June 7, 1957

FORTRAN INTRODUCTORY PROGRAMMER'S MANUAL

SECTION III

Programming Research Department

International Business Machines Corporation

590 Madison Avenue

New York 22, New York

SECTION III

Introduction

Several of the statements introduced in Section II offered a convenient method for handling one dimensional arrays in a systematic, repetitive manner. However, no provision was made for handling two and three dimensional arrays. This provision greatly facilitates the solving of many engineering and scientific problems which require matrix manipulations for their solution. This section will describe the extension of subscripting to two and three dimensional arrays and the expansion of the use of the DO statement to handle such arrays. Several other new statements will also be introduced.

The following example of matrix multiplication will serve to illustrate DO "nests" and multiple subscripts.

Example: Given the matrix A with dimensions 10 x 15 and the matrix B with dimensions 15 x 12. To compute any element Cij of the matrix C = AB select the i th row of A and the j th column of B, and sum the products of their corresponding elements: thus

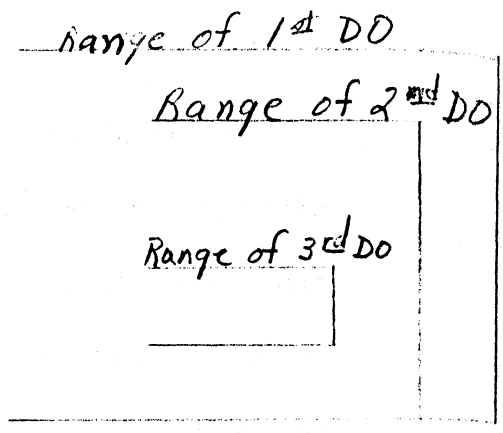
$$Cij = \sum_{k=1}^{15} A_{ik}B_{kj}.$$

The following is a possible FORTRAN program for matrix multiplication.

```
      DIMENSION A(10, 15), B(15, 12), C(10, 12)
    3 FORMAT (5E14.5)
```

```
     READ 3, A, B
 4   DO 30 I = 1, 10
 5   DO 30 J = 1, 12
 6   C(I, J) = 0.0
10   DO 20 K = 1, 15
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 cards, the format being specified by statement 3. Since two dimensional arrays are stored "columwise", the matrices A and B must be keypunched in that order; e.g. A_{11}, A_{21}, A_{31}, A_{41}, ..., $A_{10,15}$. 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 will compute and print 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 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 th

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 and thereby produces all of the rows of the product matrix.

This example illustrates the fact that one or more DO's may appear in the range of a DO statement. This "nesting" of DO's can result in a single statement being the last statement in the range of several DO's. (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 incrementing of indices following the last statement in the range of a DO.

RULE. Upon completing 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 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's containing this last statement 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 entire set 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 V.

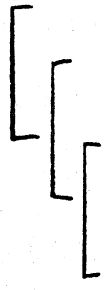
DO "Nests"

There are certain rules which must be observed when using DO's within DO's or DO "nests".

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

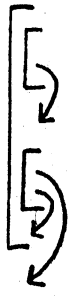
Permitted

Violation of Rule 1



2. No transfer of control by IF-type or GO TO-type statements is permitted into the range of any DO from outside its range since such transfers would not permit the DO loop to be properly indexed.

Permitted



Violation of Rule 2

All of the DO statements presented in the preceding material were written in the form

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

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 m_3 in the DO statement unless it 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 permissable only if that variable has been previously assigned a value (in this case, a value was 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 from the previous page, 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 above order.

"Assigned GO TO"

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.

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. Assuming the data to be on cards, the following program might be used:

```
      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 in the list, i.e. 30, 40 or 50 depending upon the value of N at the time of execution. The first execution of statement 26 causes control to be transferred to statement 30 since statement 5 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" is written as

GO TO N, (n_1, n_2, ... n_m)

where N is a non-subscripted integer variable appearing in a previously executed ASSIGN statement and n_1, n_2, ... n_m are statement numbers. These 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 parentheses; it must always be included.

The statement

ASSIGN 30 TO N

is not equivalent to the arithmetic formula

N = 30

A variable which has been "assigned" can be used only for directing an "assigned GO TO" until the variable, say N, is set equal to an arithmetic expression, at which time N is re-established as an ordinary variable. Likewise, an ordinary variable has no effect on an "assigned GO TO" until this variable has been "assigned".

There is a restriction on the "assigned GO TO" statement when it lies in the range of a DO statement. This restriction requires that the statements to which the "assigned GO TO" may transfer 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" to accomplish the desired branching since there is no restriction on such statement.

"Computed GO TO"

The computed GO TO is similar to the "assigned GO TO" in that both statements establish a many-way fork. They differ in that the "assigned GO TO" requires that the proper branch be chosen by pre-setting or "assigning" while, with the "computed GO TO", the proper branch is determined by the value of a integer variable. The value of this variable may be arrived at by computation; no companion statement (comparable to ASSIGN) is necessary.

