with a decimal point preceding or intervening between any 2 digits or following a sequence of digits, all of this optionally preceded by a plus or minus sign.

The number must be less than 10^{38} in absolute value and

greater than 10^{-38} in absolute value.

ii) Examples:

17.0
5.0
256.32
-.0033

2. VARIABLES

A. FIXED POINT VARIABLES

i) General Form:

A sequence of 1 or 2 alphabetic or numeric characters the first one of which is one of the following: i, j, k, l, m, n

ii) Examples:

i, ia, ii, ij, i1

B. FLOATING POINT VARIABLES

i) General Form:

A sequence of 1 or 2 alphabetic or numeric characters where the first character is an alphabetic character, not one of the following:

i, j, k, l, m, n

ii) Examples:

a, aa, ab, ai, a1

3. OPERATIONS

A. UNARY OPERATIONS (OPERATING ON A SINGLE VARIABLE OR EXPRESSION)

i) + Take the value of the following constant, variable or expression.

ii) - Take the negative of the value of the following constant, variable or expression.

B. BINARY OPERATIONS

i) + Add the constant, variable or expression preceding to the constant, variable or expression following.

ii) - Subtract

iii) x Multiply

iv) / Divide. Note that a/b/c=(a/b)/c

v) xx Exponentiation. $axxb=a^b$

4. FUNCTIONS

No specific list of functions is given since there is no limit on the number of possible functions. Functions must be single-valued.

A. GENERAL FORM:

Three or more alphabetic or numeric characters (beginning with an alphabetic character) followed by a left parenthesis followed by 1st argument followed by a right parenthesis or by a comma followed by 2nd argument followed by a right parenthesis or by a comma followed by 3rd argument, etc.

B. EXAMPLES:

i) sin(a)

ii) sqrt(a+b) : means $\sqrt{a+b}$

iii) factl(m+n) : means (m+n) !

iv) sqrt(sin(axx2)) : means $\sqrt{\sin(a^2)}$

v) max(a, b, c, d, e) : means select the largest of the quantities a, b, c, d, e.

5. EXPRESSIONS

A. INFORMAL DESCRIPTION:

Any sequence of variables and functions separated by operation symbols and parentheses which forms a meaningful mathematical expression in the normal way. Note that every adjacent pair of variables or functions must be separated by an operation symbol.

B. FORMAL DESCRIPTION:

By repeated use of the following rules, all legal expressions may be derived and all expressions so derived are legal provided they have less than 750 characters.

i) Any constant or variable is an expression.

ii) If E is an expression not of the form +F or -F, then +E and -E are expressions.

iii) If xxx denotes a function of n arguments, and if E_1, $E_2 \ldots E_n$ are expressions, then in general xxx $(E_1, E_2, \ldots, E_n)$ is an expression. Although functions may have this general form, certain functions will place restrictions on the form of permissible arguments.

iv) If E is an expression, so is (E)

v) If E and F are expressions where F is not of the form +G or -G and o is one of the permissible binary operations, then EoF is an expression.

vi) If E and F are expressions, so is E××F

C. EXAMPLES:

i) a/b/c Note that this is equivalent to (a/b)/c

ii) a/b×c Note that this is equivalent to (a/b)×c

iii) a/(b+c)×d Note that this is equivalent to (a/(b+c))×d

iv) a+sin(b×c /(d+(e+(f+g)))×cos(b))×b××2 Note the use of redundant parentheses in this example to indicate the desired order of computation.

v) 2.×r Note that the decimal point is used to denote that 2 is retained in floating point form.

vi) 1.53×10××-14 denotes 1.53×10^{-14}

vii) m/n

When the order of binary operations in an expression is not completely specified by parentheses, the order of precedence is understood to be as follows:

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

For example, the expression

a+b/c+d××2×f-g

will be taken to mean

(a)+(b/c)+(d^2×f)-(g)

Multiplication and division will have no fixed relationship of precedence, except in the sense of example ii above.

D. FIXED POINT EXPRESSIONS, FLOATING POINT EXPRESSIONS, MIXED EXPRESSIONS

i) Fixed point expressions are expressions containing only fixed point constants and variables.

a) All fixed point expressions will be evaluated by fixed point integer arithmetic. Thus, the value of i+m/n will be i+(the integral part (unrounded) of m÷n).

ii) Floating point expressions are expressions containing only floating point constants and variables with the exception of fixed point arguments of certain functions and fixed point variables or constants following the operation **.

a) Floating point expressions will be evaluated using floating point arithmetic. It may be necessary in certain cases to use redundant parentheses to indicate a particular sequence in which the operations should be performed in order to avoid obtaining intermediate results in the evaluation of the expression which might lie outside of the range 10^{-38}, 10^{38}.

iii) A mixed expression is any expression not belonging to one of the two above categories.

a) The type of arithmetic employed in evaluating a mixed expression is described below in the section headed: ARITHMETIC FORMULAS.

