BRL

REPORT NO. 1209
JULY 1963

INTRODUCTORY PROGRAMMING FOR ORDVAC AND BRLESC FORAST (Formula and Assembly Translator)

Michael J. Romanelli

RDT & E Project No. 1M010501A003
BALLISTIC RESEARCH LABORATORIES

ABERDEEN PROVING GROUND, MARYLAND

DDC AVAILABILITY NOTICE

Qualified requestors may obtain copies of this report from DDC.

The findings in this report are not to be construed as an official Department of the Army position.

BALLISTIC RESEARCH LABORATORIES

REPORT NO. 1209

JULY 1963

INTRODUCTORY PROGRAMMING FOR ORDVAC AND BRLESC
FORAST (Formula and Assembly Translator)

Michael J. Romanelli

Computing Laboratory

RDT & E Project No. 1M010501A003

ABERDEEN PROVING GROUND, MARYLAND

BALLISTIC RESEARCH LABORATORIES

REPORT NO. 1209

MJRomanelli/sf
Aberdeen Proving Ground, Md.
July 1963

INTRODUCTORY PROGRAMMING FOR ORDVAC AND BRLESC
FORAST (Formula and Assembly Translator)

ABSTRACT

FORAST is a programming language designed for use on ORDVAC and BRLESC, the high-speed digital computers of the Ballistic Research Laboratories. Programs written in this language, with minor limitations, may be executed on either computer. BRL Report No. 1172, [1] , describes FORAST in its generality and was written primarily for professional programmers. This report is intended for the novice. Fundamental concepts and details of the language are illustrated in many examples so that the novice is taught how to program and obtain practical solutions for a variety of mathematical problems. Intended as a supplement to [1.] , this report does not illustrate the full generality of the language. Some of the material is repetitious but amplified and several references are made to [1] .

TABLE OF CONTENTS

PAGE

TABLE OF CONTENTS

I. INTRODUCTION

The Ballistic Research Laboratories' high-speed computers, ORDVAC and BRLESC, are used to obtain solutions of mathematical problems. In general, the numerical solutions are obtained by instructing the high-speed computers to EVALUATE FUNCTIONS that approximate the solutions desired. Hence, to obtain solutions from high-speed computers, the computers are instructed to carry out the detailed operations required to evaluate particular functions of interest.

Conventional mathematical notation, which permits considerable freedom of expression and in which problems, solutions, and functions are generally expressed in a form governed by human convenience, is not fully acceptable by high-speed computers. Indeed, the instructions to computers that specify the essential operations and the numerical quantities involved in the operations are ultimately expressed in a PRIMITIVE LANGUAGE that is understood by the computer. For most high-speed digital computers, the primitive language consists of combinations of only two characters, 0 and 1, the basic characters of the binary number system. Specific combinations of these characters must be constructed to represent the sequence of operations and the quantities required to obtain a numerical solution. The human task of constructing the required combinations of binary characters is tedious, difficult and highly susceptible to human error. To simplify this task, HIGHER-LEVEL LANGUAGES are designed so that computers can be instructed by professionals with programs in their primitive languages to automatically translate them into their respective primitive languages. In contrast to the PRIMITIVE LANGUAGE, the HIGHER-LEVEL LANGUAGES admit combinations of the letters of the alphabet, the decimal digits, and some mathematical symbols such as + - = . etc. The programs which automatically accept the higher-level language and carry out the translation are called COMPILERS, ASSEMBLERS, TRANSLATORS, etc., and generally admit some of the freedom of mathematical notation together with some English word statements for controlling sequences of operations. Although the higher-level languages do indeed relieve the human of many tedious details, a novice may still feel burdened with the necessity for specifying particular details required in a high-level language.

FORAST is a high-level language designed for use on ORDVAC and BRLESC. BRL Report No. 1172, [1] , describing FORAST in its generality was intended as a reference manual for professional programmers. This report is intended for the novice and is confined to fundamental concepts and important details illustrated by numerous examples, teaching the novice how to program a variety of mathematical problems. Each example includes:

a statement of a problem;
a method for obtaining a solution;
a chart indicating the sequence of the necessary operations;
a program written in FORAST;
a tabulated copy of the results obtained from the computer.

Many of the methods used in the examples were chosen to illustrate particular concepts of the language and do not necessarily represent the most efficient solutions in the given situations.

II. EVALUATION OF FUNCTIONS

Solutions to many mathematical problems can be reduced to evaluation and recording of mathematical functions. Below are listed·the four associated operations:

1.) SUBSTITUTION of numerical values for each argument of the functions;

2.) PERFORMING the indicated arithmetic operations;

3.) RECORDING numerical values of relevant functions;

4.) CONTROLLING the sequence of operations involved in 1,2, and 3.

More specifically, to obtain a solution to a problem from a high-speed computer, we instruct the computer to substitute, to evaluate, to record and to control all operations required to obtain a solution.

In a given problem, all these operations must be planned in advance and the computer must receive specific instructions as indicated on a Flow Chart. A general Flow Chart is illustrated schematically in FIGURE 1.

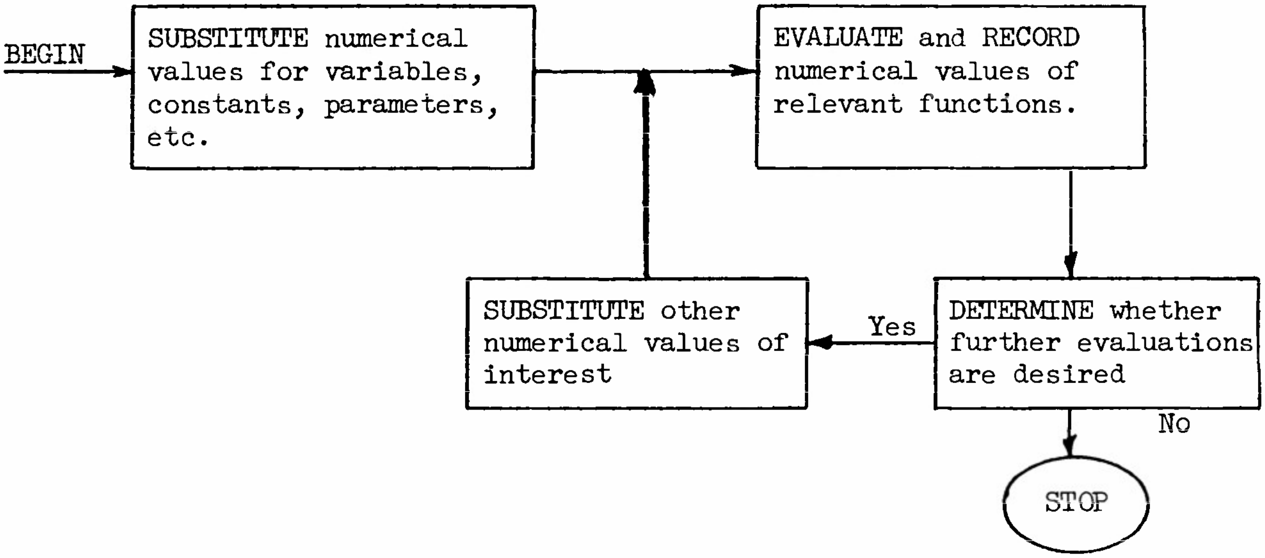

BEGIN
SUBSTITUTE numerical values for variables, constants, parameters, etc.
EVALUATE and RECORD numerical values of relevant functions.
SUBSTITUTE other numerical values of interest
Yes
DETERMINE whether further evaluations are desired
No
STOP

FIGURE 1.

GENERAL FLOW CHART FOR EVALUATING FUNCTIONS

Before describing the fundamental concepts of the language, we consider next the major components of the computers and the function of each as regards their use in obtaining a solution of a problem.

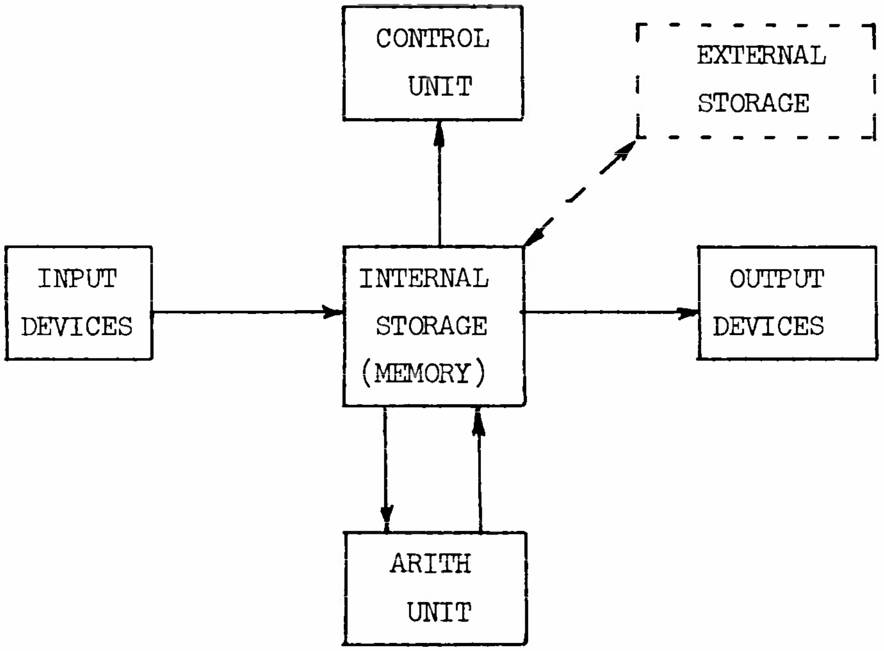


FIGURE 2.

MAJOR COMPONENTS OF HIGH-SPEED DIGITAL COMPUTERS

The components shown in FIGURE 2. are typical for most high-speed digital computers. The input devices generally consist of electro-mechanical means of reading paper cards, magnetic or paper tape.* The output devices use similar

* To record information on paper cards or paper tape, the cards or tapes are perforated, (i.e., holes are punched in them), wherein specific combinations of punches define specific characters. On magnetic tape, specific combinations of small magnetized areas correspond to specific characters.

means to print or record relevant results. Many computers use combinations of these devices together with high-speed printers and other optical or photographic devices for output. The input-output devices may also serve as external storage units. Magnetic tape units may and are used as input or output devices or even as external storage. Hence input and output devices are used to get information into and out of the internal storage as indicated by the arrows of FIGURE 2. In general, the information that passes through the input devices is recorded in internal storage and the information that is recorded on output devices is a copy of selected information that exists in internal storage. Information recorded in internal storage is retained there as long as desired, indeed it is retained there until the computer is instructed to replace it with other information. The arithmetic unit is used to perform ordinary arithmetic, i.e., addition, subtraction, multiplication and division. The control unit directs and activates all units in accordance with a set of instructions that have been recorded in internal storage.

Knowing the capabilities of the major units and the fundamental concepts of the high-level language, a programmer designs a set of instructions (expressed in the high-level language) that will when executed by the computer produce a solution to a given problem. The set of instructions and accompanying data (called a program) are recorded on paper cards. These cards are inserted in a card input device. Without loss of generality, one may assume that, after activating a few switches on the control panel of the computer, an equivalent copy of the instructions that were recorded on cards, is recorded in internal storage. The instructions are then executed and results produced accordingly. Actually, the FORAST COMPILER which is temporarily recorded in internal storage activates the card input device, and translates the high-level language to the primitive machine language. It is the primitive machine language corresponding to the high-level language that is recorded in internal storage. In interpreting and attempting to transform the high-level language, the compiler can and does detect some violations of grammatical errors; in such cases, it records pertinent information on the output device that identifies the particular violation and does not permit the computer to attempt to execute the program.

In summary, corresponding to a given problem, we:

1.) plan a solution and exhibit the plan in the form of a flow-chart;

2.) write a set of instructions in the high-level language to execute the plan;

3.) record the set of instructions and accompanying data on paper cards;

4.) submit the package to the computer;

5.) obtain the computer output and tabulate the results.

Although not explicitly stated previously, it is important to recognize that once a program has been tried, tested and proved, it can be re-used for other sets of numerical data.

EXAMPLE 1.

To illustrate some of the general concepts stated thus far, and to introduce some fundamentals of the language we consider next Example 1.

GIVEN: X_i , Y_i , where $i = 1,2,3, \ldots$

i.e., a table of numerical values for

X	Y
1	1
30	40
-8	6
.	.
.	.
.	.

REQUIRED: For each pair of values, X_i, Y_i, compute and record

$$Z(X_i,Y_i) = \sqrt{X_i^2 + Y_i^2}$$

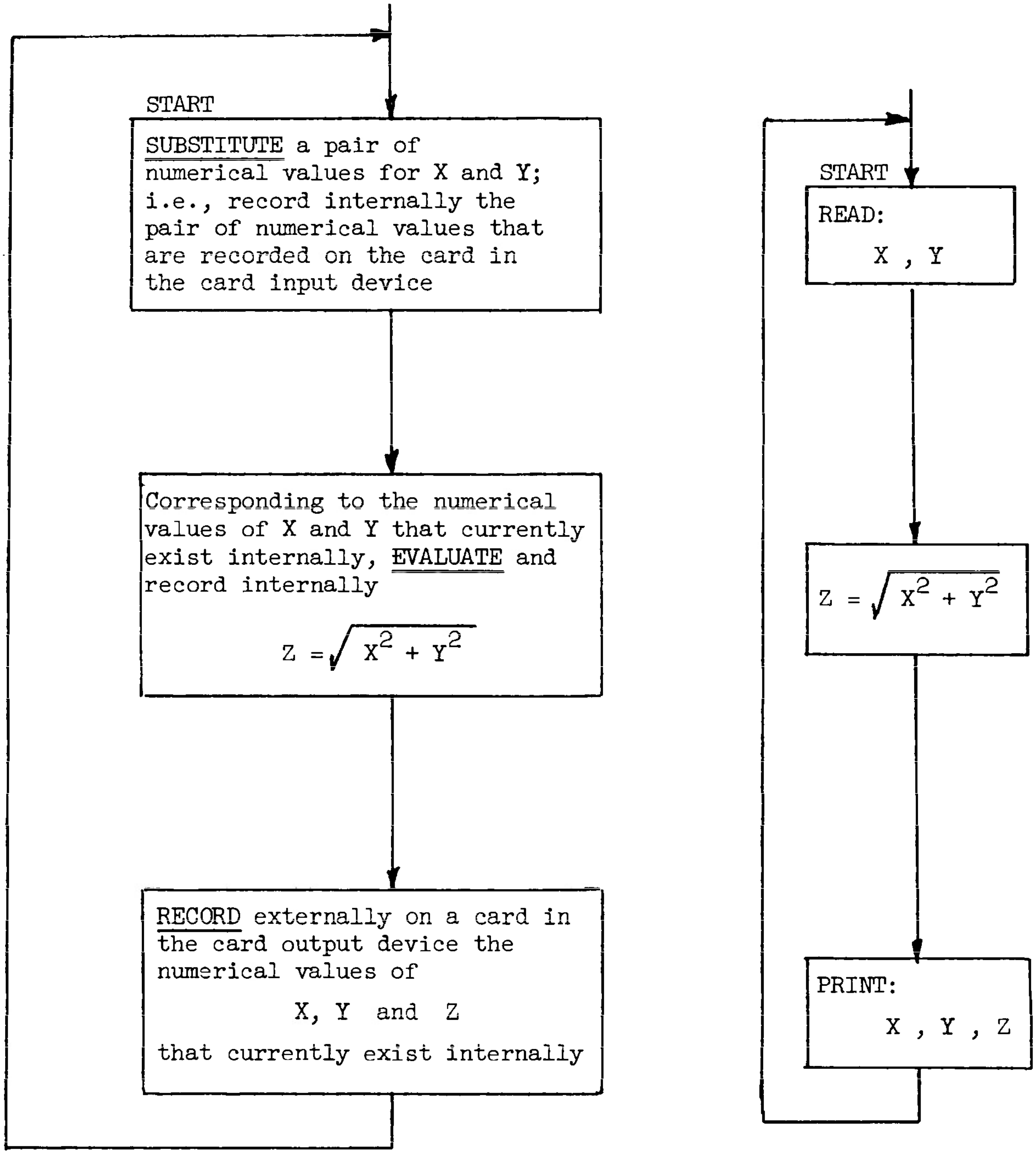
i.e., we want to produce a table with entries

X	Y	Z(X,Y)
1	1	1.414
30	40	50
-8	6	10
.	.	.
.	.	.
.	.	.

for an indefinite number of pairs, (X,Y).

To plan a solution for this problem, we tacitly assume that the given numerical values of X and Y will be recorded in a "prescribed form" on paper cards, one pair of values on each card, and utilizing as many cards as are necessary for the complete table of values. Although this is not the most efficient method for recording the data, it is a practical assumption that is readily fulfilled. One could assume two, three, or more pairs recorded on each card; however, for simplicity we assume one pair per card.

Next, we make use of a fundamental characteristic of card-input devices. Cards containing information to be used in the solution of a problem are stacked in the input-hopper of the input-device with the card at the bottom of the stack "engaged" under a "read" station. (See FIGURE 7.) When the input-device is activated by the computer, the information punched in the card under the "read" station is interpreted and the information transmitted to internal storage. During the process of interpretation and transmission, the card automatically moves to a storage bin and the "next" card in the stack moves under the read station ready for the next activation of the input device. Hence, cards that have been read accumulate in the storage bin and are not accessible to the computer for re-reading unless manually re-inserted in the input-hopper. Cards are generally read only once for a given problem since the information recorded on them is recorded in internal storage indefinitely for as many future references as desired. A similar sequence of card-processing applies to card-output devices, i.e., the automatic passage of cards from input-hopper to "punch" station and then to a storage bin. To utilize this fundamental sequence, we plan to have a pair of values read and substituted for X and Y, instruct the computer to evaluate the corresponding function, record the result on the card output device and then repeat the process using the next pair, etc. Hence our plan can be exhibited in the form of the simple Flow-Chart as shown in FIGURE 3.


START
SUBSTITUTE a pair of numerical values for X and Y; i.e., record internally the pair of numerical values that are recorded on the card in the card input device
Corresponding to the numerical values of X and Y that currently exist internally, EVALUATE and record internally
$Z = \sqrt{X^2 + Y^2}$
RECORD externally on a card in the card output device the numerical values of
X, Y and Z
that currently exist internally
START
READ:
X , Y
$Z = \sqrt{X^2 + Y^2}$
PRINT:
X , Y , Z

"WORDY" CHART

CONCISE SYMBOLIC CHART

FIGURE 3.

A SIMPLE PLAN TO OBTAIN A SOLUTION FOR EXAMPLE 1.

The "wordy" chart is intentionally "wordy" for the beginner. The concise symbolic chart is generally used and implies all that is stated explicitly in the "wordy" chart. Note the capitalized words: SUBSTITUTE and its equivalent, READ; EVALUATE and the equivalent explicit definition, $Z = \sqrt{X^2 + Y^2}$; RECORD (externally) and its equivalent, PRINT. (PRINT and PUNCH are synonomous.) Note also that both charts indicate that, after recording results corresponding to a given pair of values, the next processing desired is indicated by a line to the original START sequence which causes the "next" pair to be processed in the same manner as its predecessor. The process is to be continued until the cards in the input-device are exhausted.

