ANL-6542

Argonne National Laboratory

AN INTRODUCTION TO 704 FORTRAN

by

G. S. Pawlicki

ANL-6542
Mathematics and
Computers
(TID-4500, 17th Ed.)
AEC Research and
Development Report

ARGONNE NATIONAL LABORATORY
9700 South Cass Avenue
Argonne, Illinois

AN INTRODUCTION TO 704 FORTRAN

by

G. S. Pawlicki
International Institute of Nuclear Science and Engineering

March 1962

Operated by The University of Chicago
under
Contract W-31-109-eng-38

AN INTRODUCTION TO 704 FORTRAN

by

G. S. Pawlicki

ABSTRACT

It is hoped that this paper will permit the reader to begin to program computations on the 704 computer without referring to the Fortran Manual. Only a fraction of the possibilities of Fortran are described, but at all times an effort has been made to indicate where only a partial exposé has been made. Many specific examples are included to illustrate the word descriptions of Fortran programming. No information is supplied concerning operation of the 704 control console and accessories. These functions are performed by personnel of the Applied Mathematics Division. The main body of the paper is devoted to Fortran programming, and an appendix, which gives specific information concerning procedures of the Applied Mathematics Division relative to the use of the 704, is included.

FORTRAN is an abbreviation of the name "Formula Translator." It is called a formula translator, because formulas written almost like ordinary algebra are read from a hand-punched set of IBM cards by the 704 acting as a translator, and a set of cards is punched which serve to operate the 704 as a computer. The set of hand-punched cards is called the "Source Deck." The source deck is translated by the computer, which punches a translated deck of cards called the "Object Deck." The process of preparing an object deck from the source deck is called "Compiling." At "Object Time," that is, when a problem is being solved, the object deck is placed in the card reader, possibly followed by hand-punched "Data Cards" which supply numerical data required by the program. Data can also be read from magnetic tape, but this will not be considered.

The IBM cards used in the source deck and for data have a row of printed numbers from 1 to 80 which identify the 80 columns into which rectangular holes can be punched. The holes, which are punched in a single column, are coded to represent a unique character. The characters are the symbols - + , ( ) * . / as well as numerals and capital letters. At the top of each column is a space in which the card-punching machine also prints the character corresponding to what has been punched. Columns 73 to 80 inclusive are not read by the computer and can therefore be punched with identification such as numerical sequence of source cards and identification of cards as data and their sequence.

To describe the use of columns 1 through 72, probably it is best first to discuss the form of numbers which can be entered as data. Numbers can be entered as floating-point decimal numbers in normal or exponential representation, integers, and octal numbers. Only decimal and integer representation will now be considered. The normal decimal representation of a number will henceforth be referred to as "Floating Point." An integer represented without a decimal point will be called "Fixed Point." Any number is represented by punched holes in a certain series of columns on the IBM cards. The number of columns assigned to a specific number is called the Field Width.

The 72 columns available for numerical input of data can be divided into any desired number of fields of any width desired. The source deck must contain a FORMAT statement which instructs the computer in which way it should interpret the data card. The specification in the FORMAT statement for the floating-point format is

n F w · d

where n is the number of identical fields to be read (the first field begins with column 1 !); F is the code letter for floating-point representation; w is the field width; and d is the number of places to the right of the decimal point.

The specification of a fixed-point number in a FORMAT statement is of the form

n I w

where n and w are as previously defined and I is the code letter for fixed-point representation. In punching a data card with a number, the sign precedes the number. A floating-point number can be punched anywhere within the field. With a fixed-point number the number must be at the right side of the field; otherwise, the blank spaces at the right end of the field, read by the computer as zeros, would alter the number by powers of ten. In both representations, if the numerical sign is missing, the computer assumes that the number is positive.

Only single-line format for numerical quantities will be described. The source card for a FORMAT statement might be punched as follows:

Column 1	Blank
Columns 2 - 5	A Statement Number located anywhere in this field, an integral number without sign.
Column 6	Blank
Column 7 onward to 72	might be punched as follows:

FORMAT (3 F 6.2, F 5.4, I2, 3I4).

This FORMAT statement could be used to specify reading of a data card. The data card would have 3 fields of 6 columns, a field of 5 columns, a field of 2 columns, and 3 fields of 4 columns. The spelling, commas, and parentheses are essential parts of the statement and in all other types of statements to be described.

If a Fortran statement requires more space than the columns from 7 to 72, a statement can be continued on a second card by punching column 6. Up to 9 continuation cards can be used. With the exception of zero, what is punched in column 6 is immaterial and can even be the same on all continuation cards in the entire program, since the computer merely reads the cards in the order in which they are stacked in the source deck. Only one statement can be punched on a single card. The statement itself need not start in column 7, and any number of blanks can be inserted anywhere in a statement. The statement number is an integral number without a sign and can be located anywhere in the columns from 2 to 5. Statement numbers uniquely identify statements and need not be in any numerical sequence.