E. VERIFICATION OF CORRECT USE OF PARENTHESES

In complicated expressions involving the use of many parentheses, it is very easy to omit closing some parentheses. Therefore, in such cases, it is suggested that the programmer use the following procedure to make sure that the parentheses in an expression indicate the sequence of operations he desires. Working from left to right, number each parenthesis, right or left, as follows: Number the first parenthesis "1", label each left parenthesis with an integer one larger than the number of the parenthesis immediately to the left of it. Label each right parenthesis with an integer one less than the number of the parenthesis immediately to the left of it. Having done this, the mate of any left parenthesis labeled "n" will be the first right parenthesis to the right of it labeled n-1. It should be noted that these numbers are not part of the FORTRAN language and should not be entered in the expression.

6. SUBSCRIPTS AND SUBSCRIPT EXPRESSIONS:

Subscripts and subscript expressions described below must have non-negative, non-zero values at all times.

A. SUBSCRIPTS

A subscript is any fixed point variable or constant.

B. A SUBSCRIPT EXPRESSION

A subscript expression is a fixed point expression of not more than 3 terms where all but one term is a single fixed point variable or constant and one term may be a product of two subscripts. All but one of the variables in a subscript expression must be designated as relative constants (see section, RELATIVE CONSTANTS, under SPECIFICATION SENTENCES). Parentheses are not permitted in subscript expressions.

i) Examples:

where j and n are relative constants:

a) i+1
b) i+j
c) n×i+j
d) 2×n-i
e) 100-n×j

7. SUBSCRIPTED VARIABLES

A. A subscripted variable is a variable (fixed point or floating point followed by a left parenthesis followed by one, two, or three subscripts or subscript expressions (where each subscript or subscript expression except the last is followed by a comma) all followed by a right parenthesis.

B. Each subscript or the elements of each subscript expression may be subscripted fixed point variables.

C. Subscripted variables may be used in an expression in the same manner as ordinary variables.

D. No subscript or element of a subscript expression which is a subscript of a fixed point variable which, in turn, is the subscript of another variable may have a subscript.

E. EXAMPLES:

i) a(i)

ii) a(i, j)

iii) a(i, j, k)

iv) a(3×i+n, m) : means $a_{3\times i+n,\ m}$

v) i(j)

vi) n(i, j)

vii) a(i(j)) : means a_{i_j}

viii) i(j(k))

ix) a(n(i, j), m(k, l)) : means $a_{n_{i,j}, m_{k,l}}$

x) a(3×i(j)+2, k)

xi) a(1)

xii) a(i, i+1, i)

xiii) a(1, j)

xiv) a(5, 7, 15)

8. ARITHMETIC FORMULAS

A. An arithmetic formula is a variable (subscripted, or not), followed by an equals sign, followed by an expression.

B. It should be noted that the equals sign in an arithmetic formula has the significance of "replace". In effect, therefore, the meaning of an arithmetic formula is as follows: Evaluate the expression on the right and substitute this value as the value of the variable on the left.

C. If the variable on the left of an arithmetic formula is a fixed point variable and the expression on the right is a mixed expression, then the value of each floating point constant and variable in the mixed expression, with the possible exception of arguments of certain functions, will be truncated to integers. The value of any floating point valued function will also be converted to an integer and the entire expression will be evaluated by fixed point integer arithmetic. Similarly, if the variable on the left of an arithmetic formula is a floating point variable, and the expression on the right is a mixed expression, the values of fixed point constants and variables will be represented as floating point numbers and the expression will be evaluated with floating point arithmetic.

D. If the variable on the left of an arithmetic formula is a fixed point variable and the expression on the right is a floating point expression, the expression will be evaluated with floating point arithmetic and the result truncated to an integer. Similarly, if the variable on the left of an arithmetic formula is a floating point variable and the expression on the right is a fixed point expression, the expression

will be evaluated using integer arithmetic and the result substituted in floating point form for the value of the variable on the left.

E. EXAMPLES:

i) a(i,j)=sqrt(b(i)××2+sin(c(j)×(g+cos(h/(p+q/(r+s)))))) means:

$$a_{i,j}=\sqrt{b_i^2+\sin(c_j\times(g+\cos(h/(p+q/(r+s)))))}$$

ii) a(i,j) = i×j

iii) i=i+15 means : increase value of i by 15 or $i^{(n+1)}=i^{(n)}+15$

iv) a=b

v) n(i)=a(i)+b(i)×17.3

vi) x(i)=b××i

vii) a(i)=a(i)+5.1×sum (j, 1, 20, b(i, j)×c(j)) This formula means increase the value of a_i by the following quantity:

$$5.1\times\sum_{j=1}^{20} b_{i,j}\times c_j$$

viii) a=a+i

ix) i=a×b+n/(m+c)

9. FORMULA NUMBERS

Each FORTRAN formula may have an integer associated with it called the formula number. If a formula has a formula number, the formula number is written to the left of the formula. The formula number must be less than 100,000. If a formula is to be referred to by a control formula as described below, it must be assigned a formula number which is different from the formula number of every other formula. With this exception, the choice of formula number for a formula is completely arbitrary.