Next, we write instructions in the FORAST language to instruct the computer to carry out the desired processes shown on the flow-chart. These instructions are written on the standard CODING FORM shown in FIGURE 4.

FORAST CODING FORM

PROBLEM C906

CODER M.J.Romanelli DATE 1 Jul 63 PAGE 1

LOCATION	ORDER TYPE	FORMULAS STATEMENTS COMMENTS	CARD IDEN
1 6	7 10	11 76	77 80
	PROB	C906 M.J.ROMANELLI 45107 EXAMPLE 1.	1
START		READ(X)Y	2
		Z = SQRT(X**2 + Y**2)	3
		PRINT(X)Y)Z % GOTO(START)	4
	END	GOTO(START)	5

FORAST INSTRUCTIONS TO CARRY OUT PLAN OF FIGURE 3.

FIGURE 4.

The instructions written on the Coding Form are punched on cards which can then be submitted to the computer. The cards corresponding to the instructions written on the Coding Form of FIGURE 4. are shown in FIGURE 5.

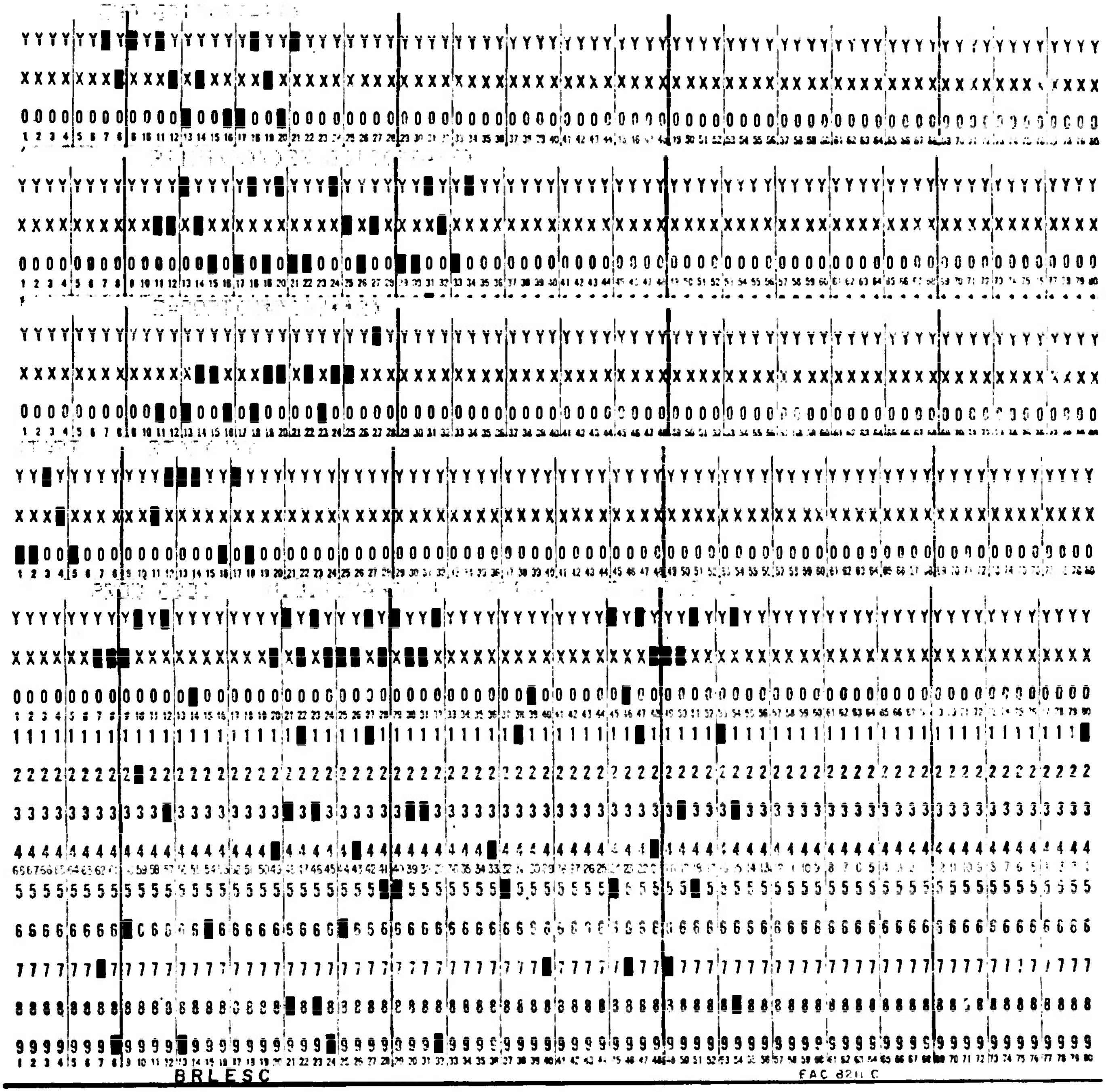


FIGURE 5.

The instruction cards of FIGURE 5, together with a few data cards, (the cards on which we assumed the X's and Y's would be recorded), are shown in FIGURE 6. These cards represent the complete package to be submitted to the computer to obtain the desired solution.

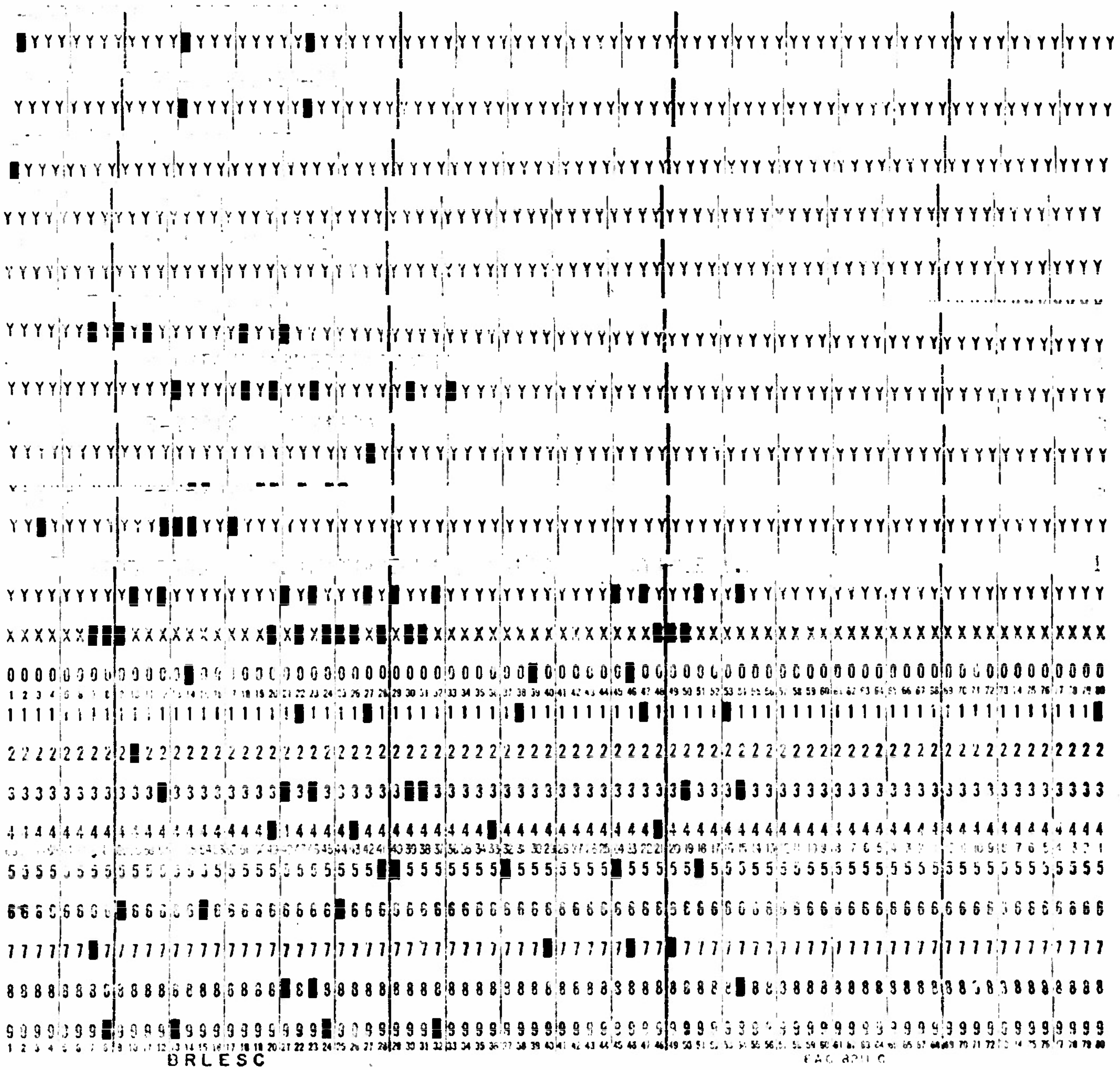


FIGURE 6.

Recall that we assumed the given input data, (the numerical values of the X's and Y's), would be recorded on paper cards in a "prescribed form". The FORAST programming language provides for instructing the computers to accept data represented on cards in many forms. A few specific forms are considered standard, (i.e., those forms that are most convenient for scientific calculations and used frequently), and only departures from standard forms must be specified in a manner provided by the language. The prescribed form chosen for this example is referred to as standard "floating-point" form. This form is similar to that generally called scientific notation, where numerical values are expressed in a two-part form; one part called the "coefficient", the other is called the "exponent". For example, the quantity 72.45 can be expressed as:

$$7.245 \times 10 \quad \text{or} \quad .7245 \times 10^{2} \quad \text{or} \quad 7245. \times 10^{-2} \text{ , etc.}$$

The base ten is generally omitted and the two-part form used consists of the signed coefficient and the signed exponent. The exponent is restricted to integers. The computers represent numbers internally in a similar manner, using binary notation instead of decimal. In ORDVAC, the base is 2, binary; in BRLESC, the base is 16, sexadecimal. The use of other bases and forms is permissible and means for transforming from one base or form to another base or form is provided.

For input data, some standard floating-point forms used are:

	Coefficient	Exponent
a.)	Sign,8 decimal digits	Sign,2 decimal digits
b.)	Sign,7 decimal digits and a decimal point	Sign,2 decimal digits
c.)	Sign,11 decimal digits	(no exponent)
d.)	Sign,10 decimal digits and a decimal point	(no exponent)

In the forms where the decimal point of the coefficient is not specified, (forms "a" and "c"), it is assumed to be before the first decimal digit punched. In each of the above forms, 12 columns of a card are used. Other variations of these forms are permitted; however, in EXAMPLE 1. we used form "a" for both input and output representations. As is conventional in mathematical notation, the positive sign may be omitted.

To introduce some of the fundamental rules of the FORAST language, we refer the reader to the standard coding form shown in FIGURE 4.

Observe the division of the form into four general headings, namely: LOCATION, (1-6); ORDER TYPE, (7-10); FORMULA STATEMENTS, (11-76); and CARD IDEN, (77-80). The numbers written here in parentheses, and written under the general headings on the coding form correspond to the columns of the paper cards on which the information will be recorded. The space allocated for columns (1-6), labelled LOCATION, is reserved for names to refer to statements and pertinent data. Columns (7-10), labelled ORDER TYPE, are reserved for classifying the information recorded in columns (11-76). Only a small number of standard names are permitted in this space. A list of such names will be furnished later. Columns (11-76) are reserved for the formulas, English word statements, and other data required to describe the processes necessary to obtain a desired solution. Columns (76-80), labelled CARD IDEN are reserved for arbitrary identification of individual cards. Although not mandatory, the cards are generally ordered in numerical order for easy reference so that if and when the compiler detects a violation when it reads a card, the identification recorded in columns (76-80) of the card is printed to identify the card containing the violation.

The first card of each program submitted to the computer must have PROB recorded in columns (7-10). Columns (11-76) of this card may be used to record a problem number, persons name, title of problem, etc. The information recorded on this card is always recorded as the first card of the output to identify the subsequent results which follow, if any!

To identify the beginning of a program, an arbitrary name composed of at most 6 characters must be recorded in columns (1-6), i.e., opposite the first statement to be executed. In the example, we chose the name, START. Note that commencing in column 11 of this card we wrote READ(X)Y, the instruction designed to start the processing of a data pair by copying in internal storage the numerical values recorded on the first data card. Next, we wrote the function to be evaluated. No doubt the beginner will note the difference in the conventional mathematical notation, $Z = \sqrt{X^2 + Y^2}$ and the cumbersome notation Z = SQRT(X**2 + Y**2). Unfortunately, the former is not acceptable to the input media. In contrast, the latter uses a serial arrangement of acceptable symbols or their combinations. Subscripts, superscripts, exponents or other conventional symbolism which is not included in the restricted set of characters acceptable by input media are denoted by special characters in the restricted set or by combinations of them. For example, in place of the symbol $\sqrt{\quad}$, we use SQRT ( ) or ( )**.5 enclosing the argument in parentheses. The latter expression indicates exponentiation, as do the forms X**2 and Y**2, which correspond respectively to X^2 and Y^2. The general form of exponentiation is

(E1)**(E2)

which represents the expression E1 raised to the E2 power. The equality symbol, =, is used in a different sense from the customary mathematical meaning in that even expressions such as

X = X + 1 or U = Z = Y + Z**2

are valid. The special meaning of equality is: EVALUATE the expression to the <u>right</u> of the equality symbol, using values of the quantities that currently exist in internal storage, then record the resulting value in internal storage as the existing value of the function or variable whose name (or names) appear to the <u>left</u> of the equality symbol. In the example, U = Z = Y + Z**2 the computer will first square the existing value of Z, add this result to the existing value of Y and then record this result in internal storage as the existing value of U and Z; the previous values of Z and U are erased just prior to the recording of the new value.

On line 4 of FIGURE 4 appear two distinct statements with the special symbol, %, used to separate them. One may write many statements or symbols on a given line in the space provided. Each line of the coding form is punched on a single card. Three lines (or 3 cards) would have sufficed for EXAMPLE 1. as illustrated below:

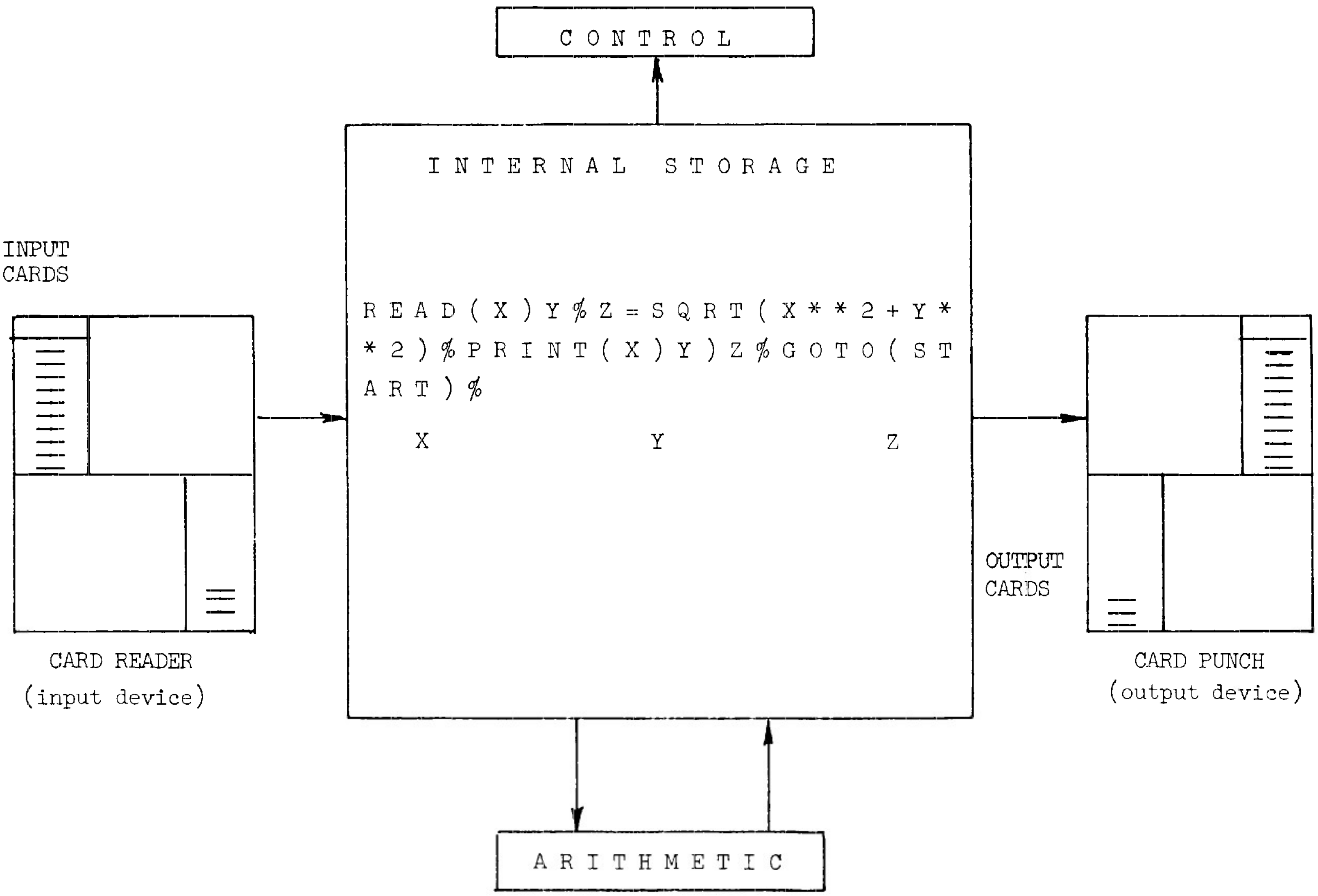
```
          PROB   C906                                                      1
START            READ(X)Y % Z = SQRT(X**2 + Y**2)% PRINT(X)Y)Z% GOTO(START)  2
          END    GOTO(START)                                               3
```

The rules for the use of parenthesis will be explained later. The last card carries the word END recorded in the ORDER TYPE space. This card serves two purposes:

1.) It signifies to the FORAST compiler the end of the translation of the FORAST program into primitive machine language. The equivalent of this card is not recorded in internal storage.

2.) The GOTO(   ) statement recorded on this card directs the computer to the first statement of the program to be executed. The first statement to be executed need not follow the PROB card, it may be any statement in the program.

The GOTO(START) statement on line 2 is recorded in internal storage and directs the computer to process every data pair following the first pair. The GOTO(START) on line 3 directs the computer to commence the program with the reading of the first data pair. It is not generally true that the processing of subsequent cases begins with the first instruction of every program.



```
READ(X)Y%Z=SQRT(X**2+Y*
*2)%PRINT(X)Y)Z%GOTO(ST
ART)%
```

FIGURE 7.