Numerical quantities in FORTRAN can be numerals or identified with a letter name. A floating-point constant has a decimal point whereas a fixed-point constant must not. The literal expression for a number can consist of a series of not more than 6 characters, which can be both letters and numerals with certain restrictions. The first character must be a letter. A fixed-point number name must begin with letters I, J, K, L, M, or N, and floating-point number names must not begin with these letters. A good rule is to avoid the use of F as the last character to prevent possible confusion with names of functions which are discussed on page 6.

Both floating- and fixed-point variables can be subscripted with up to 3 subscripts. The subscripts must necessarily be fixed-point numbers without sign. Subscripts cannot take on zero or negative values. The following examples completely cover limitations of naming and arithmetic operations for subscripts:

algebraic	FORTRAN
$a_{j,k,l}$	A(J,K,L)
x_a	X(IA)
$c_{i+2,m}$	C (I + 2, M)
$i_{j,m}$	I(J,M)
P_{2i-1}	P(2*I-1)

As previously mentioned, an equation can be punched in a form which looks very much like ordinary algebra. The algebraic symbols used are +, -, *, **, /, ( ), and =. These symbols represent addition, subtraction, multiplication, exponentiation, division, parentheses, and equality. As an example, the algebraic expression

$$y = (ax^2 + bx + c)^3$$

could be written as a FORTRAN Arithmetic Statement as follows

```
Y = (A*X**2 + B*X + C)**3
```

The mode of all terms (i.e., fixed or floating point) on the right side of the equal sign must be the same! An exception is that an exponent of a floating-point number can and should be fixed point if it is an integral number. The mode of the left side need not be the same as the right side. When the right-side mode is different from the left-side mode, the statement implies a conversion of the right-side quantity into the mode specified by the left side. In the example above, if the left side were I, I would be equal to Y truncated, to an integer.

In writing a FORTRAN arithmetic statement, the variables, constants, and equal symbol can frequently be thought to have the same meaning they have in algebra. Actually, the variables in FORTRAN are specifications of memory locations rather than the numbers themselves. The quantity computed by using the contents of memory locations specified on the right of the equal sign are to be stored in the location specified on the left of the equal sign. Thus the equal sign really has the meaning "store." This is best illustrated with the statement

```
A = A + B
```

Algebraically this only holds true for B = 0; however, in FORTRAN the results of the summing of the contents of memory locations A and B are to be stored in memory location A.

The use of parentheses in FORTRAN is very much like their use in algebra; that is, after encountering a left parenthesis, the computer will perform the indicated algebraic operations up to the right parenthesis, independently of any further algebraic operations outside of the parentheses. Thus, in the previous statement for Y, the quantity in the parentheses is computed and then raised to the third power.

In the absence of parentheses or within parentheses the order in which the algebraic operations are performed is exponentiation, multiplication or division, and finally addition or subtraction. Thus `BB = A*B**C + X` algebraically means the single variable called BB equals $(a \cdot b^c) + x$.

Parentheses are frequently necessary to write an arithmetic statement properly, and when in doubt it is probably wise to use them even if they are not absolutely necessary. To illustrate some cases in which parentheses are necessary, consider the following algebraic expression and their FORTRAN representation:

$a^{(b + c)}$	A ** (B + C)
$a^{(b \cdot c)}$	A ** (B * C)
$(a \cdot b)^c$	(A * B) ** C

An expression like

$$a^b \cdot c \qquad A**B*C$$

could have been written with extraneous parentheses as (A**B)*C. Whenever two of the characters +, -, /, *, **, appear consecutively, it is necessary to separate them with parentheses. Consider the following relationship which would not be ambiguous in algebra:

$$y = x \cdot -2. - + 4.6 \quad ;$$

this would have to be written in FORTRAN as

$$Y = X*(-2.) - (+4.6)$$

An example of an expression containing a subscripted variable might be

$$y = a_i^2 + b_j^m \qquad Y = A(I)**2 + B(J)**M$$

where the parentheses arise because of the FORTRAN subscripting notation.

FORTRAN has 5 types of functions, of which only 3 will be considered. These 3 are open (or built in) functions, closed (or library) functions, and arithmetic statement functions. These 3 functions have the same rules for their naming and for referring to them (calling).

The names of these 3 types of functions consist of 4 to 7 alphabetic or numeric characters of which the first must be alphabetic and the last is F. A function can have one or more arguments, but the function is single-valued. The mode of the arguments may be fixed or floating, and the mode of the function value can be either fixed or floating. If the value of the function is fixed point, the function name must begin with X. It is desirable to name a function with a sequence of letters which suggest its meaning.