A. EXAMPLE

i) 12 a=b

10. CONTROL FORMULAS

A sequence of arithmetic formulas indicate that the operations implied by the first formula should be carried out and then the operations indicated by the second one, etc. Certain formulas called control formulas are provided to alter this sequence of operations in various ways.

In giving the general form of the control formulas below, lower case letters and various symbols such as comma, equals sign and parentheses will be given in the way which they must appear in the particular formula. Capital letters will be used to represent a class of symbols which may appear at a given point in a formula. Square brackets are used to enclose symbols which may optionally appear in the formula.

A. DO-FORMULAS

i) Informal Description

Do-formulas specify a sequence of formulas to be repeated a number of times for different values of a specified subscript and the formula to be executed next after the required number of repetitions. Thus the formula:

do 10, 14, 50 i=4, 20, 2

will cause the sequence of formulas beginning with the formula numbered 10 and ending with the formula numbered 14 to be executed 9 times, the first time with i=4, the second time with i=6, the third time with i=8, etc. and the last time with i=20. Formula 50 will be executed after formula 14 when i=20. Thus the first number after the equals sign is the initial value of the subscript, the next number the final value or upper bound for the subscript, and the third number is the increment to be applied each time. The increment need not be given when it is 1. Furthermore, since it frequently happens that a do-formula immediately precedes the sequence of formulas to be repeated and that the formula to be executed after the proper number of repetitions immediately follows the repeated sequence, it is not necessary in such a case to specify the first formula of the sequence or the formula to be executed after the appropriate repetitions of the sequence. Thus the formula:

do 17 i= 1, 20

causes the formulas immediately following itself up to and including the formula numbered 17 to be repeated in sequence 20 times for i=1, i=2, ... i=20, after which the formula following the formula numbered 17 will be executed.

ii) Formal Description

General form:

do F, F, F S= N, N [, N]

or:

do F S= N, N [, N]

where:

F is a formula number
S is a subscript
N is a fixed point constant, a subscript, or a subscript expression.

iii) Range of a Do-Formula

The range of do-formula A which specifies one formula number is the sequence of formulas immediately following do-formula A up to and including the formula whose number is specified except those formulas in the ranges of do formulas which are in the range of do-formula A. The range of a do-formula B, which specifies three formula numbers, is the sequence of formulas beginning with the formula having the first formula number specified and ending with the formula having the second formula number specified except those formulas which are in the ranges of do formulas in the range of do-formula B.

a) Example

```
1  do 4   j=1, 10
2  do 3   i=1, 10
3  a(i, j)  =i*j
4  b(j)     =sum(i, 1, 10, a(i, j))
```

In the above example, the range of do-formula number 1 includes formula 2 and formula 4 only.

iv) Extended Range of a Do-formula

The extended range of a do-formula A comprises the formulas in the range of do-formula A plus the formulas in the ranges of do formulas in the range of do-formula A.

v) Control Formulas in the Range of a Do-Formula

If a control formula is in the range of a do-formula A and refers control to do-formula A, the next formula to be executed after the control formula will be the first formula in the range of do-formula A after the subscript specified by do-formula A has been incremented once.

vi) Execution of a Do-Formula

The execution of a do-formula A consists of the following steps:

a) Begin execution of the sequence of formulas in the range of do-formula A.

b) If the last formula in the range of do-formula A or a control formula referring control to do-formula A is encountered before a control formula referring to a formula not in the range of do-formula A, increment the specified subscript by the appropriate increment and if the resulting value is not larger than the upper bound specified for the subscript, begin step A again, if this value is larger than the specified upper bound, execute the formula having the third formula number specified in do-formula A. If do-formula A specifies only one formula number, execute the formula following the last formula in the range of do-formula A. The execution of a do-formula is considered complete only when the formulas in its range have been repeated the appropriate number of times or when a control formula in the range of the do-formula is encountered which refers to a formula not in the range of the do-formula.

vii) Restrictions on the Range of a Do-Formula

a) The third formula number specified by a do-formula A may not refer to a formula in the range of do-formula B unless do-formula A is itself in the range of do-formula B. A similar restriction applies to formula numbers specified by if-formulas and go to-formulas described below.

b) If do-formula A and do-formula B are such that neither lies in the range of the other and if S is the sequence of formulas comprising the range of do-formula A and if S' is the sequence of formulas comprising the range of do-formula B, then either S must be wholly included in S' or S' must be wholly included in S, if S and S' have any formula in common.