Illustrated in FIGURE 7 are the machine components used to obtain the solution of our problem. Note that the cards shown in FIGURE 6 are first inserted in the CARD READER, (input-device). The FORAST compiler causes the five (5) program cards to be read and the information on them is recorded in internal storage as illustrated. When the compiler recognizes the card with END punched in columns 7-10, (the 5th card), it directs the control unit to commence executing the instructions that have been recorded in internal storage. Note also that in addition to utilizing space for the instructions in internal storage, space is also reserved in internal storage for the numerical values of X, Y, and Z. Each time the computer carries out the instruction, READ(X)Y %, the numerical values on the card at the read station are transmitted and recorded in internal storage, the card itself being moved to the storage bin. Next, using the existing values of X and Y currently in internal storage, the computer evaluates $Z = \sqrt{X^2 + Y^2}$ and records the numerical value thus obtained in internal storage. To carry out the next instruction, PRINT (X)Y)Z %, the control unit activates the CARD PUNCH and records on a card the numerical values of X, Y, and Z that currently exist in internal storage. This completes the required processing for a given data pair, (X,Y), and then the instruction GOTO(START) directs the computer to READ the "next" data pair and process it in the same manner as the previous pair. After all of the cards in the CARD READER have been read and processed, the computer stops and one obtains the results in the form of the punched cards from the bin of the card punch. A few of these output cards are shown in FIGURE 8.

A tabulation of the FORAST program, the input data, and the results obtained are shown in FIGURE 9.

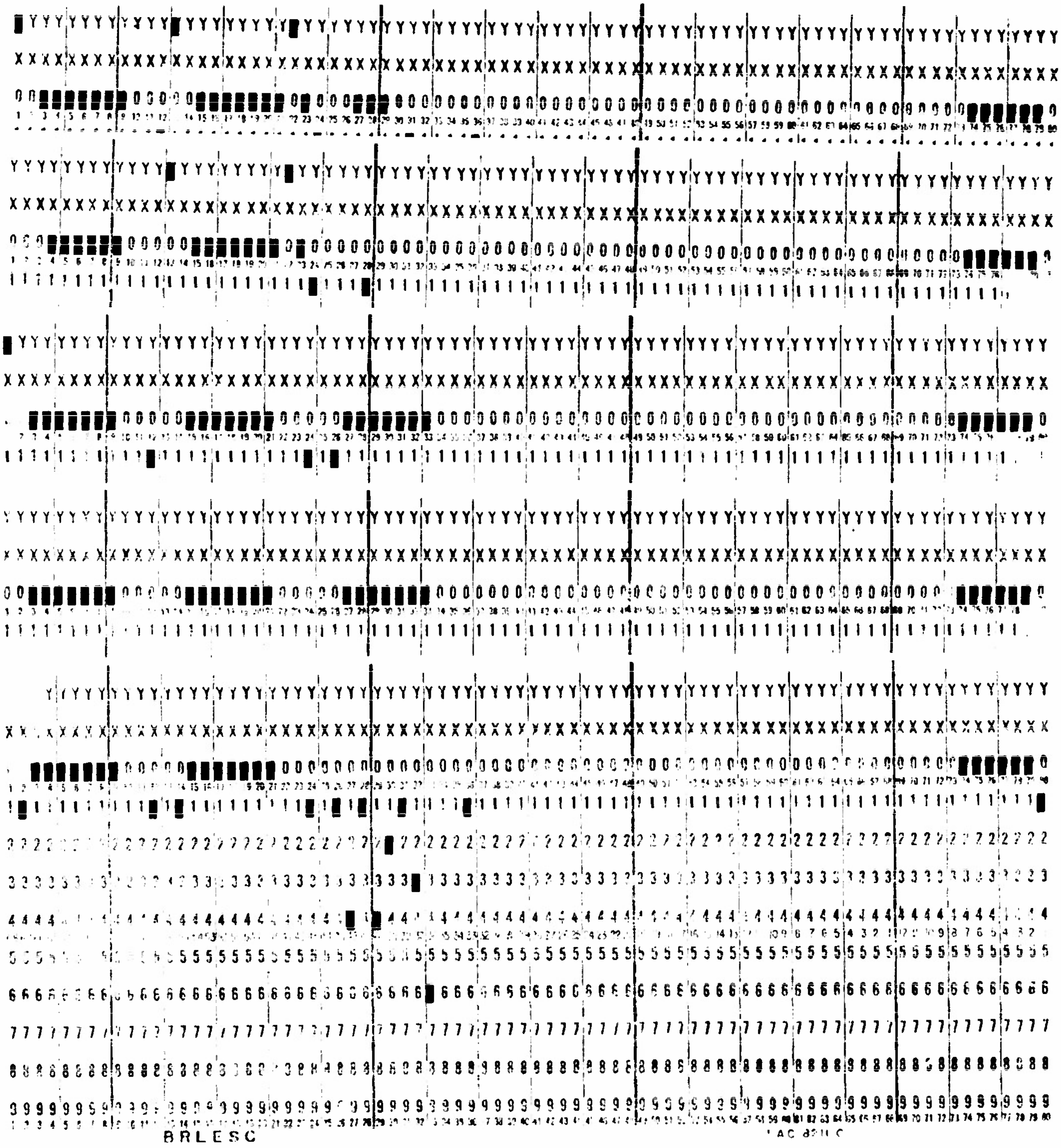
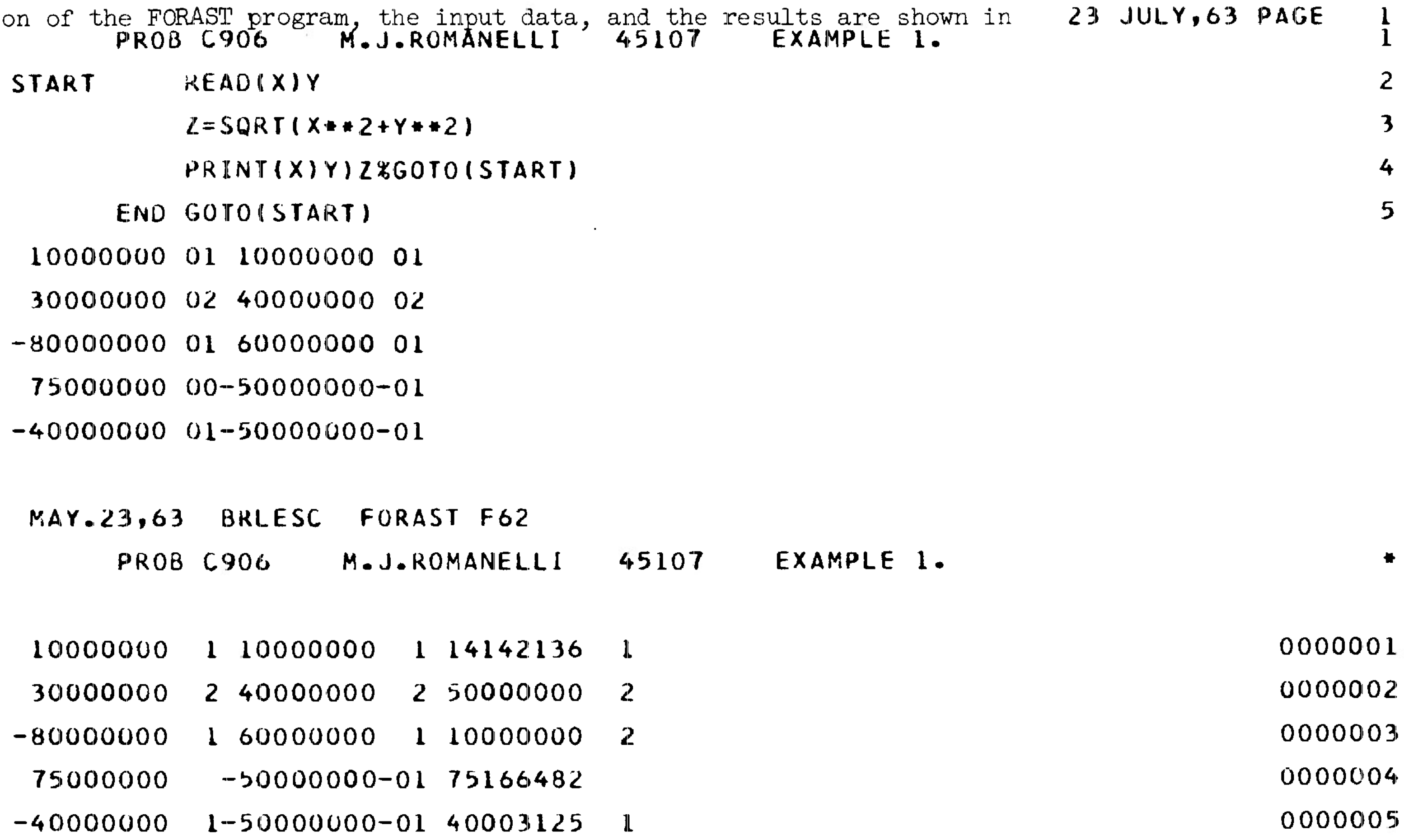


FIGURE 8.

A tabulation of the FORAST program, the input data, and the results are shown in FIGURE 9.



```
                                                                          23 JULY,63 PAGE   1
            PROB C906    M.J.ROMANELLI   45107    EXAMPLE 1.                                1
   START      READ(X)Y                                                                      2
              Z=SQRT(X**2+Y**2)                                                             3
              PRINT(X)Y)Z%GOTO(START)                                                       4
          END GOTO(START)                                                                   5
    10000000 01 10000000 01
    30000000 02 40000000 02
   -80000000 01 60000000 01
    75000000 00-50000000-01
   -40000000 01-50000000-01

    MAY.23,63  BRLESC  FORAST F62
            PROB C906    M.J.ROMANELLI   45107    EXAMPLE 1.                                *

    10000000  1 10000000  1 14142136  1                                               0000001
    30000000  2 40000000  2 50000000  2                                               0000002
   -80000000  1 60000000  1 10000000  2                                               0000003
    75000000   -50000000-01 75166482                                                  0000004
   -40000000  1-50000000-01 40003125  1                                               0000005
```

FIGURE 9.

EXAMPLE 2.

To illustrate a convenient means of identifying the "output" produced by the computer, we consider an additional requirement in the problem of EXAMPLE 1. In particular, we require that the letter X be printed (approximately centered) above the columns corresponding to the numerical values of X, and similarly the letters Y and Z centered over the columns that correspond to their respective numerical values.

To provide this facility, FORAST accepts a PRINT statement wherein we enclose in "special quotes" the literal characters that are to be printed. The "special quotes" are the less than and greater than symbols, < > . The PRINT statement takes the form

PRINT < >

and all characters within the < and > symbols are recorded on the output card. For our purpose we write:

PRINT < X Y Z >

Since the numerical values of X, Y, and Z require 12 characters in the standard form of output that we obtained, some spacing between the letters enclosed within the < > is necessary if we desire that the letters be approximately centered over their respective columns. Hence if we choose to have the letter X over column 6, Y over column 18, and Z over column 30, we indicate this spacing by writing

PRINT < 5b X 11b Y · 11b .Z >

where the circled quantities, 5b , 11b , and 11b indicate to a key punch operator and hence to the computer the number of "blanks" between the literal characters. (This is the only place in the FORAST language where "blank" characters are not ignored! i.e., blank characters are ignored everywhere except between the < and > symbols of PRINT statements). Knowing how to achieve this convenient identification, our immediate problem now is to determine how to "fit" this requirement into the original program of EXAMPLE 1. More specifically, if we recorded the above PRINT statement on a card, what is its logical place in the original program? If we inserted it between cards 3 and 4, i.e., in front of the card that instructs the computer to PRINT the numerical values of X, Y, and Z, we would obtain the printing of the literal letters X, Y, and Z each time the PRINT < X Y Z > was encountered. FORAST instructions are executed in sequence one after another unless a specific instruction directs the computer to do otherwise. Since we require the identification to be printed only once, a logical place for the identifying PRINT statement is at the very beginning of the program. Consequently the following program will produce the desired results.

```
                PROB    C906   M.J. ROMANELLI    EXAMPLE 2.       1
ADDREQ                  PRINT < 5b  X  11b  Y  11b  Z  >          1.1
START                   READ(X)Y                                  2
                        Z = SQRT(X**2 + Y**2)                     3
                        PRINT(X)Y)Z %  GOTO(START)                4
                END     GOTO(ADDREQ)                              5
```

Note that card 5 differs from the original in that instead of directing the computer to begin by reading a card at START, it directs the computer to begin at ADDREQ where it is instructed to print a card with the desired X Y Z heading. The above program and the results produced are listed in FIGURE 10.

```
                                                             23 JULY,63 PAGE   1
      PROB C906  M.J. ROMANELLI    EXAMPLE 2.                                  1
ADDREQ    PRINT<     X           Y           Z>                             1.1
START     READ(X)Y                                                             2
          Z=SQRT(X**2+Y**2)                                                    3
          PRINT(X)Y)Z% GOTO(START)                                             4
      END GOTO(ADDREQ)                                                         5
 10000000 01 10000000 01
 30000000 02 40000000 02
-80000000 01 60000000 01
 75000000 00-50000000-01
-40000000 01-50000000-01

 MAY.23,63  BRLESC  FORAST F62
      PROB C906  M.J. ROMANELLI    EXAMPLE 2.                                  *

     X           Y           Z                                           0000001
 10000000   1 10000000   1 14142136   1                                  0000002
 30000000   2 40000000   2 50000000   2                                  0000003
-80000000   1 60000000   1 10000000   2                                  0000004
 75000000    -50000000-01 75166482                                       0000005
-40000000   1-50000000-01 40003125   1                                  0000006
```

FIGURE 10.

NAMES AND OPERATIONS

EXAMPLE 1 & 2 illustrated the substitution, evaluation, internal-external recording and controls required to obtain a high-speed digital computer solution of a simple problem. Common to each of these processes is the fundamental requirement for identifying particular quantities and operations of interest. FORAST uses names constructed from a restricted set of symbolic characters. In EXAMPLE 1, we used the symbolic names X and Y to identify variables, Z to identify a particular function, SQRT another function, + and ** denoted particular arithmetic operations, READ and PRINT as names of recording operations, and START to describe an instruction.

To avoid ambiguities in the definition of names, operations, and numerical values, the 49 characters of the FORAST language are divided into two classes:

Class I	. ' A B C · · · Z	0 1 2 · · · 9	38 characters
Class II	+ - * / ( ) = % ,		11 characters

The characters of Class II have special meanings and must not be used to define non-indexed names of variables, functions, statements or other parameters.

+ is used to denote addition

- is used to denote subtraction

* is used to denote multiplication

/ is used to denote division

% is used to denote end of statement

= is used to denote (equality) evaluation

, is used to denote an indexed (subscripted) name

** is used to denote exponentiation

< is used to denote "less than"

> is used to denote "greater than"

( ) are used to denote multiplication, enclose arguments, indicate order of operations, separate names, parameters, numerical values, etc.

%% is used to denote that the information on a card following the %% is to be ignored by the computer.

Non-indexed symbolic names of variables, functions, statements and parameters may be constructed by combining characters of Class I under the following restrictions:

1. The leading character must not be zero and at least one character must be other than a decimal digit.
2. Non-indexed names must not exceed 6 characters in length unless those after the first 6 are not required for unique identification.
3. The special names such as SELF, SIN, COS, READ, PRINT, etc., have been reserved and should not be used as arbitrary names. (A complete list of reserved names is given on page 69 of [1.] .
4. Names of indices should not exceed three characters in length, 4 are permitted only if the rightmost 2 are decimal digits.

Despite the above restrictions, considerable freedom in the choice of names is permitted as illustrated in the examples given below:

X
Y8
17A
1X
X1
X15A
F300
EVALYS
FOFX

X'
XDOT
XBAR
START
SAM
WORK
N.Y.
EPS
FOFY

BOX 1.
1BOX
YACC
VELZ
FOURTH
Y"
L.A.
RHO
FOFZ

X3VEL.
Z"
DZ"
8.9
10.2.4
A.'B'
FLA.
ATPRES
FOFZ"

LAST
SINA
TANX
BETA2
9GAMMA
DELTAX
TEMPO
DENSTY
FINIS

Many problems require the processing of groups or arrays of numerical data, particularly discrete functions, vectors, matrices, etc. Conventional mathematical notation provides identification by subscripts or indices. For example, X_i, (i = 1,2,...,n) ; Y_j, (j = 0,2,4,...,m) ; Z_1, Z_2, Z_3, ... Z_k.

In the FORAST language, we use the special character, comma, to denote an indexed name; i.e.,

X_i is represented as X,I

Y_j is represented as Y,J

Z_k is represented as Z,K

As is conventional in mathematical notation, the indexed names take on specific meanings which depend on the name before the comma and the *existing value of index* whose symbolic name appears after the comma. Flexible and convenient means for setting and manipulating index values is provided; however, it is important to recognize that *the internal form of representation for index values differs from the floating-point representation* discussed previously. Index values are restricted to a limited range of integers and consequently *combining index values and floating-point values is not permissible*. The language does provide for transforming from one form to another. The convenient use of index names and means for establishing and generating desired values for them will be illustrated in later examples. The index names were introduced here briefly to indicate the general means of identification by symbolic name permitted in the FORAST language.

We introduce next the means of denoting the fundamental arithmetic operations of addition, subtraction, multiplication and division, which are denoted by the special characters, + - * and / respectively. Hence, the sum, difference, product and quotient of two quantities, A and B, are expressed as follows:

```
A + B
A - B
A * B
A / B
```

The special symbol, *, is used to denote multiplication to distinguish the product A times B from the single quantity, AB. Other means of denoting a product are

```
          (A)(B)
    or     A)(B)
    or     A)B
    or     A(B
```

or even the redundant symbol, *, is permitted in the above forms. In general, parentheses may be used to denote multiplication or to group operations in any desired sequence. In the absence of parentheses, particularly in ambiguous expressions, priority rules govern the sequence of operations.

When the order of operations is not specified, the established priorities are as follows:

1.) Single-valued functions of one argument;
2.) Exponentiation
3.) Multiplication and Division
4.) Addition and Subtraction

Next, we introduce the means to denote evaluation of the convenient elementary functions such as sine, cosine, square root, etc. These functions are denoted by special names as is conventional in mathematics. For example:

sine is denoted by SIN
cosine is denoted by COS
$\sqrt{\quad}$ is denoted by SQRT .

We classify these and similar functions as single-valued functions of a single-argument. (This is not to be interpreted as a single-valued function of one variable, on the contrary, the argument may be a function of many variables and indeed many other functions. We emphasize the classification of this set of convenient functions to distinguish them from another set which includes functions of more than one argument).

EXAMPLES:

`X = Y + Z * W` implies $x = y + zw$

since multiplication has priority over addition.

`X = Y - Z/W` implies $x = y - \frac{z}{w}$

since division has priority over subtraction.

`X = Y + Z ** 2` implies $x = y + z^2$

since exponentiation has priority over addition.

`X = SIN(Y+Z)**3` implies $x = \sin^3(y+z)$

since single-valued functions have priority over exponentiation.