To illustrate the naming of functions, we can consider the algebraic statement for y previously used:

$$y = (ax^2 + bx + c)^3 \quad ,$$

and define an Arithmetic Statement Function. The statement to be punched could be

$$YFUNF\ (X) = (A*X**2 + B*X + C)**3$$

The name YFUNF is followed by its argument X in parentheses. YFUNF has a value which is floating point (no X at start of name). Its argument on the left side clearly shows that the argument is floating point and, we note that to the right side of the equal sign the statement is correctly written in the same mode. As defined above, YFUNF has the argument X and has A, B, and C as parameters. If the left side of the statement had been written YFUNF(X,A,B,C), the function would have 4 arguments.

To have the argument floating and the function value fixed point, the statement would be

```
XYFUNF (X) = (A*X**2 + B*X + C)**3
```

If we wished to have a fixed-point argument and floating-point function, we might write

```
YFUN2F (I) = (JA*I**2 + JB*I + JC)**3
```

Note that the absence of X at the beginning of the name means that the value of the function is floating point. The argument in parentheses is fixed point, and to the right of the equal sign the arithmetic manipulations have the fixed-point variable I and the 3 fixed-point coefficient JA, JB, and JC to give a fixed-point-mode arithmetic expression consistent with the mode of the argument of the function in parentheses on the left side of the = sign.

The fourth possibility is to have fixed-point argument and fixed-point value for the function:

```
XYFUN2F(I) = (JA*I**2 + JB*I + JC*I)**3
```

In FORTRAN as in ordinary algebra we can put in any letters or numbers for the argument of a function, and so if the 4 foregoing functions are defined in a program, they may later be called by arithmetic statements as follows:

```
ALPHA   = 2.5 + YFUNF (7.2)
BETA    = 6.3 + YFUNF (Y(K))
GAMMA   = 5.75 + YFUN2F(J)
DELTA   = 20.5 + YFUN2F(31)
IOTA1   = 20 + XYFUNF(2.6)
IOTA2   = 25 + XYFUNF (Z)
LAMDA1  = 31 + XYFUN2F(IT)
LAMDA2  = 436 + XYFUN2F(39)
```

The built-in and library functions are defined within the 704 and, just as the foregoing functions were utilized in arithmetic statements, so library functions can be used by appropriately indicating the function name and its arguments. Seven library functions with floating-point values and arguments are

LOGF	natural log of the magnitude of the argument
SINF	sine (angle in radians)
COSF	cosine (angle in radians)
EXPF	exponential function
SQRTF	Plus square root of the magnitude of the argument
ATANF	$-\pi/2 <$ arctangent in radians $< \pi/2$
TANHF	hyperbolic tangent

Table I, taken from the FORTRAN manual, lists the built-in functions in the 704 FORTRAN, and their use is probably self-explanatory.

Table I

Type of Function	Definition	Number of Args	Name	Mode of Argument	Mode of Function
Absolute value	$\lvert Arg \rvert$	1	ABSF	Floating	Floating
			XABSF	Fixed	Fixed
Truncation	Sign of Arg times largest integer $\leq \lvert Arg \rvert$	1	INTF	Floating	Floating
			XINTF	Floating	Fixed
Remaindering (see note below)	Arg_1 (mod Arg_2)	2	MODF	Floating	Floating
			XMODF	Fixed	Fixed
Choosing largest value	Max (Arg_1, Arg_2,...)	≥ 2	MAXOF	Fixed	Floating
			MAX1F	Floating	Floating
			XMAXOF	Fixed	Fixed
			XMAX1F	Floating	Fixed
Choosing smallest value	Min (Arg_1, Arg_2,...)	≥ 2	MINOF	Fixed	Floating
			MIN1F	Floating	Floating
			XMINOF	Fixed	Fixed
			XMIN1F	Floating	Fixed
Float	Floating a fixed number	1	FLOATF	Fixed	Floating
Fix	Same as XINTF	1	XFIXF	Floating	Fixed
Transfer of sign	Sign of Arg_2 times $\lvert Arg_1 \rvert$	2	SIGNF	Floating	Floating
			XSIGNF	Fixed	Fixed
Positive difference	Arg_1, -Min (Arg_1, Arg_2)	2	DIMF	Floating	Floating
			XDIMF	Fixed	Fixed

NOTE: The function MODF (Arg_1, Arg_2) is defined as $Arg_1 - [Arg_1/Arg_2]\ Arg_2$, where [x] = integral part of x.

In summary with regard to the use of the 3 classes of functions described, the part of a statement which calls the function consists of the function name followed by a list of the arguments separated by commas and in parentheses, as for instance,

SOMEF (X, Y, Z(I))

Whenever a variable is subscripted, it is necessary to assign a sufficient number of memory locations for the array. The assignment of memory locations is done with a DIMENSION statement.