B. IF-FORMULAS

i) Informal Description

If-formulas enable one to state an inequality or equality condition and indicate that one formula should be executed next if the condition is satisfied and to indicate a second formula to be executed next if the condition is not satisfied.

ii) Formal Description

General Form:

If (N S N)F, F

Where:

N may be a single floating point variable or constant or a subscript or a subscript expression.
S may be one of the following symbols:

=
>
>=

F is a formula number.

Thus the symbols within the parentheses indicate an equality or inequality. The first formula number indicates the formula to be executed next if the equality or inequality is satisfied and the second formula number indicates the formula to be executed next if the equality or inequality is not satisfied.

iii) Example

If (n×i>= k+1)3, 9

This formula means "If n×i $\geq$ k+1, execute formula 3 next, otherwise execute formula 9 next".

C. GO TO-FORMULAS

i) General Form

Go to F

where F is a formula number indicating the formula to be executed next.

D. STOP-FORMULAS

i) General Form

Stop

When such a formula is executed, the machine will stop. If the start button is depressed following execution of a stop-formula, the formula following will be executed next.

E. RELABEL-FORMULAS

i) Informal Description

Relabel-formulas enable the programmer to cyclically relabel the elements in a vector, the rows or columns of a matrix, the rows or columns or planes of a three dimensional array. For example in a 4 by 4 matrix, he may wish to operate on rows 2 and 3, record rows 1 and 2 on auxiliary storage, replace rows 1 and 2 by new information, operate on rows 4 and 1, record rows 3 and 4 and replace them by new information, operate on rows 2 and 3, etc. If, after replacing the information in rows 1 and 2 with new information, he relabels the rows as follows he can then use the same formulas to carry out the second set of operations that he used to carry out the first:

Old row 3 becomes new row 1
Old row 4 becomes new row 2
Old row 1 becomes new row 3
Old row 2 becomes new row 4

Using this type of relabeling, the sequence of operations indicated above becomes simply the repetition of the following steps:

Operate on rows 2 and 3
Record rows 1 and 2
Replace rows 1 and 2 with new information
Relabel

ii) Formal Description

General Form:

Relabel V

where V may be any subscripted variable all but one of whose subscripts is the integer 1 and whose remaining subscript is either a constant or a single fixed point variable. The subscript which is not 1 indicates which element, row or column, row, column or plane is to become the new first element, first row or first column, first row or first column or first plane.

iii) Examples:

a) "Relabel a(3)" has the following significance where a is a vector of 7 elements. A reference to a(1) after the execution of this formula is equivalent to a reference to a(3) before the execution of this formula. Similarly, the

new a(2) corresponds to the old a(4), the new a(3) to the old a(5), the new a(4) to the old a(6), the new a(5) to the old a(7), the new a(6) to the old a(1), the new a(7) to the old a(2).

b) "Relabel a(1, n, 1)" has the following meaning where "a" is a 3 by 4 by 5 array and where n has the value 3, the old a(i, 3, j) become the new a(i, 1, j) for all values of i and j and finally the old a(i, 2, j) become the new a(i, 4, j) for all values of i and j.

INPUT-OUTPUT FORMULAS

Input-output formulas enable the programmer to specify that information should be brought into the 704 from cards or input tapes or information should be printed or punched or written on output tapes. Since the number of variables which may be referred to at any moment in a calculation is limited by the extent of high speed storage, it may be necessary to record the values of certain variables in auxiliary storage and at other times to assign new values to certain variables corresponding to information in auxiliary storage. Input-output formulas are provided for this purpose also. Capital letters appearing in the formula descriptions below will again be used to indicate the class of symbols which may appear in the corresponding position in the formula.

A. DESCRIPTION OF SEQUENCE OF AN ORDERED ARRAY

In specifying that the elements of a 1, 2, or 3 dimensional array of data should be recorded in auxiliary storage or in specifying that the elements of a 1, 2, or 3 dimensional array should be assigned values corresponding to certain quantities in auxiliary storage, it is necessary that a certain sequence of the elements in the array be either understood or specified. This sequence is that in which elements will be recorded or brought from auxiliary storage. If no sequence is specified, the sequence will be understood to be a_1, $a_2 \cdots a_n$ in the case of vectors or a_{11}, $a_{21} \cdots a_{n1}$, a_{12}, $a_{22}, \cdots a_{mn}$ in the case of 2 dimensional arrays or a_{111}, $a_{211} \cdots a_{n11}$, a_{121}, $a_{221}, \cdots a_{n21} \cdots a_{nm1}$, a_{112}, $a_{212} \cdots a_{nmk}$ in the case of three dimensional arrays.

When no sequence is specified for a given array which is to be recorded on or read from auxiliary storage, it will be understood that the entire array is to be recorded on or read from auxiliary storage.

i) Informal Description

A specification of sequence for a one dimensional array a(i) might be "i=1, 7". This indicates the sequence $a_1, a_2, \ldots a_7$.