EXAMPLE 3.

This example is given to illustrate the single-valued functions of single-arguments that are available in the FORAST programming language.

GIVEN:

Numerical values of two variables, x and y. Assume that they are recorded in standard floating-point form on a punched card. For simplicity, we will let $x = 1$ and $y = 2$.

REQUIRED:

Compute the following functions; identify and record each function on punched cards.

$A = \sqrt{y}$

$B = \sin x$

$C = \sin xy$

$D = \sin 4x$

$E = \sin 5x$

$F = \sin (-x)$

$G = \cos x$

$H = \cos (x + \pi)$

$I = \ln(y + \sqrt{x})$

$J = \log_{10}(y)$

$K = e^{x + y}$

$L = \arctan (x - y)$

$M = \arctan (y)$

$N = \text{arccot} (-x)$

$O = \text{arccot} (y)$

$P = \arccos (x/y)$

$Q = \arccos (-x/y)$

$R = \arcsin x/5$

$S = \arcsin (-x)$

$T = \tan xy$

$U = \tan (-x)$

$V = \cot (y - x)$

$W = \cot (-y)$

$FX = \sec x$

$FY = \csc y$

F1 = sign of x

F2 = sign of (-x)

F3 = sign of (y - y)

$F4 = \sinh x$

$F5 = \cosh x$

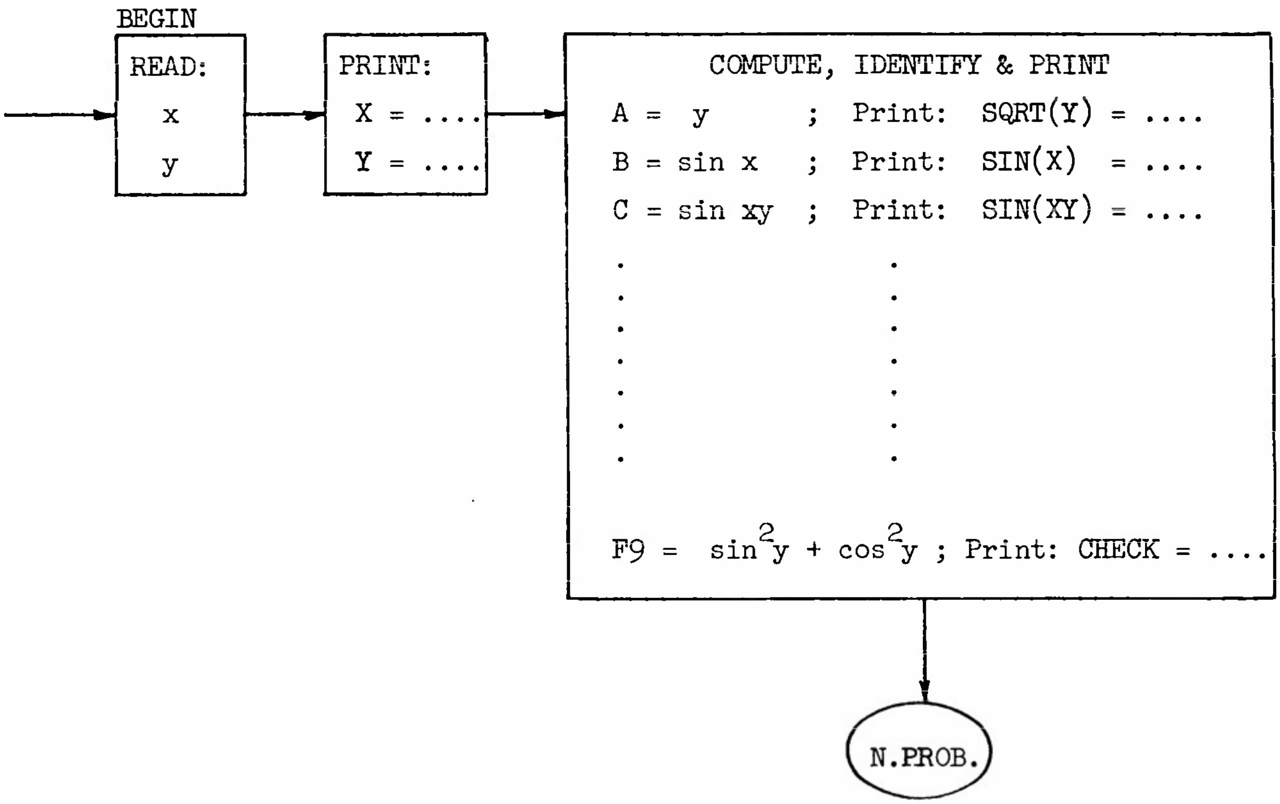
$F6 = \tanh x$

F7 = integer part of A

F8 = fractional part of A

$F9 = \sin^2 y + \cos^2 y$

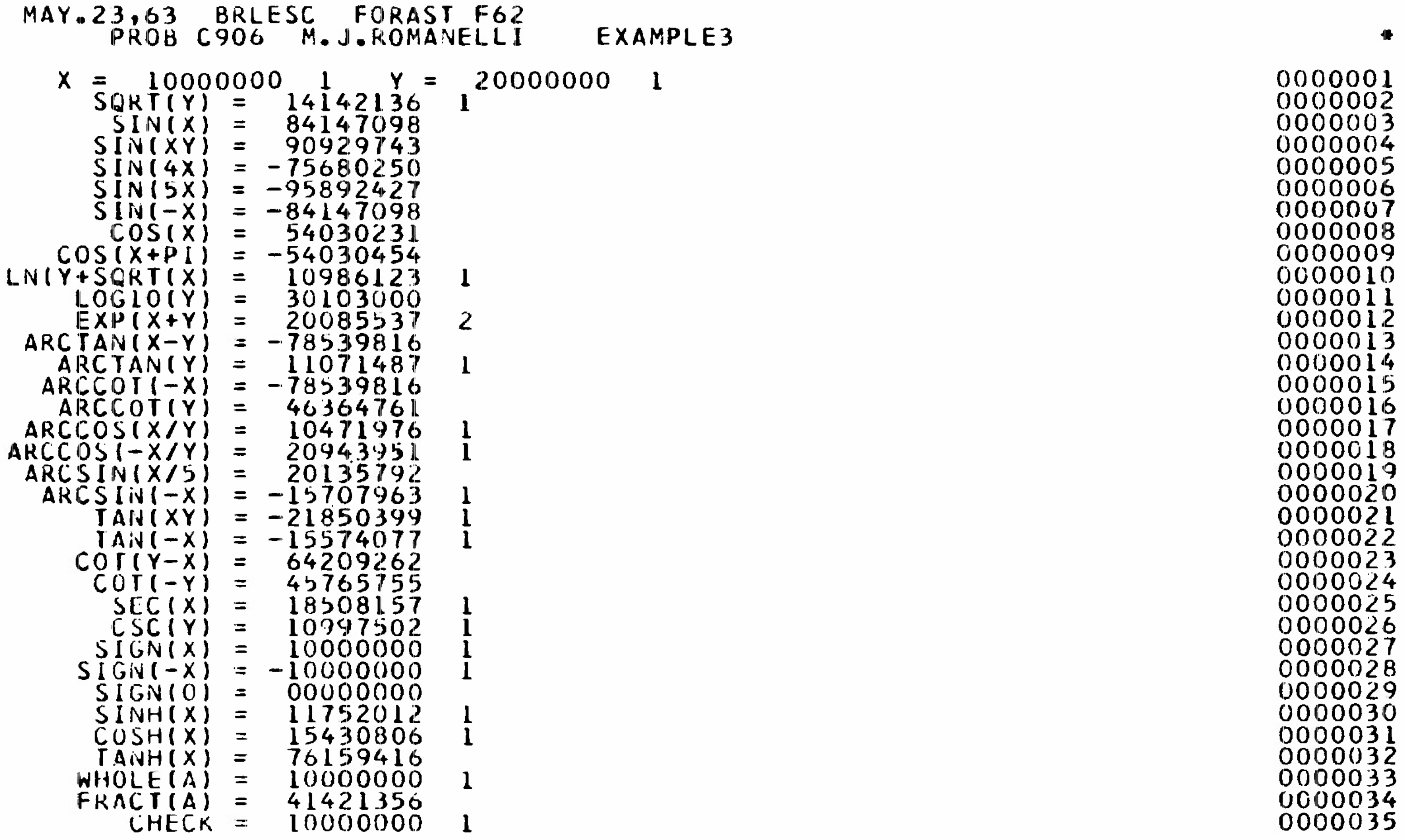
The program, input and output for this example are listed in FIGURE 12.


BEGIN
READ:
x
y
PRINT:
X = ...
Y = ...
COMPUTE, IDENTIFY & PRINT
A = y ; Print: SQRT(Y) = ...
B = sin x ; Print: SIN(X) = ...
C = sin xy ; Print: SIN(XY) = ...
F9 = $\sin^2 y + \cos^2 y$; Print: CHECK = ...
N.PROB.

FLOW CHART FOR EXAMPLE 3.

FIGURE 11.

```
                                                            23 JULY,63 PAGE    1
      PROB C906  M.J.ROMANELLI    EXAMPLE3

      COMM SINGLE-VALUED FUNCTIONS OF ONE ARGUMENT
BEGIN     READ(X)Y
          PRINT<   X = >X<    Y = >Y
          A=SQRT(Y)% PRINT<     SQRT(Y) = >A
          B=SIN(X)% PRINT<      SIN(X) = >B
          C=SIN(X*Y)% PRINT<     SIN(XY) = >C
          D=SIN(4*X)% PRINT<     SIN(4X) = >D
          E=SIN(X*5)% PRINT<     SIN(5X) = >E
          F=SIN(-X)% PRINT<     SIN(-X) = >F
          G=COS(X)% PRINT<      COS(X) = >G
          H=COS(X+3.14159)% PRINT<   COS(X+PI) = >H
          I=LOG(Y+SQRT(X)% PRINT<LN(Y+SQRT(X) = >I
          J=LOG10(Y)% PRINT<    LOG10(Y) = >J
          K=EXP(X+Y)% PRINT<    EXP(X+Y) = >K
          L=ARCTAN(X-Y)% PRINT< ARCTAN(X-Y) = >L
          M=ARCTAN(Y)% PRINT<   ARCTAN(Y) = >M
          N=ARCCOT(-X)% PRINT<  ARCCOT(-X) = >N
          O=ARCCOT(Y)% PRINT<   ARCCOT(Y) = >O
          P=ARCCOS(X/Y)% PRINT< ARCCOS(X/Y) = >P
          Q=ARCCOS(-X/Y)% PRINT<ARCCOS(-X/Y) = >Q
          R=ARCSIN(X/5)% PRINT< ARCSIN(X/5) = >R
          S=ARCSIN(-X)% PRINT<  ARCSIN(-X) = >S
          T=TAN(X*Y)% PRINT<     TAN(XY) = >T
          U=TAN(-X)% PRINT<     TAN(-X) = >U
          V=COT(Y-X)% PRINT<    COT(Y-X) = >V
          W=COT(-Y)% PRINT<     COT(-Y) = >W
          FX=SEC(X)% PRINT<      SEC(X) = >FX
          FY=CSC(Y)% PRINT<      CSC(Y) = >FY
          F1=SIGN(X)% PRINT<     SIGN(X) = >F1
          F2=SIGN(-X)% PRINT<    SIGN(-X) = >F2
          F3=SIGN(Y-Y)% PRINT<     SIGN(0) = >F3
          F4=SINH(X)% PRINT<     SINH(X) = >F4
          F5=COSH(X)% PRINT<     COSH(X) = >F5
          F6=TANH(X)% PRINT<     TANH(X) = >F6
          F7=WHOLE(A)% PRINT<    WHOLE(A) = >F7
          F8=FRACT(A)% PRINT<    FRACT(A) = >F8
          F9=(SIN(Y)**2+COS(Y)**2)**.5% PRINT<       CHECK = >F9
          GOTO(N.PROB)
      END GOTO(BEGIN)
 10000000 01 20000000 01
```



```
 MAY.23,63  BRLESC  FORAST F62
      PROB C906  M.J.ROMANELLI    EXAMPLE3              *

   X =  10000000  1   Y =  20000000  1            0000001
     SQRT(Y) =  14142136  1                       0000002
      SIN(X) =  84147098                          0000003
     SIN(XY) =  90929743                          0000004
     SIN(4X) = -75680250                          0000005
     SIN(5X) = -95892427                          0000006
     SIN(-X) = -84147098                          0000007
      COS(X) =  54030231                          0000008
   COS(X+PI) = -54030454                          0000009
LN(Y+SQRT(X) =  10986123  1                       0000010
    LOG10(Y) =  30103000                          0000011
    EXP(X+Y) =  20085537  2                       0000012
 ARCTAN(X-Y) = -78539816                          0000013
   ARCTAN(Y) =  11071487  1                       0000014
  ARCCOT(-X) = -78539816                          0000015
   ARCCOT(Y) =  46364761                          0000016
 ARCCOS(X/Y) =  10471976  1                       0000017
ARCCOS(-X/Y) =  20943951  1                       0000018
 ARCSIN(X/5) =  20135792                          0000019
  ARCSIN(-X) = -15707963  1                       0000020
     TAN(XY) = -21850399  1                       0000021
     TAN(-X) = -15574077  1                       0000022
    COT(Y-X) =  64209262                          0000023
     COT(-Y) =  45765755                          0000024
      SEC(X) =  18508157  1                       0000025
      CSC(Y) =  10997502  1                       0000026
     SIGN(X) =  10000000  1                       0000027
    SIGN(-X) = -10000000  1                       0000028
     SIGN(0) =  00000000                          0000029
     SINH(X) =  11752012  1                       0000030
     COSH(X) =  15430806  1                       0000031
     TANH(X) =  76159416                          0000032
    WHOLE(A) =  10000000  1                       0000033
    FRACT(A) =  41421356                          0000034
       CHECK =  10000000  1                       0000035
```

FIGURE 12.

Restrictions on the single-valued functions of one argument illustrated in EXAMPLE 3 are:

1.) Every argument is enclosed in parentheses, the closing right parenthesis is optional.

2.) Every argument and every result is in floating-point form.

3.) The arguments of trigonometric functions and results of inverse trigonometric functions are in radians.

4.) The results of ARCTAN and ARCSIN are in the interval

$$-\frac{\pi}{2} \leq \text{Result} \leq \frac{\pi}{2}$$

5.) The results of ARCCOT and ARCCOS are in the interval

$$0 \leq \text{Result} \leq \pi \quad .$$

6.) The arguments may be functions of many variables and functions; however, only the sign of the resulting argument is considered in the evaluation of the particular function.

Although not illustrated, names of variables, arguments and results may be indexed. Violations of restricted bounds of arguments will be detected and recorded during the execution of the program.

EXAMPLE 4.

EXAMPLE 3. illustrated the available functions of one argument. Another set of convenient functions are those which are functions of more than one argument and often produce more than one result. Included in this set are methods for interpolation, quadrature, matrix operations, solution of ordinary differential equations, etc. To instruct the computers to evaluate functions in this set we write a statement of the form,

ENTER (Name of particular function) N1)N2)......%

where the word ENTER followed by a particular name enclosed in parentheses identifies the particular function or method of interest. The N1,N2,..... represent the names or values of the arguments, parameters and the defining names of the results. Many of the available functions in this set will be illustrated in separate examples.

To introduce one of these functions, we will consider an extension of EXAMPLE 1. Assume that the given pairs (x,y) represent the rectangular coordinates of a point in a plane and that we want to produce a table which includes the corresponding polar coordinates of the point, i.e.,

GIVEN: x_i, y_i , $i = 1,2,3, \ldots$

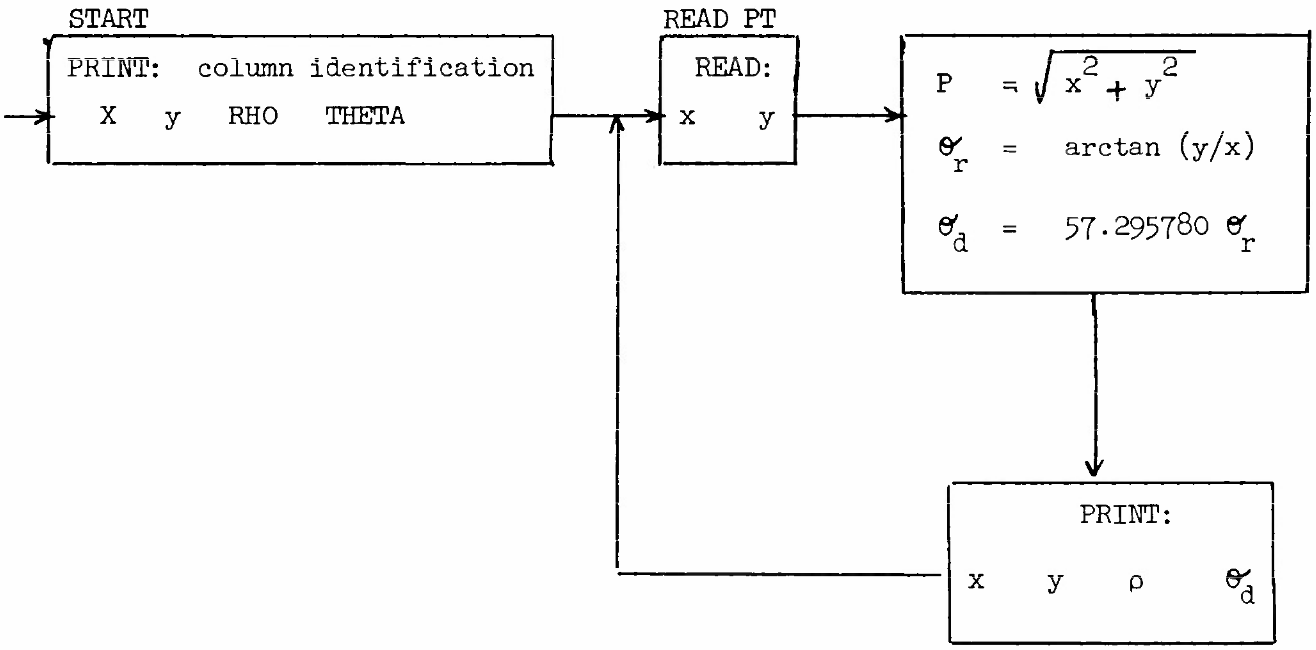
REQUIRED: Corresponding to each point, (x,y) determine, identify and list in a table the polar coordinates ρ_i and θ_i (list θ in degrees).

$$\rho_i = \sqrt{X_i^{\,2} + y_i^{\,2}}$$

$$\theta_i = \arctan (y_i/x_i)$$

Before constructing the flow-chart and writing the program which will produce the desired solution we emphasize the distinction between the available arctan function of one-argument illustrated in EXAMPLE 3, and the arctan function of two-arguments to be used and illustrated in this example. The distinction is that the arctan function of one-argument produces a result which lies in quadrants I or IV. (Note 4. on page 38, $-\frac{\pi}{2} \leq$ Result $\leq \frac{\pi}{2}$). Since the arctan function of two arguments take into account the signs of both arguments, it produces a result which may be in quadrants I, II, III or IV, i.e.,