For instance, if in a problem we have the subscripted variables A(I), B(I,J), and C(I,J,K), which would be 1- , 2- , and 3-dimensional arrays, respectively, where I takes on values from 1 to 5, J values from 1 to 4, and K values from 1 to 3, then 5 memory locations are required for A(I), 20 locations for B(I,J), and 60 locations for C(I,J,K). The DIMENSION statement for this specific example would be

DIMENSION A(5), B(5,4), C(5,4,3)

In the source deck, the DIMENSION statement must appear before the first statement which refers to the subscripted variable.

A simple problem will now illustrate the use of the statements which have been described and to introduce a few other statements. The problem will be to have the program read 5 data cards, each of which has a single value of an angle in radians, and to have the computer evaluate the sine, cosine, and tangent of these angles, and form a four-column tabulation consisting of the angle, its sine, cosine, and tangent.

Columns 2 to 5	Columns 7 to 72
1	DIMENSION A(5)
2	FORMAT (F10.5)
3	FORMAT (4F10.5)
4	READ 2, (A(I), I = 1,5,1)
5	DO 9 I = 1,5,1
6	SI = SINF (A(I))
7	CO = COSF (A(I))
8	TA = SI/CO
9	PRINT 3, A(I),SI,CO,TA
10	STOP
11	END (0,1,0,0,1)

The statements have all been numbered for ease of description, though only statements 2, 3, and 9 require statement numbers for execution of this program. Statement 1 says that a "subscripted variable A, i.e., Angle, will require 5 memory locations." Statement 2 says; "a single floating-point number occupying a field of width 10 with 5 places to the right of the decimal point." This statement will be used by statement 4 to instruct the computer how to read data cards. Statement 3 says; "4 floating-point numbers, each occupying a field of width 10 with 5 places to the right of the decimal point." This statement is used by statement 9 to instruct the computer how to type the value of angle, sine, cosine, and tangent. Statement 4 has not been previously explained; it can be read "Read data cards according to the specification of statement 2 (i.e., FORMAT statement); what is being read defines the value of subscripted variable A(I) where the index I begins with 1, and runs to 5 in steps of 1." This read statement, at object time, will cause 5 data cards to be read because of the way the index I and FORMAT were specified. The numbers read are stored in the computer memory.

Statement 5 has not previously been explained. It can be read "do the instructions specified by the statements following up to and including statement 9, and then repeat this series of instructions as governed by the index I which is to take on the values 1 to 5 in unit steps." Thus in our case the DO statement specifies that statements 6 to 9 are to be executed 5 times; each time it repeats the index I is increased by unity. We can say the DO statement has defined a loop which is to be circled 5 times. The last number (1) which defines how I is incremented need not appear in statements 5 and 4 because its absence implies that it is unity. Within the DO loop the index number I is available for use in fixed-point arithmetic expressions. After the DO loop is satisfied, the index number I is no longer defined.

Statements 6 and 7 are simple arithmetic expressions which define floating-point variables in terms of the library functions SINF and COSF. Note that in naming the variables SI and CO, it is forbidden to name them SIN or COS, since a variable must never have a name which is that of a defined function without the terminal F.

Statement 8 is an Arithmetic Statement that TA the tangent is the sine divided by the cosine.

Statement 9 has not been previously explained; it can be read "Print on the output printer (typewriter) according to the specifications of FORMAT statement 3, the 4 quantities A(I), SI, CO, and TA."

If it is desirable to print out the 5A(I) values before entering the DO loop, a PRINT Statement could be placed between statements 4 and 5:

```
PRINT 2, (A(I), I = 1,5)
```

This would print the 5 values of A(I) on 5 successive lines. Note that the PRINT statement above and the READ statement 4 have the nature of a DO loop. Note that the list of variables in PRINT and READ statements requires parentheses only if the statements define an index and require looping.

Statement 10 will be executed after the DO loop is satisfied. Its effect at object time is to stop the computer.

The END Statement is the physically last card in the source deck. It essentially separates independent programs whose source decks are put into the card reader at the same time during compiling. At the Argonne National Laboratory, the numbers 01001 are the normal END statement specifications. Note that the computer performs the program in the sequence of the source deck cards except for the change caused by statements such as DO.

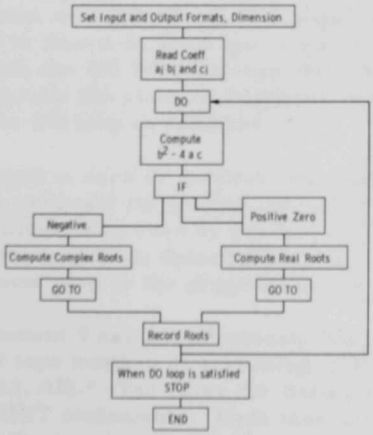
FORTRAN has a number of different conditional branching statements, of which only one will be considered. The statement

IF (A) 5,20,30

can be read "If the variable or expression A is negative, skip in the program to statement 5; if it is zero, skip to statement 20; and if it is positive, skip to statement 30." Care must be exercised in testing floating-point expressions for zero because of truncation and round-off errors which are normally introduced into a calculation. Fixed-point arithmetic, on the other hand, is exact, and the skip on zero presents no difficulties.