A specification of sequence for a two dimensional array a(i, j) might be j=4, 8, i=2, 10, 2. This indicates the sequence $a_{2,4}$, $a_{4,4}$, $a_{6,4}$, $\ldots a_{10,4}$, $a_{2,5}$, $a_{4,5}$, $a_{6,5}$, $\ldots a_{10,5}$, $\ldots$ $a_{10,8}$. Note that a third quantity specified after the range of a given subscript indicates an increment and if a third quantity is not specified the increment is taken to be one.

In general, the subscript specified first in a specification of sequence is varied least frequently, the subscript specified second is varied more frequently and the third subscript specified is varied most frequently. If a specification of sequence is given, each subscript of the array with its appropriate range and possible increment must be listed in the appropriate order.

ii) Formal Description

A description of sequence for a one dimensional array has the following form:

S= N, N[, N]

A description of sequence for a two dimensional array has the following form:

S=N, N[, N], S=N, N[, N]

A description of sequence for a three dimensional array has the following form:

S=N, N[, N], S=N, N[, N], S=N, N[, N]

where:

S may be a subscript which appears as a subscript of the array whose sequence is being specified
N may be a subscript or subscript expression

Note that square brackets enclose symbols which are optional.

B. LIST OF QUANTITIES

i) Formal Description

A list of quantities has the following form:

V [, V, V, ...]

where V may be:

1. a single variable or constant
2. a subscripted variable
3. a left parenthesis followed by one or more subscripted arrays (each except the last followed by a comma) followed by a specification of sequence followed by a right parenthesis.

ii) Examples:

a) a, (b(i, j), c(i, j) j=1, 2, i=1, 3), d, e

The above list of quantities specifies the following sequence:

a, b(1, 1), c(1, 1), b(2, 1), c(2, 1), b(3, 1), c(3, 1), b(1, 2), c(1, 2), b(2, 2), c(2, 2), b(3, 2), c(3, 2), d, e

b) a, (b(1, i), c(i), d(i, 1) i=1, 3), e(1, 1)

The above list of quantities specifies the following sequence:

a, b(1, 1), c(1), d(1, 1), b(1, 2), c(2), d(2, 1), b(1, 3), c(3), d(3, 1), e(1, 1)

C. CARD READING FORMULAS

i) General Form:

read L

where L may be list of quantities. However, none of the quantities in the list may be constants.

a) Example

read n, (a(i, j) j=1, 20, i=5, 10), b(i)

This formula indicates that the sequence of variables n, $a_{i,j}$'s and b_i's should be assigned the sequence of values coming from the card reader in a one-to-one fashion.

D. CARD PUNCHING FORMULAS

i) General Form

punch L

where L is a list of quantities

ii) Card Punching Formulas indicate a sequence of quantities to be punched on cards.

E. PRINT FORMULAS

i) General Form:

print L

where L is a list of quantities

F. TAPE READING FORMULAS

i) General Form:

read tape (N) L

or:

read input tape (N) L

where N is a tape number or fixed point variable and L is a list of quantities and no quantity is a constant.

G. TAPE WRITING FORMULAS

i) General Form:

write tape (N) L

or:

write output tape (N) L

where N is a tape number or a fixed point variable and L is a list of quantities.

H. ADDITIONAL FORMULAS FOR MANIPULATING TAPE

i) General Form:

end file (N)
rewind (N)
backspace (N)

where N may be a tape number or a fixed point variable.

I. DRUM READING FORMULAS

i) General Form:

read drum (N, M) L

where N is a drum number or fixed point variable and M is a drum location or fixed point variable and L is a list of quantities. The drum location is an integer between 1 and 2048. The effect of this formula is to cause the quantities on the given drum beginning at the given drum location and in the consecutively numbered drum locations following to become the values of the quantities specified in the list of quantities in high speed storage.

J. DRUM WRITING FORMULAS

i) General Form:

write drum (N, M) L

where N is a drum number or fixed point variable and M is a drum location or fixed point variable and L is a list of quantities.

K. RESTRICTION ON LISTS OF QUANTITIES IN DRUM READING AND WRITING FORMULAS

If a specification of sequence is given with any array specified in a list of quantities in a drum reading or writing formula, the subscripts appearing in such a specification of sequence must appear in the opposite order from the subscripts associated with the array and only the last subscript may have an arbitrary range. Subscripts other than the last must have ranges specified beginning with 1 and ending with the maximum value possible for that subscript. None of the subscripts in the specification of sequence may have increments other than 1. Only one array may appear with a specification of sequence in a single pair of parentheses.

i) Examples:

The following list of quantities may correctly be specified by drum reading or writing formulas:

a, (b(i, j, k) i=7, 10 j=1, 50, k=1, 50)

where 50 is the maximum possible value for i and j. The following list of quantities may not be correctly specified by a drum reading or writing formula:

(a(i, j, k) k=1, 50, j=3, 20, i=1, 50)

(a(i, j, k) j=1, 50, i= 1, 50, k=1, 50)

12. SPECIFICATION SENTENCES

In addition to the problem formulation in terms of FORTRAN formulas, certain additional information is either necessary or desirable to enable the FORTRAN system to produce an efficient program. Specification sentences provide the means of supplying such information to the FORTRAN system.

