April 10, 1957

FORTRAN INTRODUCTORY PROGRAMMER'S MANUAL

SECTION II

Programming Research Department

International Business Machines Corporation

590 Madison Avenue

New York 22, New York

SECTION II

INTRODUCTION

At this point it should be possible to use that part of the FORTRAN language comprising System I to direct the operation of the 704 in the solution of certain problems. However, it may be difficult or impossible to program the solution of some problems using only the six statements described in Section I. In this section the vocabulary of the System I language will be enlarged in order to make it possible to direct the 704 in the solution of many more problems.

One shortcoming of the language of System I is the laborious programming which is necessary to carry out relatively simple repetitive 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 to use the subscript notation of mathematics in System II to make the programming of such problems easier. Thus 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 its specific sense. 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, then the variables 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 permitting arithmetic formulas with subscripted variables, it is necessary to provide for a method of stating that a given set of such formulas should be executed repetitively for certain values of the subscript. The FORTRAN statement which provides this ability is

called a DO statement. Together with arithmetic formulas, DO statements are the most useful statements provided in the FORTRAN language. An example of a "DO" statement, followed by an explanation, appears below.

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

This statement instructs the 704 to "execute all statements which 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 20". Thus, to return to the example of vector addition, the FORTRAN statements necessary to add A(I) and B(I) are

```
      DO  1  I = 1, 3
1     C(I) = A(I)+B(I)
2     . . .
```

When the statement numbered 2 is finally 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

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

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

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

```
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 having been specified as arrays, the list A, B, C in the READ statement will cause all elements of A, then all elements of B, and then 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. "5F 10.4" says "there are 5 Fixed point decimal fields per card or line, each field being 10 columns wide with 4 decimal placed to the right of the decimal point". Hence A, B, C, P, and Q

will be read or printed as (__ + XX.XXXX). Statement 4 says "DO the following statements through statement number 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 number 8." Statement 9 stops the computer. The new method of notation and the use of the DO, FORMAT, DIMENSION and function statements which have been introduced here, together with several other new statements for input and output and tape manipulation, will be presented in the following pages.

Integer Constants and Variables

In Section I, only floating point constants (which must have a decimal point) and 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 a very useful notation and $X_{3.0}$ is redundant and wastes space. Integer constants and variables are more useful in this regard and are available in System II. 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.

Subscripted variables when used in FORTRAN statements, are 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, of course, 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. 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 variable, B.

It should also be emphasized that reference to a subscripted variable whose subscript is an integer variable, i.e. 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", a DO statement, 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 the one above) may be written; such expressions will be evaluated using integer arithmetic rather than floating point arithmetic. Some examples of expressions which are and are not permitted appear below.

Expression	Permissable	Arithmetic Used
A*B*(C**2)	Yes	Floating
2*A	No	------
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 formula is a legitimate one as described above, there are no further restrictions on arithmetic formulas. There are, however, certain pitfalls which may be encountered if formulas 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 and then 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 and it permits the 704 to assign the proper number of storage locations to each subscripted variable.

The 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) had appeared in a FORTRAN program, then a DIMENSION statement mentioning these variables would have to be included. Assuming that the number of elements in ALPHA(I) will never exceed 100, the number in GAMMA(J) will never exceed 25, and in VECTOR(N) will never exceed 12, then the DIMENSION statement would be written as

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

DIMENSION statements are not actually executed. No instructions corresponding to this statement will appear in the translated

program. However, a DIMENSION statement giving the size of each array should 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 appeared in the introduction to this section. The usefulness of such a statement for carrying out repetitive calculations was mentioned briefly then. The standard form for a DO statement in System II 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 statement is "execute the statements 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 following the DO statement and extending through statement N is called the range of the DO statement. In

System II, none of the statements in the range of a DO statement can be another DO statement. However, the range may contain GO TO or IF statements, and these may transfer control out of the range of the DO.

To illustrate the usefulness of this feature as well as giving another example of a DO statement, consider a set of fifty numbers, A(J) and a number B. The problem is to select the smallest value of J, if there is one, for which B = A(J). A program to accomplish this could be written as follows:

```
10   DO  12   J = 1, 50
11   IF (B - A(J)) 12, 20, 12
12   CONTINUE
13   (if control reaches this statement the search has been
      unsuccessful)
;
20   (if control reaches this statement, the desired value
      of J is available for use).
```

It should be noted that control passes to statement 20, out of the range of the DO statement, as soon as the index of the DO, in this case J, reaches a value for which B - A(J) equals zero. Any reference

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 statements in that program.
2. The function name must have four to seven alphabetic or numeric 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 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 will be a floating point quantity unless the name of the function begins with X, in which case the value will be an integer quantity.

An example will serve to show some properties of function statements.

```
 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 permissable 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 it is not an argument. A may be used like any other variable in the problem and its current value will be used each time FIRSTF is evaluated.

Tape Input and Output

Thus far, the only methods that have been mentioned for transferring decimal data into and out of the internal storate of the 704 are

use of the card reader and printer attached to the computer. 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 internal storage, namely use of magnetic tapes. Just as a READ statement directed the 704 to read data from cards, a READ INPUT TAPE statement directs the 704 to read data into 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 will have available two machines which are not connected to the computer. One of these machines will be capable of reading information from punched cards and writing this information on magnetic tape. The other machine will be able to read information written on magnetic tape and print this information. By using the card-to-tape machine, the information contained in a deck of data cards can be written on magnetic tape and read into the 704 by a READ INPUT TAPE statement. The output information written on magnetic tape by a WRITE OUTPUT TAPE statement can be read and 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 (explained on page 18), and 'List' denotes a list of names of quantities to be read or written. Note the resemblance between these statements and the general form for READ and PRINT

READ N, List

PRINT N, List

where, in System I, N was either 1 or 2 and referred to particular FORMAT statements supplied by the computing center.