The unconditional branch statement is GO TO.

The following problem will serve to illustrate the use of IF and GO TO statements. The problem is to factor 5 quadratic equations by means of the quadratic formula. The roots of the equations can be either real or complex. The block diagram of the program is as follows.



```
      DIMENSION A(5), B(5), C(5)
5     FORMAT (3 F 10.5)
6     FORMAT (4 F 10.5)
      READ 5, (A(I), B(I), C(I), I = 1,5)
         DO 9  I = 1, 5
            D = B(I)**2 - 4.*A(I)*C(I)
               IF(D)10,4,4
10          R1 = -B(I)/(2.*A(I))
            R2 = R1
           AI1 = SQRTF (D)/(2.*A(I))
           AI2 = -AI1
                  GO TO 9
4           R1 = (-B(I) + SQRTF(D))/(2.*A(I))
            R2 = (-B(I) - SQRTF(D))/(2.*A(I))
           AI1 = 0.
           AI2 = 0.
9           WRITE OUTPUT TAPE 2, 6,    R1, AI1, R2, AI2
               END FILE 2
               STOP
               END (0,1,0,0,1)
```

In the foregoing program, the IF statement tests if the discriminant $b^2 - 4ac$ is negative, zero, or positive. If it is negative, the program beginning with statement 10 is followed, since the roots are complex. After the complex roots are found, the GO TO statement tells the computer to execute statement #9, which records the roots on magnetic tapes, and then the program continues until the DO loop is satisfied.

If the discriminant is zero or positive, the roots are real, and the IF statement says that the program for finding the real roots begins with statement #4. If the discriminant was zero or positive, statement 9 follows immediately after the roots have been determined, and no GO TO statement is necessary to have the next step in the program be the recording of the roots.

The output statement 9 has not previously been explained. It can be read: "Write on output tape number 2, according to FORMAT statement 6, the numbers R1, AI1, R2, AI2." The rules for listing the output quantities are the same as for PRINT statements. Each tape unit has a ten-position switch which identifies the tape number, but at Argonne, tape number 1 is

not to be used. Since magnetic tape recording is much faster than the "On line" printing (on the Printer directly associated with the 704), the time charges on the 704 are reduced. The recorded output tape is later read and printed by an "Off line" printer for which there is no additional charge at the Argonne computer center.

The END FILE statement marks the output tape number 2 after all the output has been recorded. The END FILE mark on the tape is used to stop the off-line printer. At the Argonne computer, the convention is not to include a REWIND statement for the output tape, so that the computer operator can record the output from several problems on the same reel of tape.

Thus far, a major part of the description has been concerned with introducing various types of statements and describing the form in which they are written; some further mention must be made of restrictions.

Of the statements described, the FORMAT and DIMENSION statements are called non-executable statements, that is, they do not cause an operation to be performed. The first statement after a DO statement must not be a non-executable statement. The last statement of a DO loop must not be a statement which transfers control, as does GO TO. When a control transfer statement is the last executable statement in a DO loop, the non-executable dummy statement CONTINUE is made the last statement of the DO loop.

The successive statements between the DO statement and the last statement of the DO loop define the range of the loop. Frequently, other DO statements occur within a DO loop. When this occurs, the range of the earlier DO statement encountered must extend up to or beyond the range of any DO loops defined within the range of the first DO loop. This says that DO statements must be nested; there cannot be just a partial overlap of ranges of DO loops.

Transfer into a DO loop from statements outside of the range of a DO loop can only be made to the DO statement itself, and not to any other statement in the range of the DO loop. With a single DO loop, transfers are permitted to statements within the range of the loop. If there are nested DO statements in a program, any transfers to the inner DO statements can only be to the DO statements itself so as not to violate the rule in regard to transferring stated above. The calling of any function or subroutine involves a transfer out and then back into a program; this sort of transfer is permitted within the range of a DO loop provided the subroutine returns to the same part of the loop and that the subroutine does not alter the value of indices. It should also be emphasized that no statement within a DO loop must redefine the indices. It should be noted that all restrictions apply to entering DO loops and not to exiting.

Some mention should be made of size limitations of numbers. A fixed-point number must be less than or equal to 2^{17}. A floating-point number can be between 10^{+38} and 10^{-38} or zero; however, the computer carries only 8 significant digits!! A greater number of significant digits on a data card are not read.