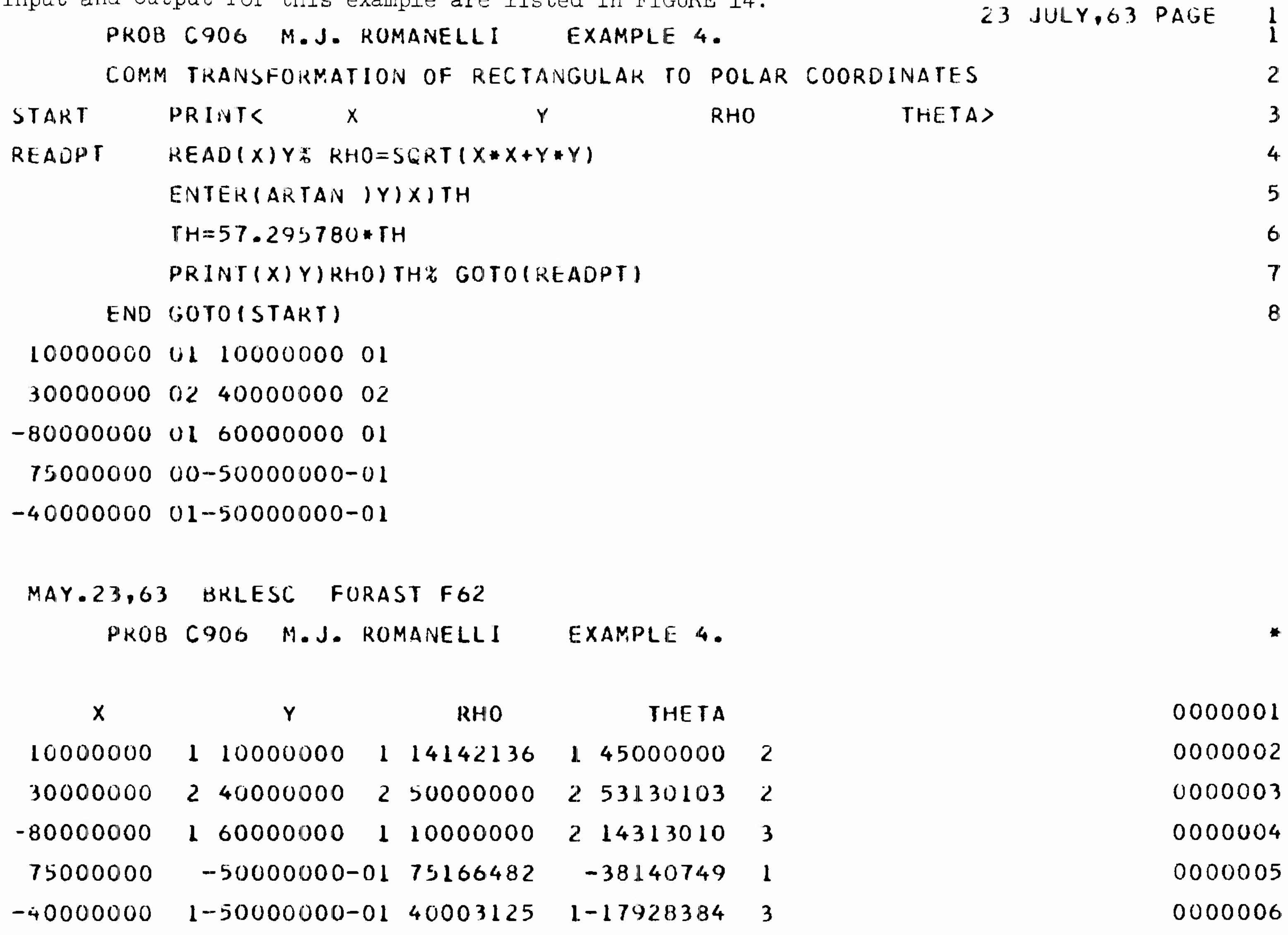
$$-\pi \leq \text{Result} \leq \pi \text{ .}$$


START
PRINT: column identification
X y RHO THETA
READ PT
READ:
x y
$P = \sqrt{x^2 + y^2}$
θ_r = arctan (y/x)
θ_d = 57.295780 θ_r
PRINT:
x y ρ θ_d

FLOW CHART FOR EXAMPLE 4.

FIGURE 13.

The program, input and output for this example are listed in FIGURE 14.



```
                                                                    23 JULY,63 PAGE    1
          PROB C906  M.J. ROMANELLI    EXAMPLE 4.
          COMM TRANSFORMATION OF RECTANGULAR TO POLAR COORDINATES
START       PRINT<      X            Y            RHO           THETA>
READPT      READ(X)Y% RHO=SQRT(X*X+Y*Y)
            ENTER(ARTAN )Y)X)TH
            TH=57.295780*TH
            PRINT(X)Y)RHO)TH% GOTO(READPT)
          END GOTO(START)
 10000000 01 10000000 01
 30000000 02 40000000 02
-80000000 01 60000000 01
 75000000 00-50000000-01
-40000000 01-50000000-01

 MAY.23,63   BRLESC   FORAST F62
          PROB C906  M.J. ROMANELLI    EXAMPLE 4.                                       *

    X             Y             RHO           THETA                          0000001
 10000000    1 10000000    1 14142136    1 45000000    2                     0000002
 30000000    2 40000000    2 50000000    2 53130103    2                     0000003
-80000000    1 60000000    1 10000000    2 14313010    3                     0000004
 75000000     -50000000-01 75166482     -38140749    1                      0000005
-40000000    1-50000000-01 40003125    1-17928384    3                      0000006
```

FIGURE 14.

EXAMPLE 5.

In the previous examples, the output results were recorded in standard floating-point form. In this example we will illustrate a means of obtaining the tabular results in a form with the decimal point actually punched (and printed) instead of the exponent. For example, instead of obtaining the printed value for $\sqrt{2}$ as 14142136 1, we will obtain the printed value for $\sqrt{2}$ as 1.414 . This example will illustrate a special case of the general PRINT statement where a departure from standard form is desired and must be specified accordingly.

To designate a non-standard form, the PRINT statement is written in the form

PRINT-FORMAT (Name of specifying form) - ( )) ···) %

where the "Name of specifying form", enclosed in parenthesis after the word FORMAT, identifies a quantity which explicitly defines forms and horizontal spacings desired. We refer to this specifying quantity as a "format word". To provide for arbitrary spacing of the output forms and for transformations from internal to external forms the format word is composed of parenthesized expressions of the form

(T - S - L)

where T, S, and L are decimal digits with defined meanings:

T generally specifies a type of transformation, a repetition of a previous format, spacing of quantities, end of format, etc. ;

S generally specifies a scale or relative location of a decimal point;

L generally specifies the length of a field, i.e., the number of columns or characters to be used in representing a given form.

The parenthesized expressions may contain all three of the above descriptors,

(T - S - L)

or only two descriptors, (T - L)

or only one descriptor , (T)

Similar expressions may be constructed for use with the general READ statement. The permissible values of T and their corresponding meanings are given in [1.] . Only a few of the most frequently used T's will be illustrated here.

In EXAMPLE 4. the entries in the output table were expressed in standard floating-point form; i.e., coefficients with corresponding exponents. To depart from the standard form and obtain numerical values with decimal points printed instead of exponents, we will specify (using a format word) the particular transformations desired, the relative location of the decimal-point, the number of columns to be used for each quantity and the spacing between individual quantities. In the standard form we used 12 columns for each quantity and provided for no spacing between them.

Assume that we want the following format for the table corresponding to the output of EXAMPLE 4.

```
    X           Y            RHO         THETA
± DDD.DDD   ± DDD.DDD   ± DDDD.DD   ± DDD.DDD
    "           "           "           "
    "           "           "           "
    "           "           "           "
```

where the D's represent decimal digits. That is, we will specify a total of 8 characters for each quantity and 4 blank spaces between each quantity. Note that in the total length, 8, we include the algebraic sign and decimal point in the count. (We chose four spaces between quantities since the column headings, X, Y, RHO and THETA were approximately centered over 12 column entries;

hence, with the choice of 8 characters for each quantity and 4 spaces between quantities, we will not have to alter the spacing of the characters of the column headings).

To specify this format we construct a format word which we arbitrarily label SEE1.

```
    LOC          O.T.                  FORMULAS, STATEMENTS, COMMENTS            IDEN.
 1      6     7      10      11                                                  77-80

  SEE1.          FORM       (3-2)12-3-8)3-4)12-3-8)3-4)12-4-8)3-4)12-3-8)2        7.1
```

Note that the arbitrary name chosen for the format word is written in the LOCATION columns and the word FORM in columns 7-10 of the coding sheet. The parenthesized expressions are written in columns 11-76.

The PRINT Statement that refers to this format word is written in the following form:

```
                     PRINT-FORMAT(SEE1.) - (X)Y)RHO)T %                    7'
```

The expressions (3-L) in the format word, SEE1., correspond to T = 3 which denotes that a spacing of L columns is desired. (Note that we begin with 2 spaces before the first quantity, X, and denoted spacings of 4 thereafter). The expression (12-3-8) corresponds to T = 12

S = 3

L = 8

and represents the form for the quantity, X.

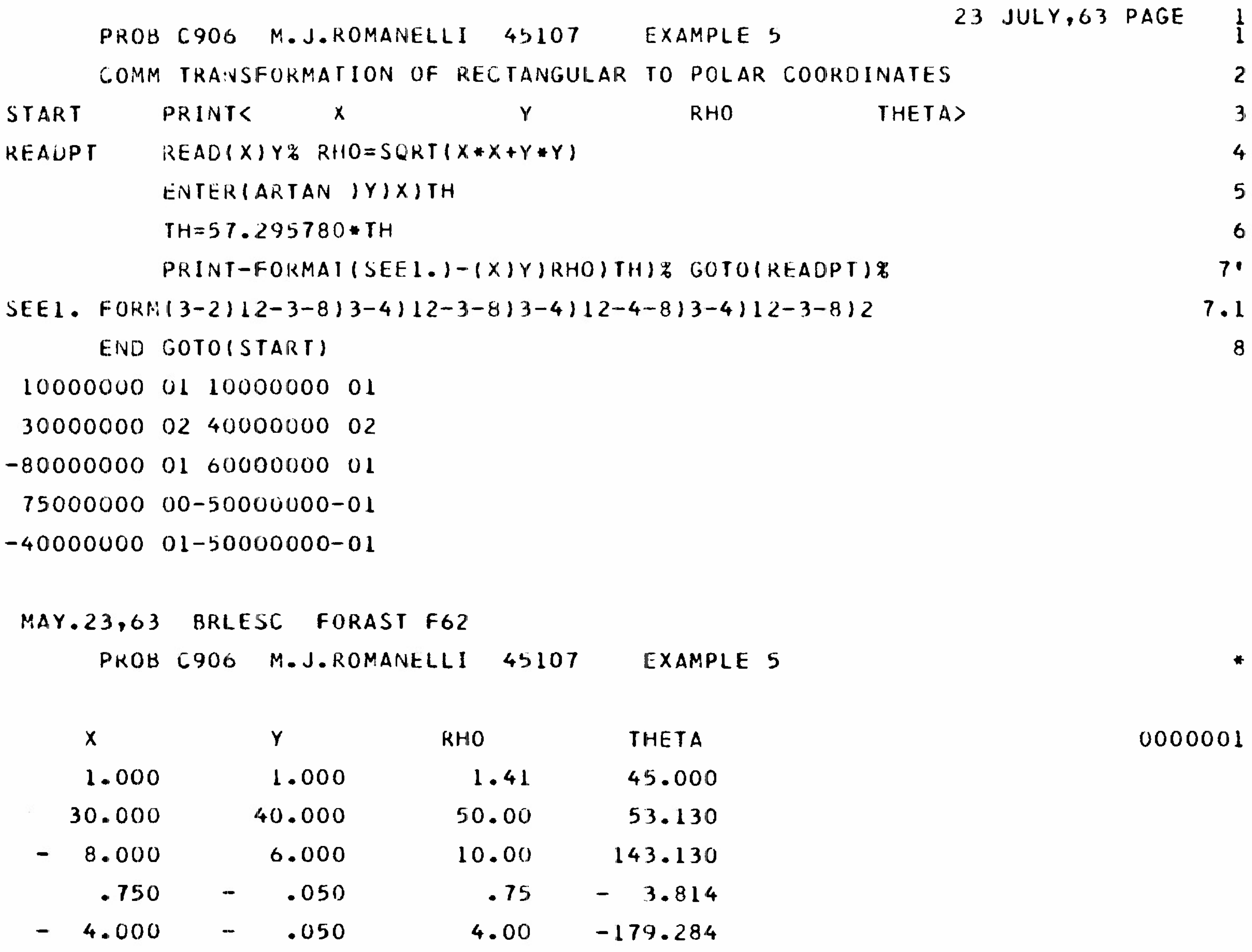
T = 12 denotes that the quantity exists internally as a floating-point number and the output form desired is a "fixed" form with the "fixed" decimal-point location specified by the corresponding S.

S = 3 denotes that the "fixed" decimal-point is to be recorded after the 3 digits that follow the algebraic sign.

L = 8 denotes that a total of 8 columns (or characters) are to be used for the quantity.

Note the identical forms for the quantities X, Y, and THETA and the decimal-point after 4 digits in the representation for RHO. The 2 after the last parenthesized expression corresponds to T = 2 and denotes the end of the format word.

To obtain the desired results, we will replace Card 7 of EXAMPLE 4. with the card identified as 7' above, (i.e., the PRINT Statement that refers to the specified format), and add the card identified as 7.1 that defines the specified format. The program, input and output are listed in FIGURE 15.



```
                                                                 23 JULY,63 PAGE    1
      PROB C906   M.J.ROMANELLI   45107    EXAMPLE 5                                1
      COMM TRANSFORMATION OF RECTANGULAR TO POLAR COORDINATES                       2
START      PRINT<     X           Y           RHO          THETA>                   3
READPT     READ(X)Y% RHO=SQRT(X*X+Y*Y)                                              4
           ENTER(ARTAN )Y)X)TH                                                      5
           TH=57.295780*TH                                                          6
           PRINT-FORMAT(SEE1.)-(X)Y)RHO)TH)% GOTO(READPT)%                         7'
SEE1. FORM(3-2)12-3-8)3-4)12-3-8)3-4)12-4-8)3-4)12-3-8)2                          7.1
      END GOTO(START)                                                               8
 10000000 01 10000000 01
 30000000 02 40000000 02
-80000000 01 60000000 01
 75000000 00-50000000-01
-40000000 01-50000000-01


 MAY.23,63  BRLESC  FORAST F62
      PROB C906   M.J.ROMANELLI   45107    EXAMPLE 5                                *

     X           Y           RHO          THETA                              0000001
     1.000       1.000        1.41       45.000
    30.000      40.000       50.00       53.130
 -   8.000       6.000       10.00      143.130
      .750   -    .050         .75   -    3.814
 -   4.000   -    .050        4.00     -179.284
```

FIGURE 15.

In addition to accepting FORMULA statements which involve addition, subtraction, multiplication, division, exponentiation and many single-valued functions of one-argument, FORAST also accepts eleven English word statements and twenty-two special English words. The latter are called "pseudo order-types" and are labelled "ORDER TYPE" over columns 7 through 10 of the standard coding form. Only 11 of the 22 permissible order types will be illustrated in this report. Included in the eleven English word statements is the general ENTER statement which provides for evaluations of approximately forty functions of more than one argument.

We have in the previous examples illustrated special cases from each of the above categories. Each example included at least one English word statement and at least one single-valued function of one argument. The READ, PRINT, GOTO and ENTER statements illustrated 4 of the 11 English word statements. PROB and END, (which are required in every problem), illustrated two of the special O.T. English words. Listed below are the 11 English word statements and the 11 order-types to be illustrated in this report.

ENGLISH WORD STATEMENTS	SPECIAL ENGLISH WORDS (O.T.)	
GOTO	PROB	(Problem
SET	END	
SETEA	DATE	
INC	COMM	(Comment)
COUNT	CONT	(Continue)
IF	FORM	(Format)
CLEAR	LIST	(Listing)
MOVE	BLOC	
ENTER	SYN	(Synonym)
READ, PRINT (PUNCH)	DEC	(Decimal)
HALT	DEC =	(Decimal Equality)

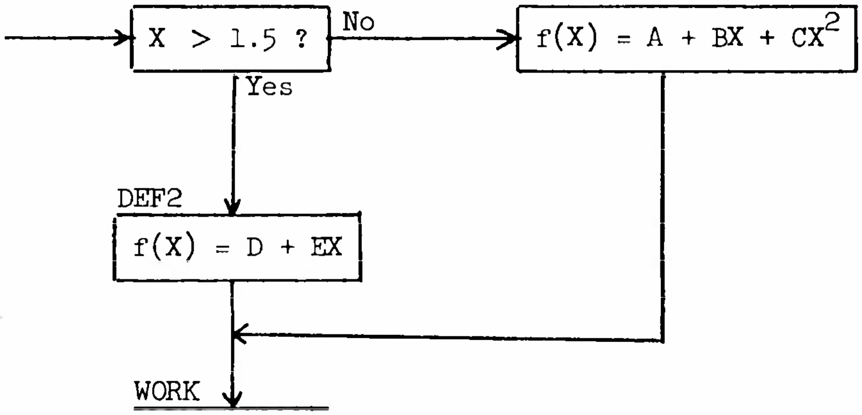
In general, the English word statements may appear anywhere in a program and are transformed to computer instructions that will be executed during the running of the program. The special English words are not transformed into computer instructions, the words themselves may only appear in columns 7 through 10. (Note that none of these names exceeds four characters). Rather than describe each of the above in their full generality, we will continue as in the previous examples to introduce a few in each example, with variations illustrated in succeeding examples.

Many problems require definitions of functions, or more generally, many problems require control of processes which are based on conditional relationships. For example, it may be necessary to evaluate

$$f(X) = A + BX + CX^2 \quad \text{if} \quad X \leq 1.5$$

$$f(X) = D + EX \quad \text{if} \quad X > 1.5$$

We assume that X may take on values in either range but as stated above f(X) is defined accordingly. Schematically, the above may be illustrated as follows:



We have arbitrarily labelled one of the definitions DEF2 and indicated that after the appropriate definition is applied (i.e., after the appropriate function is evaluated) the computer is to continue operations at a place called WORK. To instruct the computer in the FORAST language to carry out the above plan, we write:

```
LOC.    O.T.                FORMULAS, STATEMENTS, COMMENTS

                  IF(X >1.5)GOTO(DEF2)% FX = A + X(B + C * X)% GOTO(WORK)
DEF2              FX = D + E * X
WORK
```

The above is an illustration of a simple conditional statement. It has the form

```
                  IF (a specified condition is satisfied) GOTO(someplace)%
```

As indicated in the form, if the specified condition is satisfied the computer is directed to GOTO the someplace denoted for subsequent instructions, otherwise the computer will execute the statement immediately following the conditional statement. It is to be emphasized that the above is an illustration of a simple conditional statement, more general compound forms are permissible. The conditional expressions may contain < , > , and equality relations whose terms involve the arithmetic operations and single-valued functions of one argument. Indeed a single conditional statement may contain many conditional expressions separated by the logical operations AND or OR and each expression may be prefixed with ABS, NOT, INT, etc., where: ABS denotes absolute value; NOT denotes negation of the parenthesized relation, INT denotes that the quantities involved in the relation are integers in integer form, (not floating-point form). In the following example, we will illustrate the simple form of a conditional statement.