The Meaning of a List

Examples of lists have already appeared in READ and PRINT

statements although they were not identified as such. A list is a set of items separated by commas and when the 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

WRITE OUTPUT TAPE 3, 20, A, B, C

has the list "A, B, C" and the quantities A, B, and C will be written 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 written. 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 written (referring to the statement above) would be

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

If A and B were large arrays and one desired 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 it were desired 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.

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 should be followed by commas as shown. The indexing information, as "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 to be printed. It also refers to the number of a FORMAT statement which specifies the external 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.

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

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

The above statements might read the following lines of input data from cards into the 704.

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; however, only eight significant digits will be retained. If the decimal point punched in the card does not agree with the specifications in the FORMAT statement, the decimal point over rides the specification. 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 keypunching, 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.

```
      PRINT 10, A, B, C, D, I, E
   10 FORMAT (E14.4, F11.3, F10.5, E13.3, I6, F7.1)
```

The above statements would 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 provided for by the FORMAT statement. 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 will 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 additonal space for field separation.

FORMAT 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 spaced for field separation.

General Information About the Use of Tapes

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

When a WRITE OUTPUT TAPE statement is executed, the tape writing mechanism writes, as a single record, all of the quantities in the List. For example

```
WRITE OUTPUT TAPE 2, X, Y, Z
```

would cause the three numbers which are the current values of X, Y, and Z to be written as a single record (in the order X, Y, Z) on tape 2. Physically, the tape would be moved forward over the stationary

tape read-write head which records magnetically the three 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 writing the next record.

The effect of the statement

READ INPUT TAPE 2, X, Y, Z

would be to move tape 2 forward over the read-write head to start reading the first record encountered, assigning the first number of the record as the value of X, the second as the value of Y, and the third as the value of Z. If the list in the READ INPUT TAPE statement specifies more quantities then there are numbers in the record, the computer will stop since this condition is regarded as an error. If the list specifies N quantities and the next record on the tape contains more than N quantities, only the first N numbers will be read from the record. After reading these N numbers, the tape will be moved (without reading) to the next inter-record gap.

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. These are

REWIND I

BACKSPACE I

where I is the number of the tape unit.

REWIND moves the tape back to the physical starting point regardless of its current position and may be used to position the tape at the beginning of the first unit record to be written or read.

BACKSPACE moves the tape 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 TAPE I", with no list specified, may be used.

By use of the above statements, a tape may 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 will make it impossible to read any old information following this new record. It would not be possible to write over a record in the middle of a tape and then read old information written after this point. 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.

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 which can later be recognized by the tape reading mechanism to stop tape reading at that point.

Examples

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

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 will be 25 degrees apart and will 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 will include the amount of each gas present, the three empirical constants for each gas, the base temperature, and the maximum temperature (in ^{o}C).

A possible FORTRAN program 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.

```
    9  DIMENSION X(10), B(10), C(10), CP(10)
   10  FORMAT (10F6.3)
   11  FORMAT (10E11.3)
   12  READ INPUT TAPE 4, 10, X, A, T1, T2
   13  READ INPUT TAPE 4, 11, B, C
   14  SUM = 0.0
   15  T1K = T1 + 273.1
   16  TK = T1K
   17  TK = TK + 25.0
   18  IF ((TK - 273.1) -T2) 19, 27, 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, CP
26  GO TO 17
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 (F8.1, E14.5)
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 = ±xxx.x

Statements 31 and 32 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. Statement 14 sets the location designated as SUM to zero. The calculation of the absolute temperature in degrees Kelvin from the base temperature is carried out by statement 15. Statement 16 sets the original value of the temperature range to zero, while statement 17 causes the range to be increased by the specified increment. 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 and 25. 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 will be the indication that the problem is completed). If it does, control reaches statement 28, and end-of-file is written, tapes 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.

2. Given Xi, Yi, Zj for i = 1, 10 and j = 1, 20 to compute:

$$\text{PROD} = \left(\sum_{i=1}^{10} Ai \right) \left(\sum_{j=1}^{20} Zj \right)$$

where $Ai = X_i^2 + Y_i$ if $|X_i| > |Y_i|$

$Ai = X_i + Y_i^2$ if $|X_i| < |Y_i|$

$Ai = 0$ if $|X_i| = |Y_i|$

A possible FORTRAN program follows.

```
    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 $\sum_{i=1}^{10} Ai$. Statement 15 computes $\sum_{j=1}^{20} Zj$ under the control of DO statement 14. The following statements compute and print PROD. Statement 12, CONTINUE, serves as a common re-

ference point; and since it is the last statement in the range of the DO, after its completion I is increased and the next repetition begun.

SECTION II Check List

II. 1 All subscripted variables must appear in a DIMENSION statement. This statement must appear in the program before reference is made to the variables.

II. 2 Negative subscripts are not permitted.

II. 3 Subscripting of subscripts is not permitted.

II. 4 Integer variables and constants can be used only as subscripts and exponents in a floating-point expression.

II. 5 Integer constants are written without a decimal point; integer variables must begin with I, J, K, L, M, or N.

II. 6 The last statement in the range of a DO may not be a transfer.

II. 7 Decimal integers larger than 32767 are treated modulo 32768.

II. 8 An end-of-file should always be written on output tapes.

II. 9 Provision for rewinding tapes should be made in the program.

II. 10 No constants may be given in a LIST, only variables.

II. 11 The first character of the first field in a FORMAT statement for output must be a blank.

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