The maximum length of a FORTRAN statement is 660 characters or blanks, and requires 9 continue cards. The maximum number of characters and blanks which can be printed on a single line is 120. The length of a single tape record on output tape is also 120 characters and blanks. A tape record is printed off-line as a single line.

Whenever an Arithmetic Statement Function is defined, it must precede the first executable statement in the program.

It is frequently convenient to include comment card in the source deck. Comment cards are punched with a C in column #1, and any comments such as program name, date, and program author are punched in columns 2 through 80 on any number of cards in succession. Comment cards at the beginning are a convenient way for the operator to identify the program being compiled, since the comment cards and all the statement cards are printed during compiling. Comment cards produce no effect upon the object deck.

In writing FORTRAN, care must be taken not to confuse the letter O with the numeral zero, which are punched differently on a card. The letter O should be written as Ø. The letter I and the number 1 can also cause trouble although this is due more to careless writing. The printing on a punched card is different for I and 1, so no confusion results.

Acknowledgments

The author appreciates the suggestions and comments of numerous people in the Applied Mathematics and International Institute divisions. The author is particularly grateful to Mr. Burton Garbow of the Applied Mathematics division for his detailed review of the original manuscript. Suggested rephrasing of certain parts of the text which might be more satisfying to an experienced 704 user have not been incorporated in the final form of the paper in the hope that its form is more comprehensible to a novice.

APPENDIX

Some additional information on procedures of the Argonne computer center will now be given. This will be helpful in carrying through a program to the final solution of the problem.

The program is most conveniently written on the form shown in Figure 1. Data are written on the form shown in Figure 2. The forms are available in the stationery section of the Building 203 stockroom, Room R129 on the main floor.

After preparing the written program, the source cards can be punched in Room C055 in the basement of Building 203. After an initial keypunch practice period, source cards should be punched by the girls in Room C055. Arrangements for card punching should be made with Lead Data Transcription Operator, Room C055, Extension 2876. Other card punches are located in various other divisions of the Laboratory, including one in Building 25A, Extension 2294. Generally, at each card-punching machine a supply of blank IBM cards is available. Figure 3 shows a sample source card and a data card.

The IBM 704 computer is currently installed in Room D001 in the basement of Building 203. In the same room is a 1401 computer which is programmed to check cards of the source deck. The checking of a source deck is referred to as "Pre-processing." An applied mathematics memo is available which includes a list of errors checked by the preprocessor and the code name by which they are identified. There is no additional time charge for preprocessing and, since it greatly increases the probability that the program will work on the 704, all source decks should be preprocessed. The 1401 is scheduled for preprocessing from 1030 to 1115 and from 1500 to 1545; if requested, the 1401 operator will check the results and, if it appears that the program can be compiled, will place the deck in the next compiling batch. Compiling is accomplished from 1200 to 1300 (if requested) during the day; also, compiling is done at night for pickup the next morning.

An application for authorization to obtain use of the 704 must be filed with the Applied Mathematics Division in the division office, Room B153, Building 203, by means of form AMD 6. AMD then establishes a job number and job time cards are sent to the computer room. Argonne National Laboratory employees should become acquainted with the policy of their Division in relation to authorization of computer use. Use of the 704 by persons not employed by Argonne is authorized by the Division sponsoring the person.

The specific time at which the computer is to be used is scheduled at the computer room or by calling the scheduler at Extension 2877. Requests for time of 15 min or less from 0830 to 1200 are made after 1200 of

the preceding day; requests for time from 1300 to 1600 are made prior to 1200 of that day. The schedule is written on the blackboard in the computer room. Time on the 704 can be scheduled from 1700-2400 for periods of time greater than 15 min.

Whenever time is scheduled, a time card and a set of instructions for the computer operator must be provided. Form AMD 5 is used for the instructions. The necessary information on the form is the following:

(A) The nature of the run, that is, whether it is compiling, computation, or both.

(B) Does the program have on-line printing or punching?

(C) Does it have output on magnetic tape and, if so, which tape switch number setting should be made? It should be specified if a single or double copy should be produced by the off-line printer. The single and double copy are referred to as 1 or 2 part paper, respectively. If the tape is to be saved, what identification should it have on its label? (Label information should include the name of the job, the name of the person who requested that the tape be saved, and the date after which the tape can be reused.) If no labeling information is provided, it is assumed that the tape need not be saved.

(D) Does the program use any data cards?

(E) What are the sense switch settings? (This introduction has not covered sense switches, so one merely says they are all up.)

(F) What is estimated duration of the run? The operator should be informed when to stop a run, particularly if the program is new, to avoid wasting time on a program which may malfunction. For programs which have been previously run successfully, the operator can be told to allow the program to run to completion and the time required can be predicted with reasonable accuracy.

After compiling, the source and object decks and the printed sheets produced by the compiler are on distribution shelves in D-001 where they can be picked up.