A. DIMENSION SENTENCES

The maximum possible dimensions of each 1, 2 or 3 dimensional array referred to in any formula in the problem formulation must be specifically given. Thus if a(i, j, k) is specified as a $5 \times 10 \times 20$ array, then at no time when a reference to a(i, j, k) is made should i exceed 5, or j exceed 10, or k exceed 20. Having so specified a(i, j, k), it is nevertheless possible to regard a(i, j, k) as representing a $4 \times 4 \times 4$ array in a particular instance. This type of situation will obtain where the dimensions of an array are input parameters.

i) General Form:

Dimension V[, V, V, . . .]

where V is a subscripted variable whose subscripts are fixed point constants. Thus a(10, 11, 12) occurring in a dimension sentence indicates that the maximum dimensions of the array a are $10 \times 11 \times 12$.

Note that dimension sentences specifying the dimensions of all arrays appearing in a problem formulation must be given.

B. EQUIVALENCE SENTENCES

In certain cases, it may be possible for the FORTRAN system to assign the same storage location to several variables. For the purpose of defining when this is possible, we shall say that a variable appears in a formula in a type 1 position if the execution of the formula could not possibly alter the value of the variable and we shall say that a variable appears in a formula in a type 2 position if the execution of the formula could result in changing the value of the variable. Thus a variable appears in a type 1

position if:

1) it is on the right side of an arithmetic formula.

2) it is a subcript of a variable on the left side of an arithmetic formula.

3) it appears in an output formula.

4) it appears in a do-formula but not as the subscript to be varied.

5) it appears in an "if" or a "go to" formula.

And similarly, a variable appears in a type 2 position if:

1) it is the variable on the left side of an arithmetic formula.

2) it appears in an input formula.

3) it appears as the subscript to be varied in a do-formula.

Thus a set of variables may be assigned the same storage location if for any two variables a and b in the set, a type 2 appearance of a followed by a type 1 appearance of b always means there is an intervening type 2 appearance of b, where the order of appearance is the order of execution of the formulas. Under the same conditions it is also possible to allot overlapping storage space to the elements of two different arrays. Equivalence sentences specify sets of variables and arrays such that all variables or arrays in the same set may be assigned the same storage area.

i) General Form:

Equivalence (V, V[, V, V...])[, (V, V[, V, V, ...]), ...]

where V is a variable symbol. The variable symbol may be either one associated with a simple variable or one associated with an array. Thus, to indicate that the variable a, the array b(i, j) and the array c(i, j, k) can be assigned overlapping storage space, one includes in a dimension sentence the set (a, b, c). If the product of the maximum dimensions of c(i, j, k) is greater than the product of the maximum dimensions of b(i, j), the inclusion of the above set in an equivalence sentence means that the storage space allotted to b(i, j) will be included in the storage space allotted to c(i, j, k) and that the storage space allotted to a will be included in that allotted to b(i, j).

C. FREQUENCY SENTENCES

Frequency sentences enable the programmer to provide the FORTRAN system with information concerning estimates of the frequency with which certain portions of the program will be executed. Thus the programmer may indicate that he expects the condition specified by an if-formula to be satisfied 10,000 times and that the condition will not be satisfied 400 times during the execution of the program. If the if-formula has formula number 3, this estimate would be stated in a frequency sentence as follows:

(3, 10000, 400)

Similarly, if a do-formula has a variable range for the subscript that is to be varied, the programmer may specify that on the average he expects the do-formula to call for, say, 200 repetitions. If the do-formula has the formula number 17, the programmer would indicate this estimate as follows:

(17, 200)

as part of a frequency sentence, and finally if a go to-formula has a fixed point variable included in it, the programmer may give estimates of the frequency with which the fixed point variable will assume the various possible values. If the go to-formula has the formula number 2 and reads "go to n" and if n may take on the values 14, 15 and 16, then the estimate (2, 13, 100, 14, 10, 15, 1000) indicates that he expects n to take on the value 13, 100 times, the value 14, 10 times and the value 14, 1000 times. The above three types of estimates, one for if-formulas, one for do-formulas and one for go to-formulas are the only permissible types of estimates which can appear in a frequency sentence.

i) General Form:

Frequency E[, E, E...]

where E is an estimate of any of the three types described above.

D. RELATIVE CONSTANT SENTENCES

In certain cases it will be possible for the FORTRAN system to produce a more efficient program for a problem if it is supplied information specifying those fixed point variables whose values change very infrequently on a relative basis. Relative constant sentences offer the programmer the opportunity of providing this information to the FORTRAN system.

i) General Form:

Relative constants N[, N, N, ...]

where N is a fixed point variable. Thus the sentence,

Relative constants i, n

where i is a single fixed point variable and n(j) is a fixed point vector, indicate that the value of i and the values of n(1), n(2), ... change very infrequently.