EXAMPLE 6.

GIVEN: A, B and N recorded in standard floating-point form in the first three fields of a punched card; (Assume several such cards.)

$f(X) = \sin X \quad ; \quad H = \frac{B - A}{N}$.

REQUIRED: Use the trapezoidal rule to obtain an approximation for the definite integral

$$I = \int_A^B f(X)dx$$

Trapezoidal rule:

$$\int_A^B f(X)dx = I$$

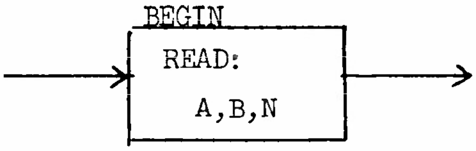
$$I = H\left[\frac{f(A)}{2} + f(A+H) + f(A+2H) + \cdots + f(B-H) + \frac{f(B)}{2}\right]$$

Print and identify: A; B; N and I .

Note that the trapezoidal rule requires that the function (integrand) be evaluated at (N+1) discrete values of X, the end values (A and B) are weighted 1/2, the interior values have unit weight.

To construct a flow chart that outlines a plan to obtain the desired solution we begin by denoting that we want the computer to read the card that contains the pertinent values A, B and N. Recall that once they have been read, they are recorded internally for as many future references as desired.

Hence to indicate that we want to begin by having the computer read a card, we write

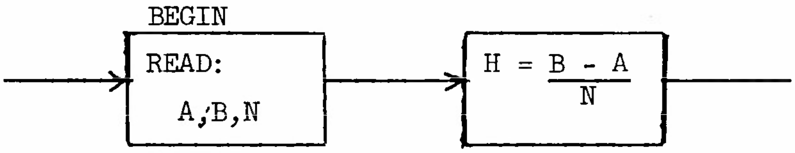


Next, we observe by studying the definition of the trapezoidal rule that we need the value H for two purposes:

1.) it is needed as a factor to obtain the final result after the weighted integrand values are summed;

2.) it is needed to construct the discrete interior values of X, i.e., A + H, A + 2H, A + 3H, etc.

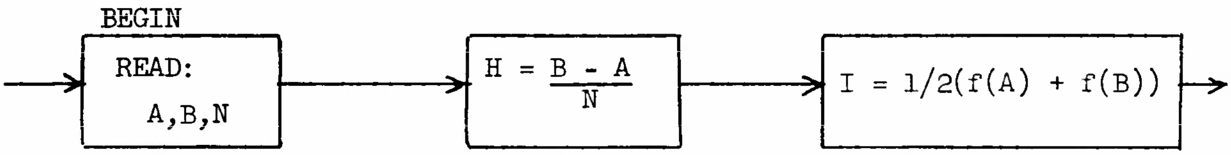
Hence, since we need H for the two purposes given above we append to the chart a box indicating the desired definition and evaluation of H.



Next, noting that the definition requires weights of 1/2 on the "end" terms, we can dispose of these by instructing the computer to evaluate the following function

$$I = 1/2(\sin A + \sin B) \quad .$$

We indicate this on the chart accordingly



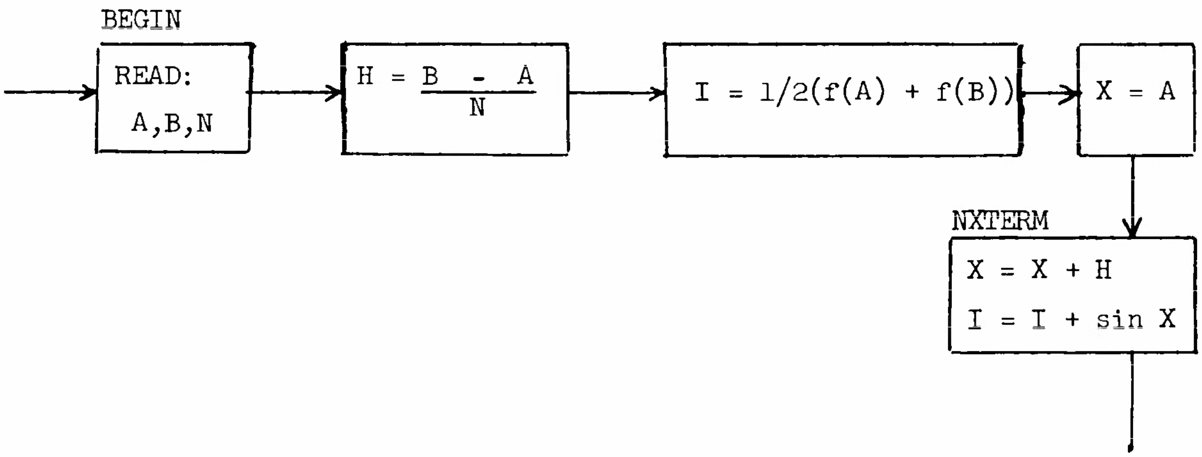
Now note that the remaining required arguments, A + H, A + 2H, etc., can be obtained in general by adding H to the previous argument. If we let X be an existing argument, the next argument can be obtained by adding H to X. Symbolically we can write this as

$$X = X + H$$

where it is understood that the X on the right of the equality represents an existing value which will be used to generate a "new" existing value. Hence we have generalized, in that anytime we want the computer to generate a new argument, we need only direct it to the generalized expression. Note also that each of these arguments will undergo "similar treatment", i.e., each is the argument for the function, sine. Hence we can generalize a two step process and write it in the form

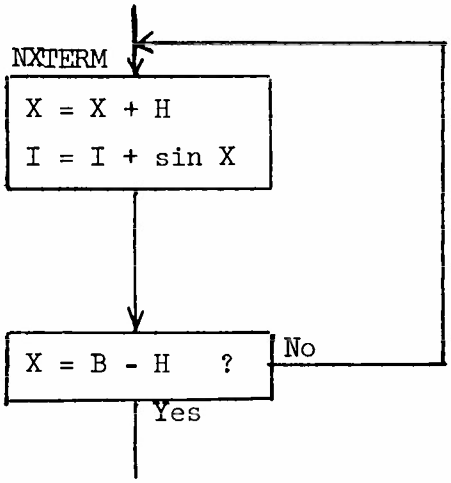
$$X = X + H \% \qquad I = I + SIN(X)$$

If we let X take on the initial value of A and direct the computer to the above two step process, it (the computer) would in the first step evaluate the function for existing values of X and H which result in a value of X = A + H. In the next step, the computer would add the sin(A + H) to the existing value of I. Hence, after the first execution of the two step process, I would then represent the sum of three terms of the bracket [ ], namely the first, second and last terms. To schematically illustrate the above, we augment the flow chart as follows:



We have arbitrarily labelled the generalized definition of arguments and the summing of the general term, NXTERM. We will want to make reference to this process and direct the machine to carry it out as many times as required. At this point we can apply a simple conditional statement.

If one refers to the definition of the Trapezoidal Rule, it will be noted that the "last term" we want the computer to add to the existing I is f(B - H), i.e., sin (B - H), (since we have already accounted for the term $\frac{f(B)}{2}$). Hence we need only instruct the computer to determine, after adding a term to the existing I, whether the corresponding argument was equal to B - H. If indeed X = B - H, all of the required terms have been evaluated and summed; otherwise, we can instruct the computer to "go back" to evaluate and add the *next term* to the existing I. Schematically this condition can be illustrated as follows:



The corresponding conditional statement in the FORAST language would be written as

```
IF(X = B - H)GOTO(FINISH)% GOTO(NXTERM)
```

however, since the computers use a *finite* number of places in the calculations the conditional statement written above may not be adequate for all values of

A, B and N. To illustrate this, suppose:

$$A = 0$$
$$B = 1$$
$$N = 3 \quad ,$$

then, $H = 1/3$ and the equivalent internal representation = $.333\cdots3$; similarly, $B - H = 2/3$ and the equivalent internal representation = $.666\cdots7$ (The internal representations result from conventional rounding and truncation procedures). The X argument, "corresponding" to B - H, generated by the specified definition would be $A + H + H = .666\cdots6$! Note the "6" in the least significant digit. Hence this value is not equal to the $.666\cdots7$, i.e., $.666\cdots6 \neq .666\cdots7$. In conditional expressions, equality relations are not satisfied unless the resulting numerical values on both sides of the equality have identical representations. Hence, we should relax the stringent equality condition and ask if X differs from B - H by a "tolerable" amount, say ϵ, i.e.,

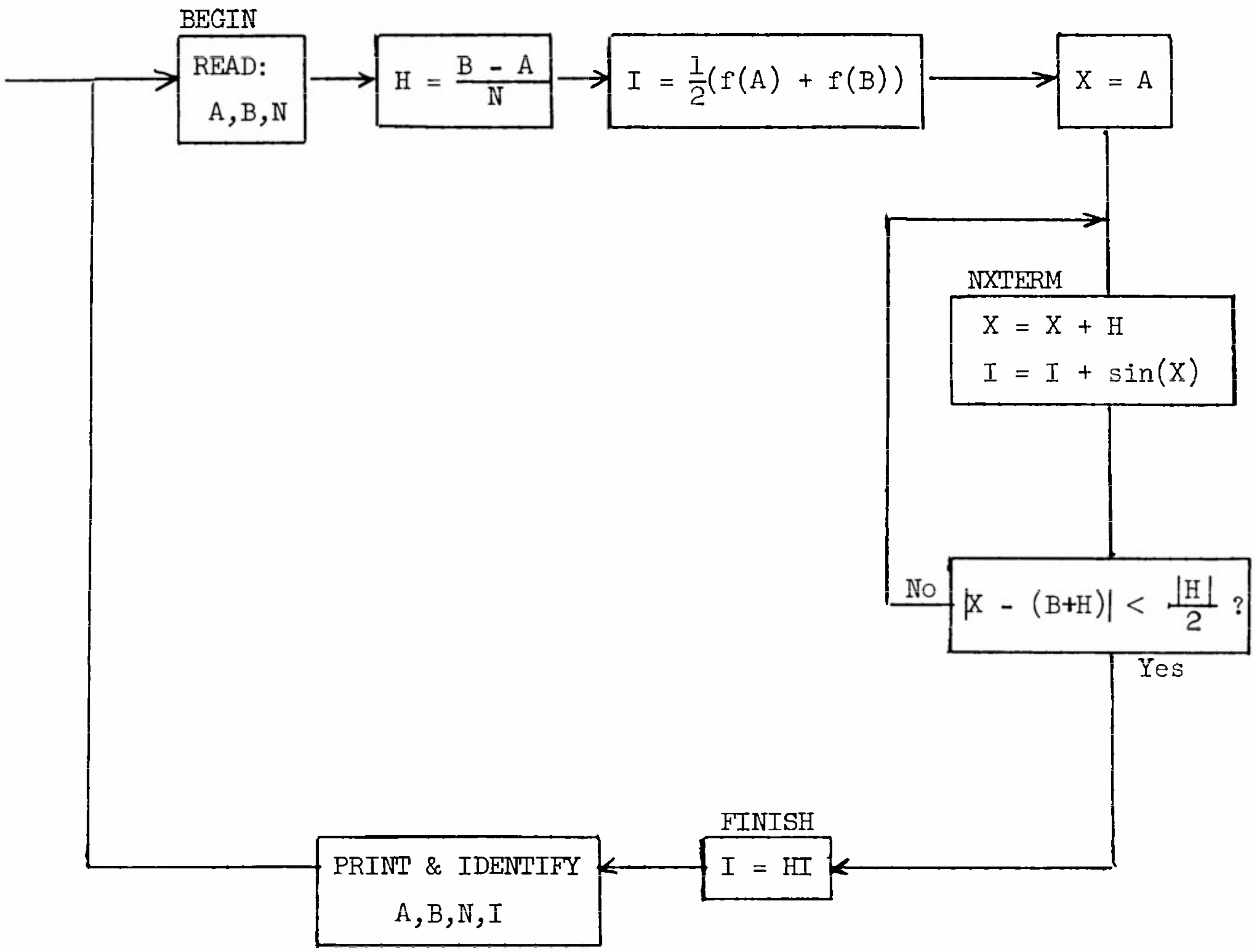
$$|X - (B-H)| < \epsilon$$

If we choose $\epsilon = \frac{|H|}{2}$, we will provide for a maximum tolerable error. We express this conditional statement in the following form:

```
IF-ABS(X-(B-H) < H/2)GOTO(FINISH)%   GOTO(NXTERM)%
```

When ABS precedes a conditional expression, the computers first evaluate both sides of the conditional expression, then take the absolute values of both sides before checking to see if the relation is satisfied. In essence, the above statement corresponds mathematically to

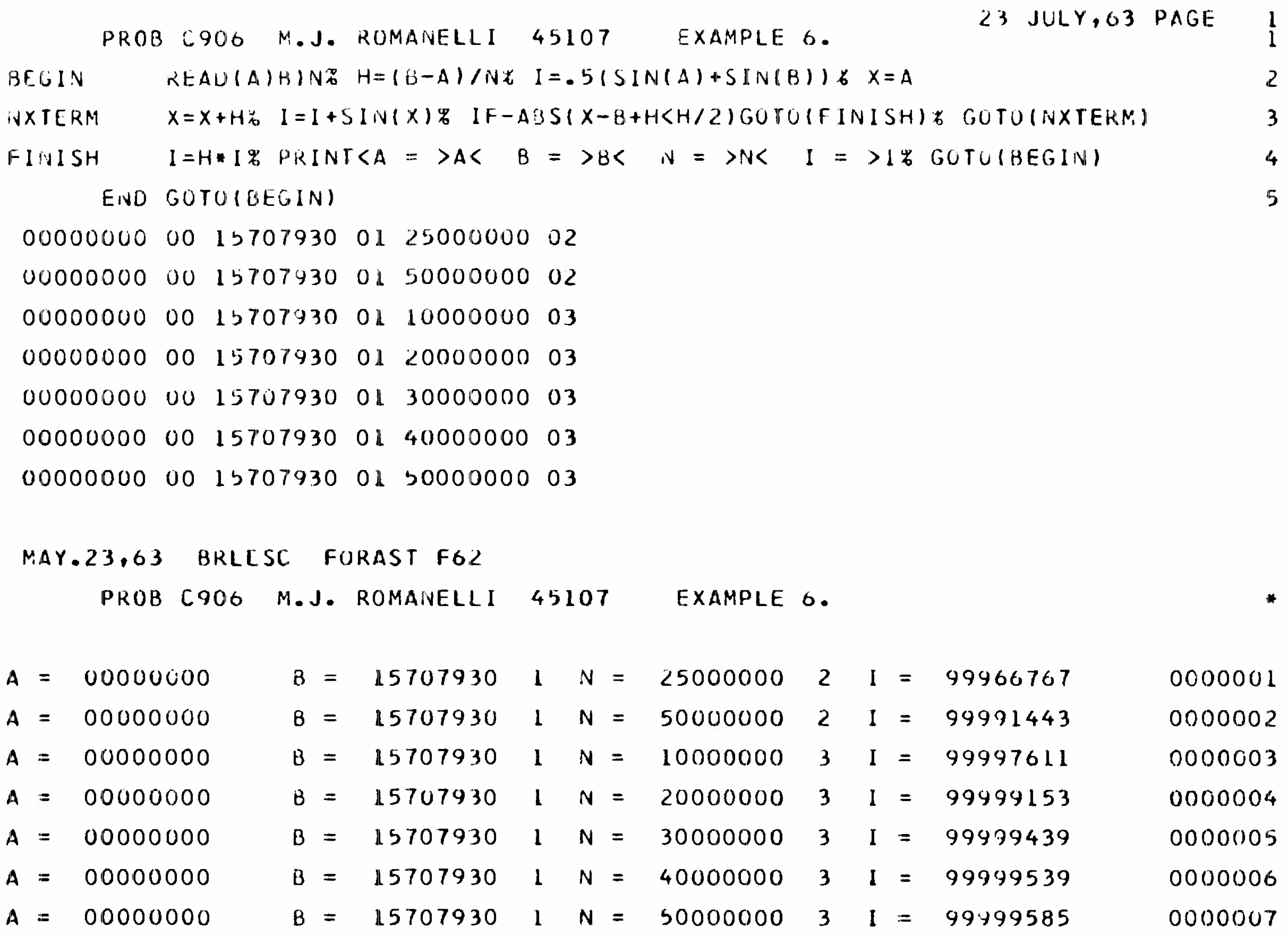
if $|X-B+H| < \frac{|H|}{2}$ go to finish, otherwise go to the computation of the next term. At the location called FINISH, we want the computer to multiply the existing value of I by H to complete the definition of the trapezoidal rule. The complete flow-chart is shown in FIGURE 16.



FLOW CHART FOR EXAMPLE 6, (TRAPEZOIDAL RULE).

FIGURE 16.

The program, input and output for EXAMPLE 6 are listed in FIGURE 17.



```
                                                                   23 JULY,63 PAGE    1
      PROB C906  M.J. ROMANELLI  45107    EXAMPLE 6.                                  1
BEGIN      READ(A)B)N%  H=(B-A)/N%  I=.5(SIN(A)+SIN(B))%  X=A                         2
NXTERM     X=X+H%  I=I+SIN(X)%  IF-ABS(X-B+H<H/2)GOTO(FINISH)%  GOTO(NXTERM)          3
FINISH     I=H*I%  PRINT<A = >A<  B = >B<  N = >N<  I = >I%  GOTO(BEGIN)              4
      END  GOTO(BEGIN)                                                                5
 00000000 00 15707930 01 25000000 02
 00000000 00 15707930 01 50000000 02
 00000000 00 15707930 01 10000000 03
 00000000 00 15707930 01 20000000 03
 00000000 00 15707930 01 30000000 03
 00000000 00 15707930 01 40000000 03
 00000000 00 15707930 01 50000000 03


 MAY.23,63  BRLESC  FORAST F62
      PROB C906  M.J. ROMANELLI  45107    EXAMPLE 6.                                  *


A =  00000000     B =  15707930  1  N =  25000000  2  I =  99966767      0000001
A =  00000000     B =  15707930  1  N =  50000000  2  I =  99991443      0000002
A =  00000000     B =  15707930  1  N =  10000000  3  I =  99997611      0000003
A =  00000000     B =  15707930  1  N =  20000000  3  I =  99999153      0000004
A =  00000000     B =  15707930  1  N =  30000000  3  I =  99999439      0000005
A =  00000000     B =  15707930  1  N =  40000000  3  I =  99999539      0000006
A =  00000000     B =  15707930  1  N =  50000000  3  I =  99999585      0000007
```

FIGURE 17.