The off-line printing from magnetic tapes is routinely done during the day within an hour and a half of the completion of the computer run. The printed output can be picked up from the shelves adjacent to the 1401 computer.

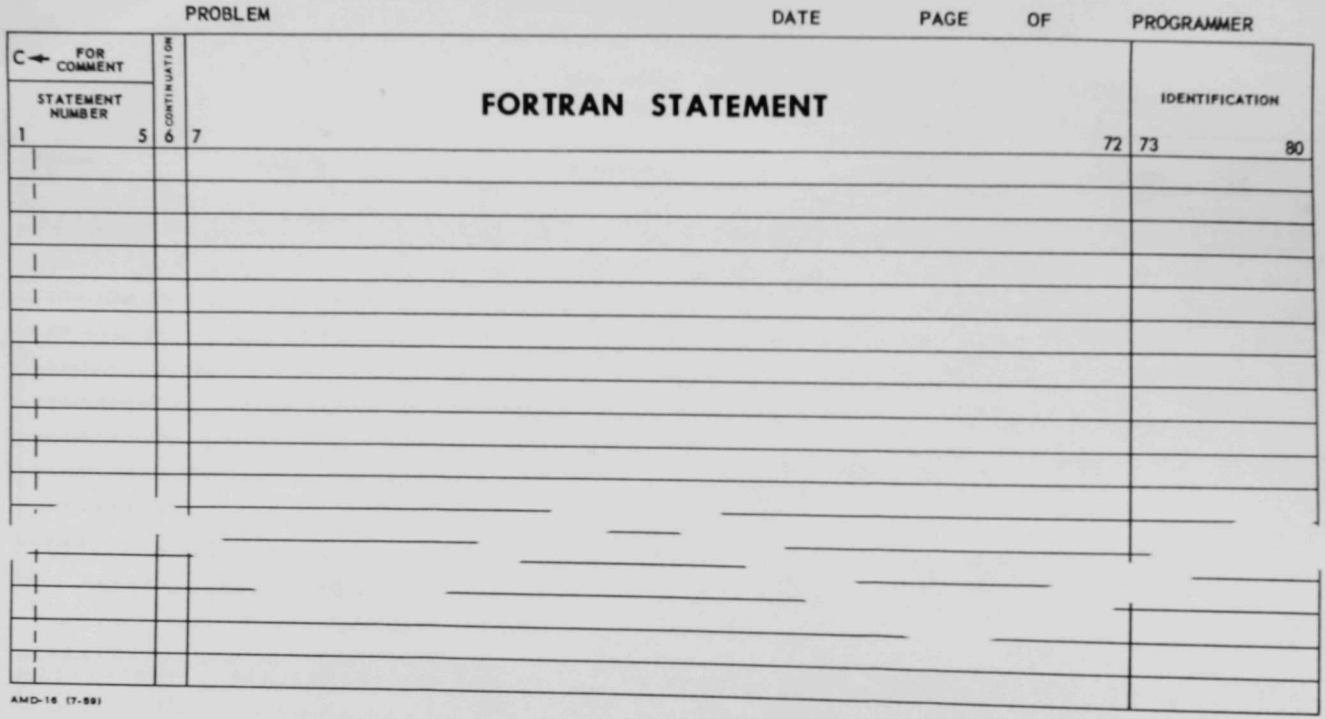


Figure 1

IBM Fortran Programming Form (reproduced by permission of International Business Machines Corporation.)