13. PROBLEM PREPARATION

Problem preparation for automatic coding by the FORTRAN system consists of the following steps:

A. PROGRAMMING

The formulas specifying the problem are written in the form given above. Note that the exact symbol used for writing, say, multiply, is arbitrary provided the proper Hollerith code for multiply is punched in the formula cards. In addition to the formulas specifying the problem, dimension sentences giving the maximum dimensions of all arrays in the problem and possibly other specification sentences must be written in the form described above.

B. DATA PREPARATION

Input data, referred to by card reading formulas or read input tape formulas in the problem, should be written on standard forms suitable for key-punching in standard card forms associated with card reading formulas and read input tape formulas.

C. CHECK OF DATA STORAGE SPACE REQUIRED

The data storage required for a given problem if no equivalence sentences are specified, is computed as the number of single variables and constants plus the sum of the products of the maximum dimensions of each array referred to in the problem. In computing data storage space, it is only necessary to count one space for a sequence of constants separated by arithmetic operations. If equivalence formulas are given, the amount of storage space required is the number of constants plus the number of single variables not appearing in an equivalence sentence plus the sum of the products of the maximum dimensions of arrays not appearing in equivalence sentences plus the sum of the products of the maximum dimensions of the largest array appearing in each set in an equivalence sentence plus the number of the sets, containing only single variables, which appear in equivalence sentences. The data storage space required for a program must be less than a certain amount which will depend on the total high speed storage space of the machine on which the problem is to be

run. The amount of storage space that would be available in any machine with 4096 words will be at least 3,000 units. Problems must be planned in such a way that the data storage space required is less than the appropriate amount.

D. KEY PUNCHING

The formulas specifying the problem are punched on cards in the exact form in which they are written. There will be space for approximately 65 characters on each card. There will be a space on each card for a formula number which will be left blank if the formula has no number assigned. Large formulas may extend over many cards. An indication on each card will indicate whether or not the information on the card is a continuation of a formula on a preceding card. Spaces (denoted by blank columns on a card) are ignored by the FORTRAN system. This means that, if desired, the key puncher can space between symbols in exactly the way they are written, or not, without disturbing the meaning of the formula. Specification sentences are also punched in a manner similar to that of formulas. Note again that dimension sentences must be punched for any problem making reference to arrays. Data cards are punched in the appropriate form to be accepted by card reading formulas or to prepare tapes which are to be read by input tape reading formulas.

E. PREPARATION OF CARD DECKS AND INPUT TAPES

The FORTRAN system offers two options:

1) punching of binary program deck for the problem or preparation of similar program tape and printing of program.

2) immediate execution of problem.

If the user of the system selects option 1, he should prepare a deck of cards in the following order: all specification sentence cards followed by formula cards in the correct order followed by a specially punched card indicating the end of the FORTRAN formulas for the problem. If the user selects option 2, he should prepare the same deck as above and, in addition, a deck of data cards for each input tape employed in the problem and for the card reader, if employed.
Having prepared the above decks of cards, he should then prepare the appropriate input tapes, if any, on auxiliary card to tape equipment. He may further elect to enter the FORTRAN formula deck directly from the 704 card reader or to prepare an input tape from this formula deck and enter the formulas in the 704 from this input tape.

F. AUTOMATIC PROBLEM CODING OF PROBLEM BY THE FORTRAN SYSTEM

If the user has selected option 1 (to obtain the binary cards representing his program or a tape representing his program), he should simply load the FORTRAN system from its tape and place the deck of FORTRAN formulas in the card reader or the corresponding tape on a tape unit. He should then set a sense switch indicating that he has elected option 1. He should set another switch indicating that the program should either be punched on binary cards or that it should be written on magnetic tape. Pressing the start button will then cause the FORTRAN system to write the required program, check it, and either punch it on binary cards or write it on tape and prepare an output tape which can be used to print the program on auxiliary tape-to-printer equipment. (Installations not having auxiliary tape-to-printer devices may arrange to have the program printed directly).

If the user selects option 2 (immediate execution), he should put the appropriate input tapes on the appropriate tape units and the appropriate card deck, if any, in the card reader. When the FORTRAN system is loaded and the appropriate switches set and the start button pressed, the FORTRAN system will write the required program and cause its execution to begin immediately thereafter.