To identify the output quantities, we have used a variation of the general PRINT statement. The statement

```
PRINT  < Ab=b> A < bbBb=b> B < bbNb=b> N < bbIb=b> I %
```

instructs the computer to PRINT(PUNCH) the name of the quantity, an equality symbol, followed by the corresponding numerical value of the quantity. (The lower case b's merely serve to indicate blanks and are used to obtain convenient spacing between symbols, numerical values, etc.). Since all characters specified between < and > in PRINT statements are printed, the above statement provides for output in the form

```
A = ----  B = ----  N = ----  I = ----  .
```

Observe in FIGURE 17 that the numerical values are represented in standard floating-point form since we did not specify any format to depart from the standard representation.

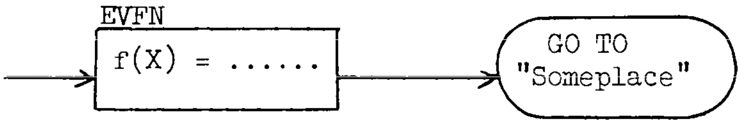
EXAMPLE 7.

To illustrate a concept that is applicable in many problems we will write another program to obtain a solution to the problem of EXAMPLE 6. The concept to be illustrated is one in which we will "vary" the "exit" of a general function evaluation so that we will direct the computer to various destinations in order to weight (or in general utilize) the evaluated function in accordance with given definitions. Recall that the definition of the trapezoidal rule

$$I = H \left[\frac{f(A)}{2} + f(A+H) + f(A+2H) + \cdots + \frac{f(B)}{2} \right]$$

requires the end terms be weighted by 1/2, and the interior terms require weights of unity. In EXAMPLE 6, we disposed of the end terms by treating them separately; i.e., we explicitly defined them and constructed general definitions for the arguments and functional values corresponding to the interior terms. In this example we will include all arguments and terms in general definitions. Although the generalizations are relatively simple for this example, the concept is most advantageous when general definitions are complicated and mere repetitions are laborious.

Consider the following generalization:

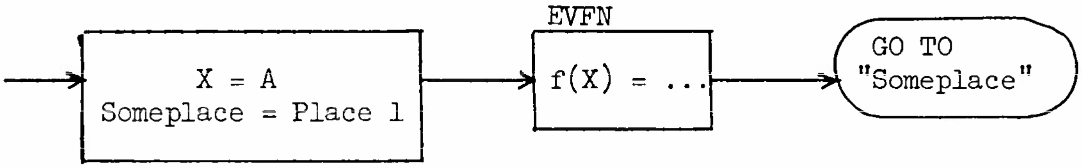


It is assumed that prior to entering EVFN, (evaluate function), X has taken on a prescribed value. The function is then evaluated and recorded as per definition and the computer then goes to "someplace" for subsequent instructions. In addition to specifying the X for which the functional value is required, we can specify the "someplace" for subsequent instructions! That is, we can let "someplace" take on a specific name prior to entering EVFN.

For example, suppose:

First: Let X = A
Let "someplace" = Place 1
Go to evaluate function

This is appended to the above generalization and illustrated schematically below



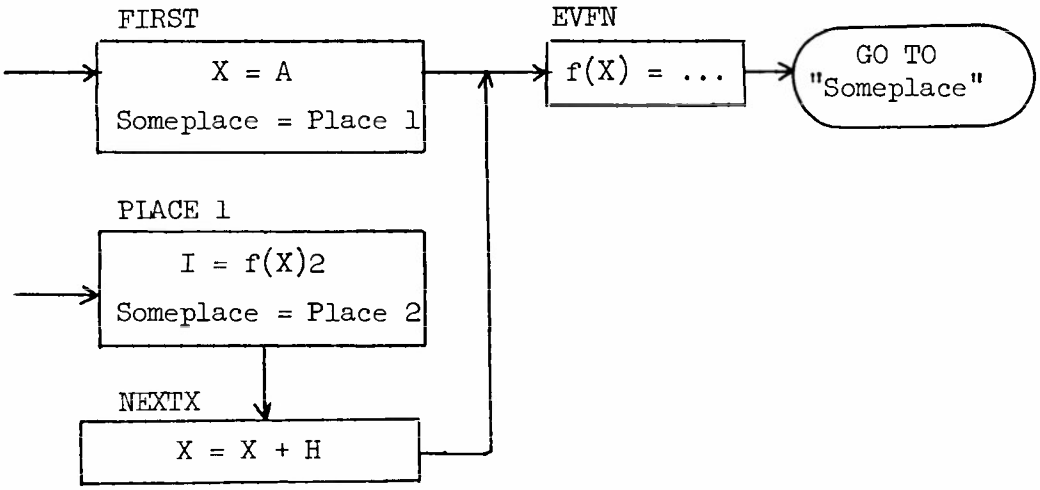
When the computer "arrives" at Place 1, (since someplace has existing value Place 1), f(X) has been evaluated for X = A and hence is available for whatever treatment is desired. Suppose then at

PLACE 1 : Form I = f(X)/2
Let "Someplace" = Place 2
Go to generate next X

where at

NEXTX : Form X = X + H
Go to evaluate function

The flow-chart is augmented to reflect the above as follows:



Prior to evaluating the function for $X = X + H$, "someplace" has existing value Place 2, hence at Place 2, $f(X)$ corresponding to $f(X+H)$ is available for subsequent treatment. Now, at Place 2, "subsequent" treatment depends on whether $f(X)$ has just been evaluated for $X = B$, (for if it has we want to apply weight 1/2, otherwise weight unity!). Hence at

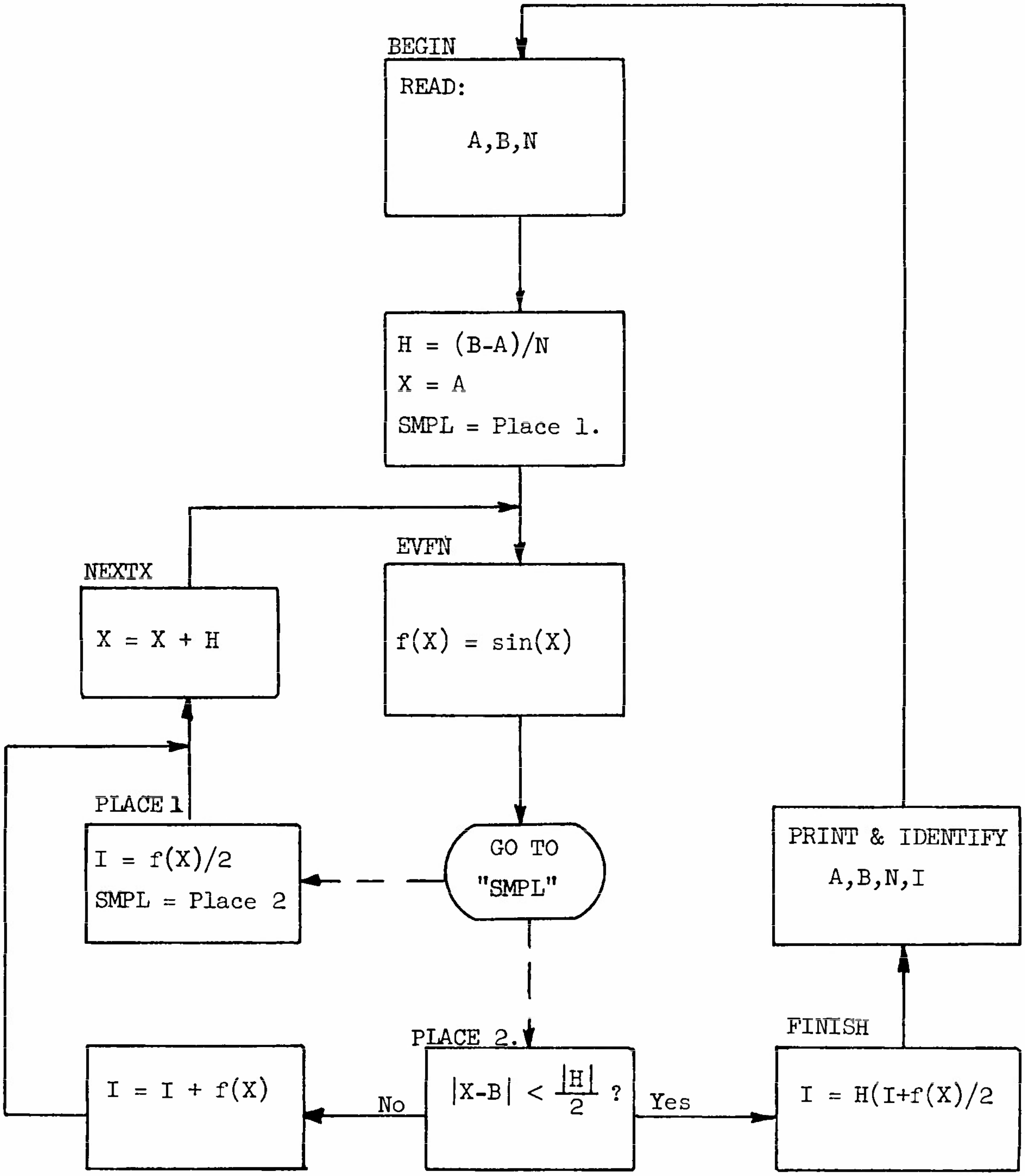
```
PLACE 2.    If $|X - B| < \frac{|H|}{2}$    go to finish ;
            otherwise, form    $I = I + f(X)$
                   Go to next X .
```

(Note that "someplace" retains the undisturbed existing value, (Place 2).

When the computer "arrives" at finish, the existing value of I represents the sum of the first N terms. Now we need only add 1/2 of the existing $f(X) = f(B)$ to I and multiply by H to obtain the desired solution. Hence at

```
FINISH:    $I = H\ (I + f(X)/2)$
           Print ---------------- etc.
```

The complete flow-chart which includes the above is shown in FIGURE 18, the corresponding program and results are shown in FIGURE 19.



FLOW CHART FOR EXAMPLE 7.

FIGURE 18.

```
                                                            23 JULY,63 PAGE    1
      PROB C906 M.J.ROMANELLI   45107 EXAMPLE 7
BEGIN      READ(A)B)N% H=(B-A)/N% X=A% SET(SMPL=PLACE1)
EVEN       FX=SIN(X)% GOTO(,SMPL)
PLACE1     I=  FX/2% SET(SMPL=PLACE2)
NEXTX      X=X+H% GOTO(EVEN)
PLACE2     IF-ABS(X-B<H/2)GOTO(FINISH)% I=I+FX% GOTO(NEXTX)
FINISH     I=H(I+FX/2)%
           PRINT<A = >A<  B = >B<  N = >N<  I = >I% GOTO(BEGIN)
      END GOTO(BEGIN)
 00000000 00 15707930 01 25000000 02
 00000000 00 15707930 01 50000000 02
 00000000 00 15707930 01 10000000 03
 00000000 00 15707930 01 20000000 03
 00000000 00 15707930 01 30000000 03
 00000000 00 15707930 01 40000000 03
 00000000 00 15707930 01 50000000 03

 MAY.23,63  BRLESC  FORAST F62
      PROB C906  M.J. ROMANELLI  45107   EXAMPLE 7                            *

A = 00000000    B = 15707930  1  N = 25000000  2  I = 99966767     0000001
A = 00000000    B = 15707930  1  N = 50000000  2  I = 99991443     0000002
A = 00000000    B = 15707930  1  N = 10000000  3  I = 99997611     0000003
A = 00000000    B = 15707930  1  N = 20000000  3  I = 99999153     0000004
A = 00000000    B = 15707930  1  N = 30000000  3  I = 99999439     0000005
A = 00000000    B = 15707930  1  N = 40000000  3  I = 99999539     0000006
A = 00000000    B = 15707930  1  N = 50000000  3  I = 99999585     0000007
```

FIGURE 19.

We have in our previous examples made direct references by identifying quantities by names, i.e., we constructed symbolic names of variables, functions, statements, etc. It is important to distinguish between "names" of quantities and the quantities themselves. If for example,

$$X = 1.3$$
$$Y = 72$$

and we write

$$Z = X + Y$$

Z, X and Y are names, whereas 73.3, 1.3 and 72 are the quantities or one may say that 73.3, 1.3, and 72 are the numerical values of Z, X, and Y.

In arithmetic expressions and formulas such as the above it is understood that the operations are to be performed on the numerical values and not their names. As we shall illustrate shortly, in some statements it is the name and not the numerical value that is inferred. A trivial example is the statement, GOTO(BEGIN). In general, one need not associate a numerical value with the name BEGIN, (we shall leave such interpretations to the professional programmer!).

In this example we introduce the SET statement which illustrates a case where the name and not the value is inferred. SET statements are generally of the form

SET(A = B)

where:

A is generally the symbolic name of an index;
B is either a symbolic name or an explicit integer.

If B is a symbolic name, then A takes on a value that is the name B, (not the value of B).

If B is an explicit integer, then A takes on the value that is the explicit integer.

Hence, the "value" that A takes on depends on whether B is a symbolic name or whether B is an explicit integer.

EXAMPLES:

SET(A = SAM) , then the existing value of A is SAM.
SET(A = 4) , then the existing value of A is 4.

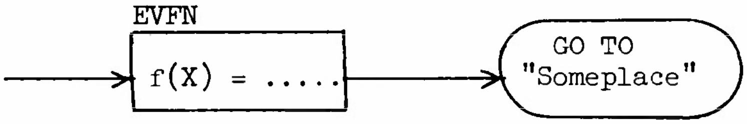
Recall that in the previous discussion of general names, a name that contains the special character comma is an "indexed" name and indexed names depend on the existing "value" of the index. For example, X1,I is an indexed name.
We call X1 the base name and I an index name.
If I has existing value 2, then X1,I represents the name X3.
Similarly, the name ,J is an indexed name.
If J has existing value PETE, then ,J represents the name PETE or we may say that the "value" of ,J is PETE.

Hence, indices may take on two types of values:

a.) the value of an index may be a name * ;
b.) the value of an index may be an integer.

(In EXAMPLE 8 we will illustrate an example where the value of an index is an integer).

In the generalization



to let "someplace" = Place 1 prior to entering the evaluation of the function

* In general, an index can only take on integer values since all names are automatically transformed to unique integers during the transformation from the FORAST language to the primitive machine language.

with X = A, we write

SET(SMPL = PLACE 1).

Then following the definition of f(X) we write

GOTO(,SMPL)

Hence, after evaluation of f(A), the existing value of

SMPL is PLACE 1

consequently the computer goes to PLACE 1 for subsequent instructions. Similarly at PLACE 1, we defined

I = f(X)/2 and followed this definition with
SET(SMPL = PLACE 2)

thereby establishing the existing value of SMPL as PLACE 2. Throughout the remainder of the computation, for the existing A, B and N, SMPL retains its value of PLACE 2 so that all exits from the evaluation of the function are directed to PLACE 2.

EXAMPLE 8.

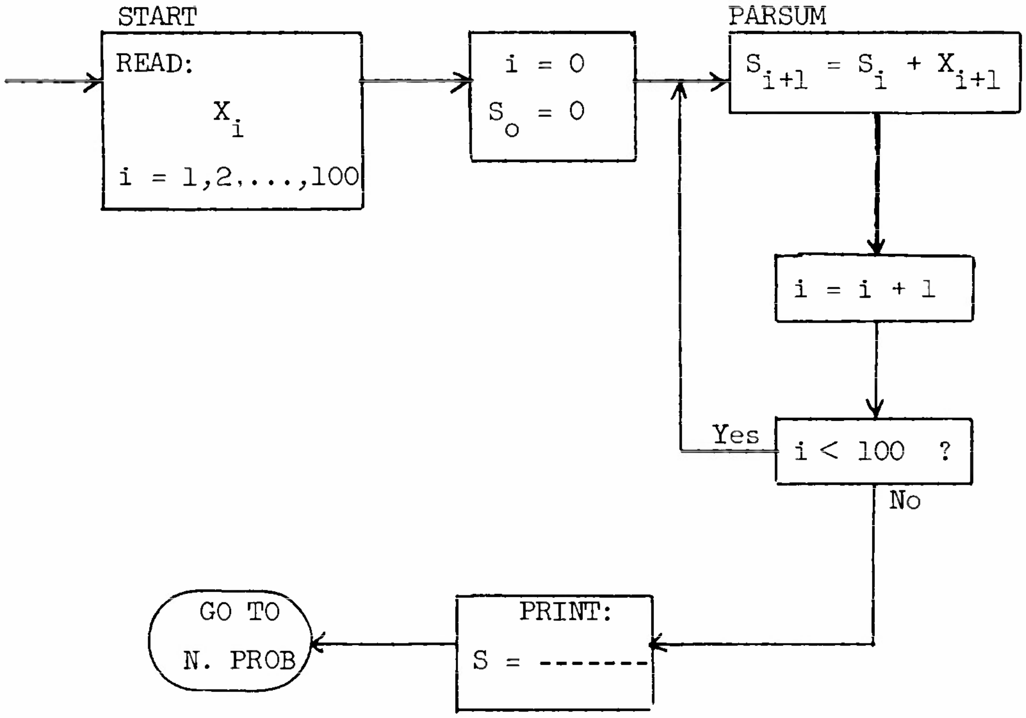
This example illustrates a convenient use of an index which takes on prescribed integer values. We will also introduce a COUNT statement, a BLOC statement, another form of the general READ statement and a GOTO(next problem) statement.

GIVEN:

X_i recorded on cards in standard form, six per card, $i = 1,2,\ldots,100$

REQUIRED:

$$S = \sum_{i=1}^{100} X_i \text{ ; Print: } S = \text{--------------- in standard form.}$$



FLOW CHART FOR EXAMPLE 8

FIGURE 20.

On the flow chart of FIGURE 20, we have indicated that we want the computer to START by reading and recording the 100 numerical values corresponding to the given X_i. We assume that they are recorded on cards in standard form, 6 numerical values per card. To instruct the computer to do this we write

```
READ(100)NOS.AT(X1)%
```

The integer enclosed in parentheses immediately after READ specifies the number of values that are to be read and recorded. (Actually, one may write the explicit integer as above, or the name of an existing integer). The name enclosed in parentheses immediately after AT specifies where they are to be recorded internally.

To insure that there will be no conflict in identifying the 100 X's, we reserve internal space by writing a BLOC statement

```
BLOC(X1 - X100)%
```

The BLOC statement not only reserves 100 consecutive spaces for the X's, it also establishes X2, X3, ..., X99 as valid names associated with the X's. (This may seem trivial, however, it is necessary since names such as XA, X', X., etc. are also permissible names!). Since BLOC is an order type (O.T.) word it may only appear in columns 7-10, the bloc definitions themselves are recorded commencing in column 11. Also, since BLOC is an order type (O.T.) word, it is not transformed into machine instructions, hence it is generally placed after the PROB card and before the START card.