704 INPUT DATA

FORM I

COST CODE _____

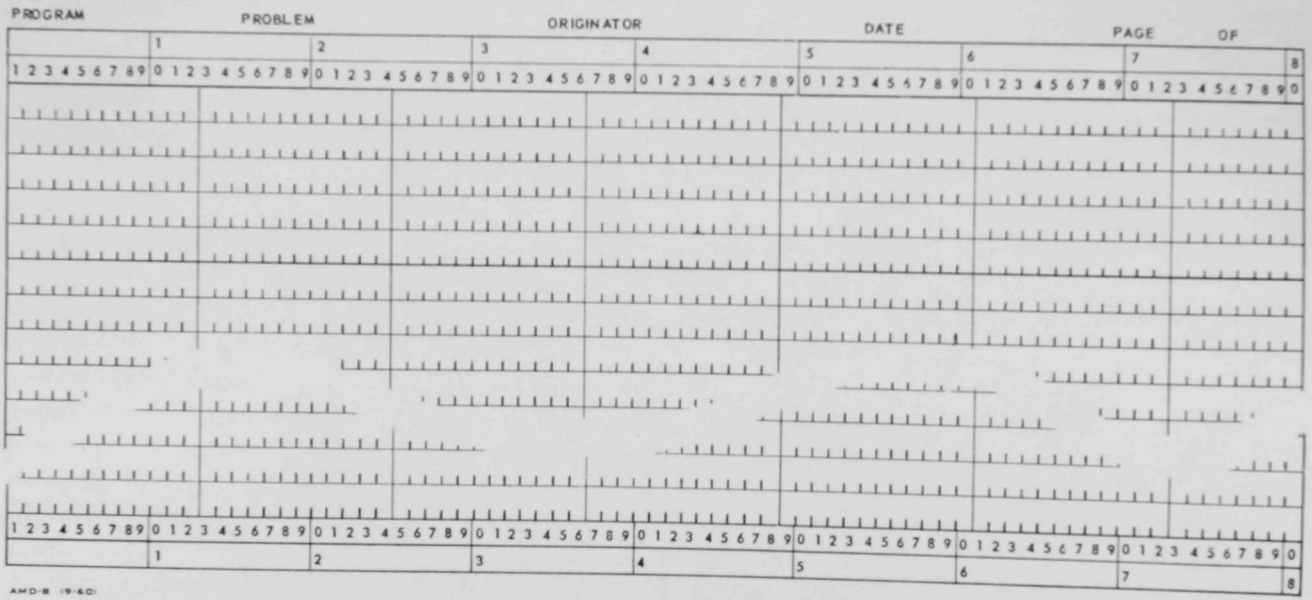
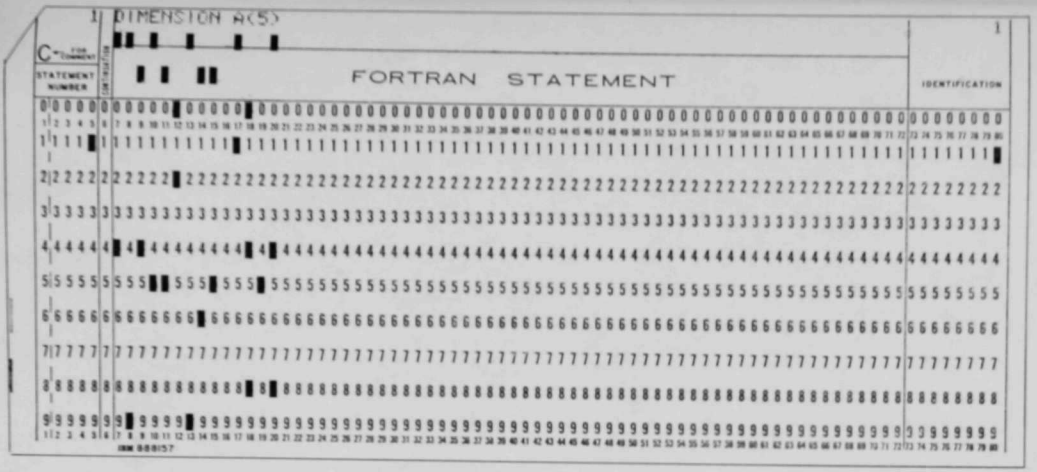
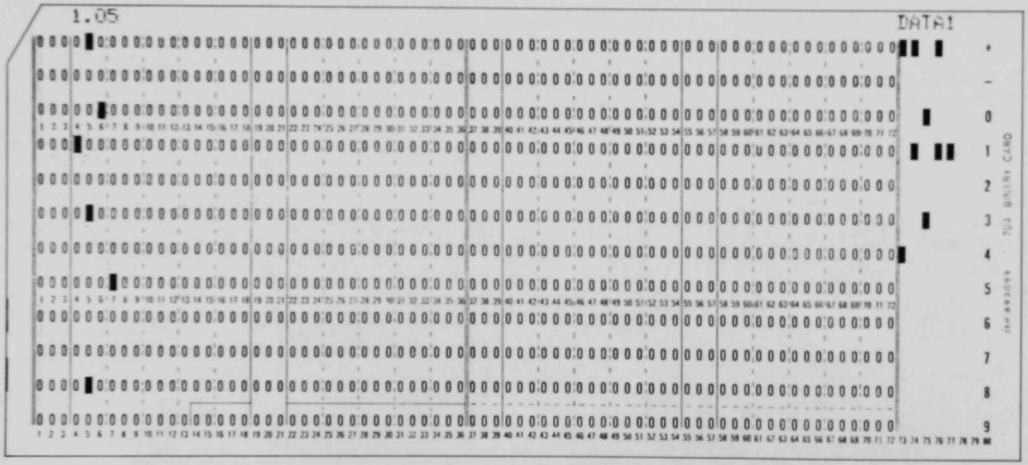


Figure 2



Sample FORTRAN Statement card. Note difference in printing of capital letter I and the numeral one. The letter O and zero can only be distinguished by the punched holes.



Sample data card. Note that a different type of card is normally used for the source cards and data cards.

Figure 3

IBM Forms 888157 and 893099 (reproduced by permission of International Business Machines Corporation).