14. FUTURE ADDITIONS TO THE FORTRAN SYSTEM

The language of FORTRAN formulas and sentences described above is to be regarded only as the basic FORTRAN language. The FORTRAN system will be constructed in a manner to make the addition of new formulas, new sentences and new functions as easy as possible. It is expected that the FORTRAN language will be continually enriched by such additions to make it more economical, more convenient and more efficient. Some of the possibilities for future additions to FORTRAN are listed below:

A. A VARIETY OF NEW INPUT-OUTPUT FORMULAS WHICH WOULD ENABLE THE PROGRAMMER TO SPECIFY VARIOUS FORMATS FOR CARDS, PRINTING, INPUT TAPES AND OUTPUT TAPES

B. POSSIBLE ADDITIONAL CONTROL FORMULAS

i) Begin Complex Arithmetic

ii) End Complex Arithmetic

iii) Begin Double Precision Arithmetic

iv) End Double Precision Arithmetic

v) Begin Matrix Arithmetic

vi) End Matrix Arithmetic

vii) Sort the Vectors on Tape Number N using the kth element of each vector as indicative information

viii) Solve the following N simultaneous equations

ix) Solve the following system of ordinary first order differential equations

x) Find the vector x(i) which maximizes the linear function f and satisfies the following linear inequalities

C. POSSIBLE ADDITIONAL FUNCTIONS

There will, of course, eventually be a large list of arithmetic functions available to the FORTRAN system. The following items indicate certain slightly unusual types of functions.

i) General Function:

Such a function would enable a programmer to avoid rewriting a set of formulas describing a function peculiar to his problem but which occurs frequently in his problem. Such a function would enable the programmer to specify the formula numbers of the formulas describing his function and the arguments to be used in a given instance. The value of the function would be the value of the right hand expression of the last specified formula in the function description, having substituted the specified arguments for the original arguments appearing in the formula description of the function.

ii) Definite Integral

Such a function would enable the programmer to specify the independent variable, the limits of integration and the expression to be integrated.

iii) Summation

This function would enable the programmer to specify the index of summation, the limits of summation and the expression to be summed.

iv) Table Lookup

This function would enable the programmer to specify the table number and the argument (or arguments if the particular function was bivariate).

15. DESIRABLE TECHNIQUES TO USE IN PROGRAMMING A PROBLEM TO BE CODED BY FORTRAN

Although the FORTRAN system is being designed to produce a correct program from a correct meaningful set of FORTRAN formulas and although the programmer will invariably discover many possible formulations of the same problem, the use of certain techniques will, of course, result in more efficient 704 programs.

A. REPRESENTATION OF COMPLICATED EXPRESSIONS

In translating a single arithmetic formula, the FORTRAN system will permute the operations indicated in the expression on the right wherever this is permissible in order to minimize the number of STORE instructions which will be required in the resulting 704 program. Thus a × b × c /d/e would be permuted to a/d×b/e×c. However, any order of computation which is specified by use of parenthesis will be followed. Furthermore, if certain portions of an expression are identical to certain other portions of the same expression (all in the same formula), the system will recognize this and avoid duplicate calculations. To enable the FORTRAN system to recognize duplications of various subexpressions in an expression on the right side of an arithmetic formula, it will only be necessary to enclose duplicated subexpressions where they appear as part of a term in the expression. Where duplicated subexpressions occur as complete terms, it will not be necessary to enclose the term in parentheses. Furthermore, if the duplicated subexpression is a function which appears in several places with the same argument, it will not be necessary to enclose the function in parentheses even though it may be a portion of a term. Thus the following expression:

a × b × c ×(a×b×c+e×cos(a))/(a×b×e+f×cos(a)) +sqrt (a×b×e+f×cos(a))

may be written in the following form to avoid duplicate calculations:

((a×b)×c)×((a×b)×c+e×cos(a))/((a×b)×e+f×cos(a)) + sqrt((a×b)×e+f×cos(a))

In general then, if a complicated expression is involved in a problem, it is best not to introduce new dependent variables to represent portions of the complicated expression and then to represent the complicated expression as an expression involving the new dependent variables. Adherence to this principle allows the FORTRAN system to carry out the maximum amount of optimization.

B. FORMATION OF LOOPS

In specifying operations on sequential items in ordered arrays, it is best to use do-formulas wherever possible since such formulas present the control information which the system needs in forming loops in a consolidated form. The use of formulas such as

$$i = i + 1$$

$$\text{if } (i>n)\ n1,\ n2$$

may result in some unnecessary instructions in the resulting program if such instructions are used to form loops which could be otherwise formed by the use of do-formulas.

C. DEBUGGING

No special provisions have been included in the FORTRAN system for locating errors in formulas. After some experience has been gained in the use of the system, it will be possible to write a program to locate the most common of the frequently occurring errors in a formula program. Since FORTRAN formulas are fairly readable, it should be possible to check their correctness by independently recreating the specifications for the problem from its FORTRAN formulation. In this way it should be possible to write correct formula programs from which the FORTRAN system will of course produce correct 704 programs.

D. PROGRAM CHECKS

There are no automatic provisions in the FORTRAN system for including checks on correct machine operation in an automatically coded program unless the checks are provided for in the original formula program. Since FORTRAN-written 704 programs will be written in accordance with certain uniform principles, it should be relatively simple for an operator experienced with FORTRAN-written programs to determine what has happened in a program after a machine failure.