Example:

Given: Ai, Bi, Ni, Xi, Yi for I = 1, 10 to compute

$Zi = \sqrt{A_i X_i^2 + B_i Y_i}$ for Ni = 1

$Zi = \sqrt{A_i X_i^2 - B_i Y_i}$ for Ni = 2. A possible FORTRAN program follows.

```
      DIMENSION A(10),B(10),N(10),X(10),Y(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 example, statement 7 transfers control to statement 10 if J = 1 or to statement 20 if J = 2. Since J is set equal to Ni by statement 6, the correct formula for Zi is selected. Statement 6 is necessary since J cannot be a subscripted variable.

As was noted in the example the "computed GO TO" has the form

```
GO TO (5, 10, 15, 20), I
```

where the numbers enclosed in parentheses are statement numbers. Control is transferred to the first statement in the list (in this case, to statement 5) if, at the time of execution, the value of I is one; it is transferred to the second statement in the list (in this case, to statement 10) if the value of I is two; etc. Any number of statement numbers may appear in the list. The value of I may be arrived at in any manner desired (e.g. by an arithmetic statement, as the result of DO indexing) and its value at the time of execution of the "computed GO TO" 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 discussed. In this section Scale Factors, Hollerith fields, and Multiple-line formats will be introduced.

Scale Factors

The use of scale factors allows greater flexibility in an output format. As noted in Section II, the specification

(2E14.4)

might print the following output line

-0.4321E 04 0.5678 E-06

If the specification is written as

(2P2E14.4)

the same output data would print as

-43.2147E 02 56.7839E-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, e.g. the specification

(-1PF10.3, 7PF8.3)

would print the following data

-4321.47 .0000005678

as

-432.147 5.678

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 and followed by the desired English characters (including blanks). In order to print the factors X and Y as

well as 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, there is no comma after a Hollerith field in the format specification.

Multiple-line FORMAT

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 have been necessary, namely

```
10    FORMAT(2PE13.4, F8.2, 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, F8.2, I5/I9, F12.3, F5.2)
```

The slash (/) separates the different line formats. Thus, in this example, lines 1, 3, 5,... have the format (2PE13.4, F8.2, I5) and lines 2, 4, 6, ... have the format (I9, F12.3, F5.2). Each line may have a maximum of 119 characters.

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 IF statement is used in conjunction with the sense switches.

IF(SENSE SWITCH i) n_1, n_2

where i refers to 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 "UP" control is transferred to statement number n_2, if sense switch i is "DOWN" 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 to compute and print

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

The following FORTRAN program is written and compiled (i.e. translated into 704 language).

```
      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 "UP" (which causes the printing of intermediate results). Assume the test case has the following input data

a_1 = -.23456 b_1 = 12.34111 c_1 = +27.86523

Then the first line of output is

42.72019 -15.52412 -30.75996

The hand calculations show that for I = 1

SUM 1 = 42.72019

SUM 2 = -15.52412

SUM 3 = -779.36577

SUM 1 and SUM 2 results agree, 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. The 704 operator is instructed to run the production data with sense switch 1 "DOWN". 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 tapes and/or drums for additional storage. In order to determine whether additional storage will be 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 specified by entries in DIMENSION statements, call this value B. (e.g. A(12, 6) required 72 locations).
3) Use the rules on page 18 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 these numbers for the functions used in the program, call this 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, then 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 then 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 drums and/or binary tapes for intermediate storage, consult the FORTRAN Programmer's Reference Manual for information regarding the necessary statements. The Programmer's Reference Manual also 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 statements appearing add the corresponding

number. (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

Example

Given n points (x_i, y_i) to fit by 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) $S_1 a_0 + S_2 a_1 + \text{---} + S_{m+1} a_m = V_1$

. . . . . .

. . . . . .

.. . . . .

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

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

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

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

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

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

Having computed the S's and V's, the normal equations are solved using the method of elimination which is illustrated by the following solution of the normal equations for a second degree polynomial.

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

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

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

The forward solution is as follows:

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

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

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

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

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

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

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

Steps 1 and 2 are repeated using equations (5) and (6). The resulting equations are

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

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

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) $a_2 = {}^{c_{34}}/_{c_{33}}$ from equation (8)

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

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

The following is a possible FORTRAN program for these calculations. In this program $n \leq 100$ and $m \leq 10$. 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).

```
      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
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 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 added to the correct sum counters. The powers of X_i are multiplied by Y_i and added to the correct V counters. In order to save machine time when running the object program, 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. The augmented matrix is stored (statements 17 - 22), the forward solution is computed (statements 23 - 31), and the backward solution (statements 33 - 40) is computed. The coefficients a_0, a_1, a_2, ... a_m are stored in the A array.

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 numeric 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, i.e. "A" and "a" will both be treated as "A".
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 it is an integer variable.
   c. Any character following the first may be alphabetic or numeric, but not one of the special characters.
   d. The names of all functions appearing in the list of functions as well as these names without their final letter "F" must not be used as variable symbols,

For example, if SINF appears in the list of functions, then neither SINF or SIN can be used as a variable symbol.

5. In indicating a function, the name of the function must agree exactly with the name appearing in the list of functions.

6. The argument of a function is enclosed in parentheses.

7. If a function has more than one argument, the arguments are separated by commas.

8. The left side of an arithmetic formula must always be a variable.

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 only exception is the operation symbol "**", which is regarded as a single symbol.

11. Blank spaces can be 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 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 which causes a transfer to some other statement in the program (a GO TO or an IF statement).

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. 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. No calculation which changes the index or indexing parameters of a DO is permitted within the range of that DO.

28. "Assigned GO TO" statements have a comma between the variable and the left parenthesis.

29. "Computed GO TO" statements have a comma between the right parenthesis and the ~~left~~ variable.

30. An ASSIGN statement must be encountered by the program prior to encountering an "assigned GO TO" statement.

31. The ASSIGN statement is not equivalent to the arithmetic formula.

32. 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, or be completely outside the nest.

33.