To obtain the sum of the X's we could write a statement

$$S = X1 + X2 + X3 + \text{etc.},$$

however; i may be large (several thousand) and consequently we generalize so that we do not have to write and identify each and every term. We will generalize by computing "partial sums", generally denoted by S_i. We first SET(I=0) and define the corresponding partial sum, $S_o = 0$. The general partial sum is defined in terms of the previous partial sum; i.e.,

$$S_{i+1} = S_i + X_{i+1}$$

When our ultimate goal is to obtain the final sum, we need not even distinguish between the (i+1)st and the ith partial sum, hence we may write

$$S = S + X_{i+1}$$

because the existing value of S is used on the right of the equality before the resulting value is assigned to the name on the left of the equality. We have arbitrarily labelled the general partial sum definition, PARSUM. The program, corresponding to the portion of the flow chart which has been discussed thus far, is as follows:

```
                PROB      C906    M.J. ROMANELLI    45107    EXAMPLE 8 .
                BLOC      (X1 - X100)
START                     READ(100)NOS.AT(X1)% SET(I=0)%  S = 0
PARSUM                    S = S + X1,I
```

When the computer encounters the PARSUM statement for the first time, the existing value of S is zero and I = 0, hence after executing the PARSUM statement for the first time, the computer obtains

$$S = 0 + X1 = X1 .$$

Since we have generalized, we could obtain our desired solution if we could instruct the computer to execute the PARSUM statement exactly 100 times, corresponding to I = 0,1,2,...,99. The general COUNT statement was designed specifically for such situations! The COUNT statement is generally written in the form

COUNT (A) IN (B) GOTO (C) %

where:

A is the name of an explicit integer or name of an integer;

B is the name of an integer;

C is the name of a place.

First, the index, whose name appears in the parenthesis after IN, 'is automatically increased by unity.

Next, the existing value of the B index is compared with the integer indicated in parenthesis after COUNT, (or with the existing value of the index whose name appears in parenthesis after COUNT), i.e. the existing value of B is compared with the existing value of A.

Finally, if the existing value of B is less than the existing value of A, then the computer goes to C for subsequent instructions; otherwise, i.e., if the existing value of B is > or = the existing value of A, the computer goes to the next statement, i.e., the statement immediately following this COUNT statement.

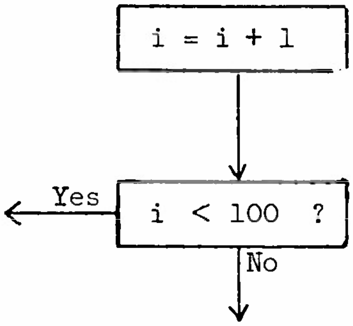
Again, to generalize, if the initial existing value of B is the integer b and the existing value of A is the integer a, then the instructions at C will be executed exactly

(a - b) times .

For our example, we write

COUNT(100)IN(I)GOTO(PARSUM) %

Since, we initially set I = 0 so that X1,I initially represents the first X, we have a = 100 and b = 0, hence the PARSUM statement will be executed (100-0) times. Of equal importance is the fact that the index i will automatically take on the desired values 1,2,3,...,100. We have emphasized the fact on the flow chart by writing $i = i + 1$ in the box just prior to the determination as to whether $i < 100$? Note also that even though i will take on the value 100, the PARSUM statement will not be executed corresponding to $i = 100$, since as indicated, i is advanced by unity immediately after the execution of the PARSUM statement and just prior to the determination of $i < 100$? Hence, the single COUNT statement corresponds to the two boxes of the flow chart indicated below:

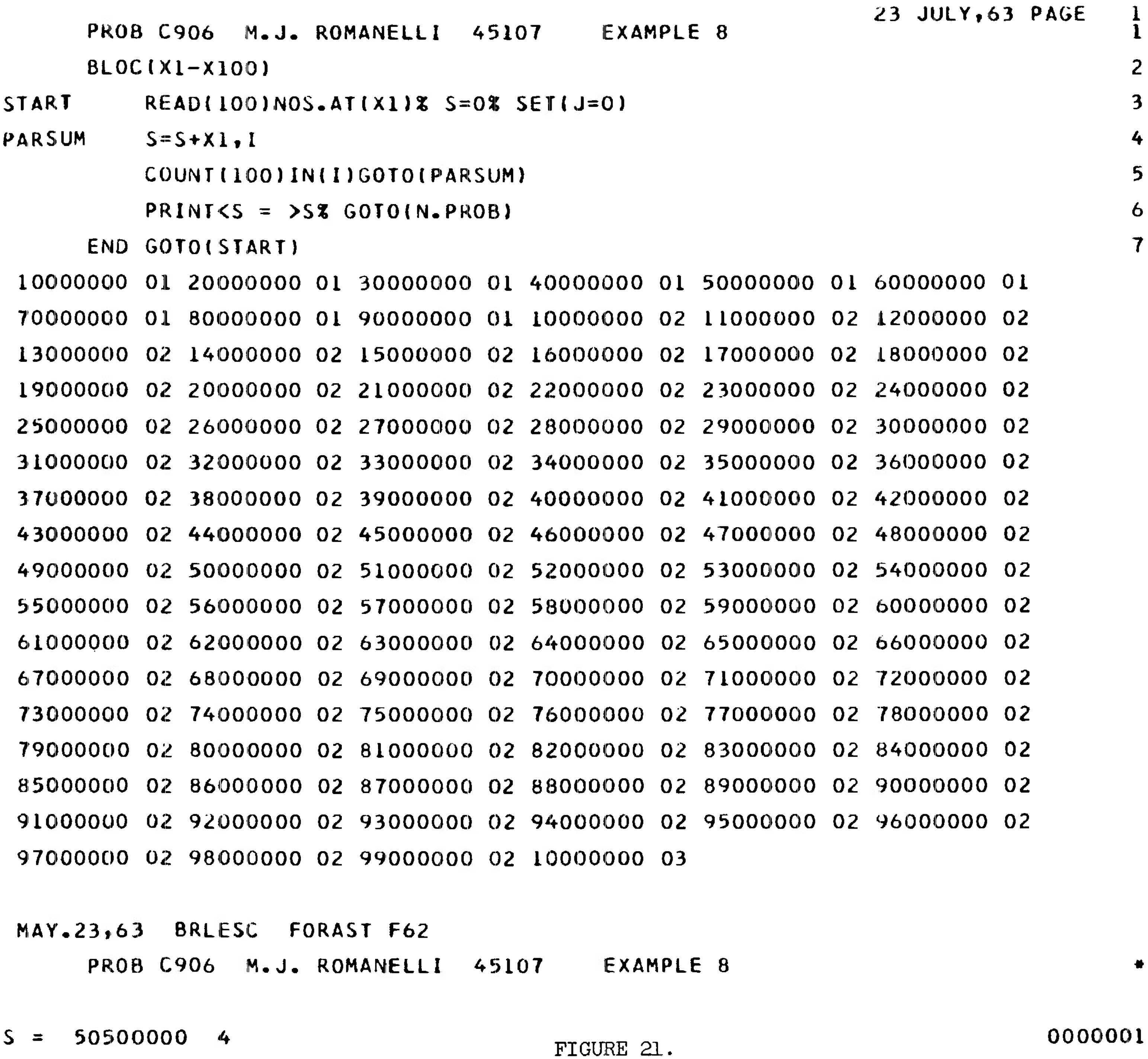


Hence, immediately after the PARSUM statement, we write the COUNT statement. The COUNT statement is followed by the PRINT statement and then the "new" statement

GOTO(N.PROB)

is written. We use this latter statement to signal to the computer the end of the execution of our problem, and it directs the computer to the reading of the next problem residing in the input device.

The program, input and output corresponding to EXAMPLE 8 is listed in FIGURE 21.



```
                                                                 23 JULY,63 PAGE    1
      PROB C906  M.J. ROMANELLI  45107    EXAMPLE 8                                 1
      BLOC(X1-X100)                                                                 2
START       READ(100)NOS.AT(X1)% S=0% SET(J=0)                                      3
PARSUM      S=S+X1,I                                                                4
            COUNT(100)IN(I)GOTO(PARSUM)                                             5
            PRINT<S = >S% GOTO(N.PROB)                                              6
      END GOTO(START)                                                               7
 10000000 01 20000000 01 30000000 01 40000000 01 50000000 01 60000000 01
 70000000 01 80000000 01 90000000 01 10000000 02 11000000 02 12000000 02
 13000000 02 14000000 02 15000000 02 16000000 02 17000000 02 18000000 02
 19000000 02 20000000 02 21000000 02 22000000 02 23000000 02 24000000 02
 25000000 02 26000000 02 27000000 02 28000000 02 29000000 02 30000000 02
 31000000 02 32000000 02 33000000 02 34000000 02 35000000 02 36000000 02
 37000000 02 38000000 02 39000000 02 40000000 02 41000000 02 42000000 02
 43000000 02 44000000 02 45000000 02 46000000 02 47000000 02 48000000 02
 49000000 02 50000000 02 51000000 02 52000000 02 53000000 02 54000000 02
 55000000 02 56000000 02 57000000 02 58000000 02 59000000 02 60000000 02
 61000000 02 62000000 02 63000000 02 64000000 02 65000000 02 66000000 02
 67000000 02 68000000 02 69000000 02 70000000 02 71000000 02 72000000 02
 73000000 02 74000000 02 75000000 02 76000000 02 77000000 02 78000000 02
 79000000 02 80000000 02 81000000 02 82000000 02 83000000 02 84000000 02
 85000000 02 86000000 02 87000000 02 88000000 02 89000000 02 90000000 02
 91000000 02 92000000 02 93000000 02 94000000 02 95000000 02 96000000 02
 97000000 02 98000000 02 99000000 02 10000000 03

 MAY.23,63  BRLESC  FORAST F62
      PROB C906  M.J. ROMANELLI  45107    EXAMPLE 8                                 *

S =  50500000  4                                                              0000001
```

FIGURE 21.

EXAMPLE 9.

This example illustrates the MOVE and INC (increment) statements, and the SYN (synonym) order type.

GIVEN: X_i and $f(X_i)$ recorded on cards in standard form, i = 1,2,3,...,20. Assume the X_i on four consecutive cards, 6 values on each of the first three cards, and two values on the fourth card. Assume the corresponding $f(X_i)$ on the next four cards, again 6 values on each of the first three cards and two values on the last card. Assume that the X_i are not in monotone sequence.

REQUIRED: Obtain a "re-arranged" table of the X_i and corresponding $f(X_i)$ such that

$$X_i \leqq X_{i+1} \quad \text{for all i.}$$

Print both tables; i.e., the original and re-arranged tables with the numerical entries in standard form. In particular, print a table in the following form:

ORIGINAL		REARRANGED	
X1	FI	XI	FI
~	~	~	~
~	~	~	~
~	~	~	~

Since we will have to instruct the computer to determine if each $X_i \leqq X_{i+1}$, we will first provide for getting the given table of numerical values recorded internally. This can be accomplished with the following READ statements:

```
READ(20)NOS.AT(X1)%

READ(20)NOS.AT(F1)%
```

Alternatively, we can provide for the internal recording of both the X_i and corresponding $f(X_i)$ with the following READ statement:

READ(44)NOS.AT(X1) %

With this single READ statement, we are instructing the computers to include in the 44 numerical values, the four "extraneous" X_i, (i = 21,22,23,24), of the 4th card. (Recall that only two X_i, (i = 19,20) are recorded on the 4th card). In the absence of a format, the computers assume 6 standard values per card until the READ (or PRINT) statement is fulfilled.

To reserve space in internal storage for the X_i and $f(X_i)$ we write the BLOC statement

BLOC(X1 - X44)

For future references to the $f(X_i)$ we write a SYN statement

SYN(F1 = X25)

In SYN statements, the equated names are assigned identical internal names (identical integers) and hence references to any of the equated names refer to the same quantity. Again, recall that the BLOC statement not only reserves space in internal storage, it also establishes X1,X2,X3,...,X44 as valid names. The above SYN statement establishes F1 and X25 as synonomous names. Without the SYN statement we could refer to the $f(X_i)$ in many ways, for example,

X24,I where I = 1,2,...,20
or X1,J where J = 24,25,...,43.

However, with the SYN statement we can refer to the $f(X_i)$ as

F1,I

thereby retaining the symbolic identification of the given data. Further, if we refer to the X_i as X1,I and if we refer to the $f(X_i)$ as F1,I we need only "manipulate" the numerical value for I to make references to X_i and the corresponding $f(X_i)$.

Our problem requires that both the original and re-arranged forms of the table are to be printed. To have both forms available in internal storage for printing, we can have the given table recorded in two areas. In one area we can retain the original form, in another area we can interchange any values required to obtain the desired re-arranged form. To reserve space for a second area we can augment the BLOC statement; i.e., we simply add another parenthesized expression which defines the second area. Let us arbitrarily denote the second area as (X'1 - X'44). The BLOC statement which reserves space for both areas is then of the form

```
BLOC(X1 - X44)(X'1 - X'44)%
```

(Alternatively, one could define a BLOC (X1 - X88) and an accompanying SYN (X'1 = X45) to reserve space for both tables and identification of the second area beginning at X45).

Now, the single READ statement will provide for the recording of the given table in the X1 - X44 area. To obtain a copy of the given table in the X'1 - X'44 area, we use the convenient MOVE statement. The MOVE statement is generally of the form

```
MOVE(A)NOS.FROM(B)TO(C)%
```

where:

A is either the name of an integer or an explicit integer which defines the number of quantities that are to be moved;

B is the name of the first quantity in a source area;

C is the name of the first quantity in a destined area.

The quantities are retained in the source area and hence after the MOVE statement has been executed the quantities exist in both areas. In the above MOVE statement it is assumed that the quantities in both areas will be uniformly spaced,one unit apart. Uniform spacings other than one unit may be specified for the source and(or) destined area by following the symbolic name with a / and the integer defining the spacing. For example,

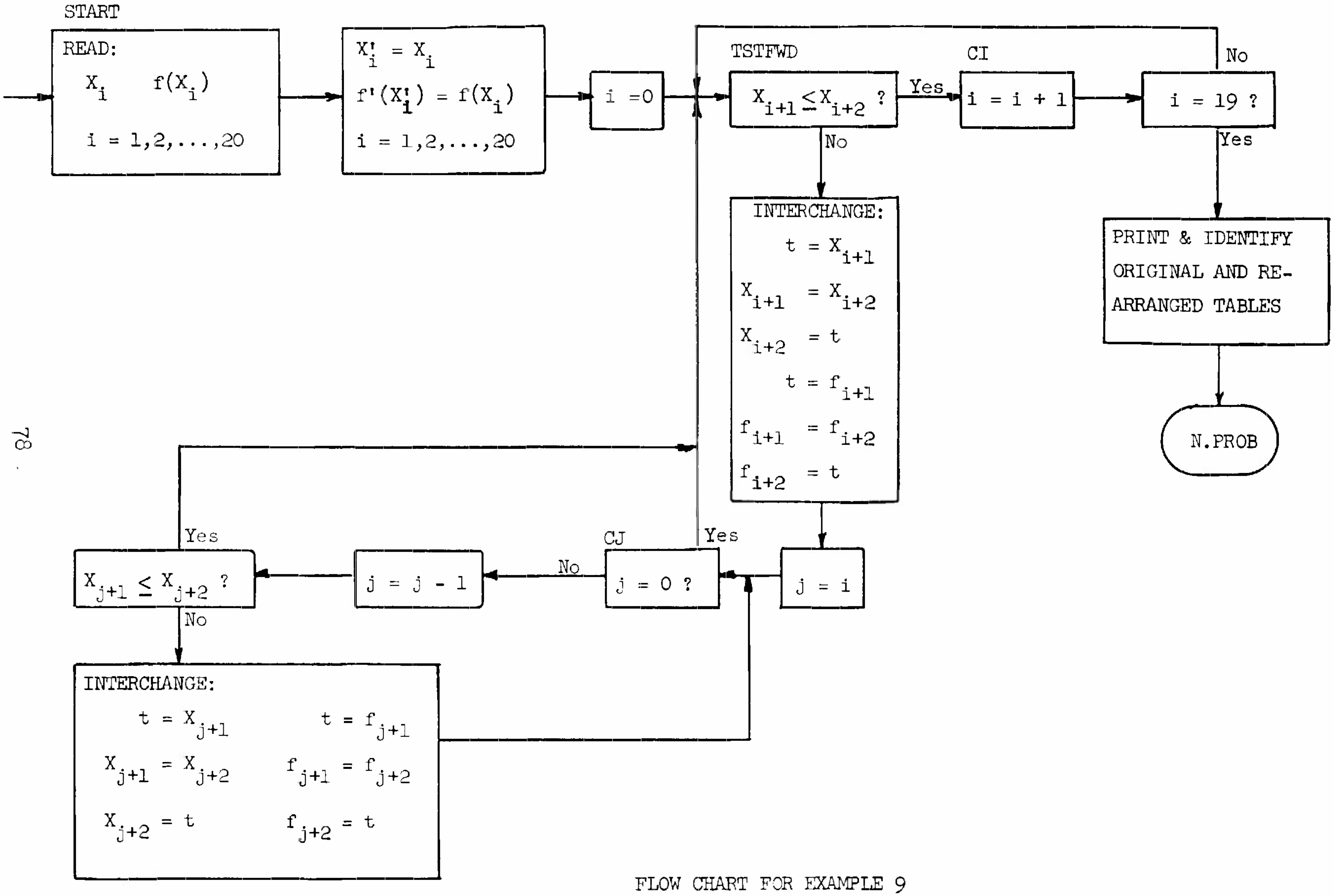
```
MOVE(N)NOS.FROM(A/3)TO(B/5)%
```

indicates that the N quantities in the source area are three units apart, the N quantities in the destined area are to be spaced five units apart.

For our example we write

```
MOVE(44)NOS.FROM(X1)TO(X'1)%
```

Ordinarily we plan a solution by drawing a flow chart. In this example, we delayed the drawing of the chart until the capabilities of the new statements were discussed so that the novice obtains some insight as to "how" the flow chart solution can easily be expressed in the FORAST language. The complete flow chart for a solution is given in FIGURE 22.



FLOW CHART FOR EXAMPLE 9

FIGURE 22.

Note that the first two boxes of the flow chart of FIGURE 22 indicate the recording of the given data in two areas of internal storage, the primed and unprimed areas. The third box indicates that the index i is to take on value zero. The next box labelled TSTFWD (test forward) indicates the monotone test. If the monotone condition is satisfied, the index i is advanced by one and thence a determination is made as to whether all of the X's in the table have been tested. If all have been tested, the existing value of i is 19 and hence we indicate the printing of the tables. If all X's have not been tested, ($i < 19$), hence we want the computer to test the next pair.

If the monotone condition is not satisfied, we want to interchange X_{i+1} and X_{i+2}, and the corresponding functional values. This interchange will guarantee that this pair now satisfies the monotone condition, but once an interchange is made, we have no guarantee that the monotone condition prevails for all X's that have previously been tested. Hence, we must test backwards until one pair of X's does satisfy the monotone condition. To perform this backward test, note that we define another index j which initially takes on the existing value of i. Next, if $j = 0$ no further backward testing is necessary, hence we indicate that forward testing is to be resumed. If j is not zero, the existing value of j is diminished by one and the backward test continued. Note that an interchange is indicated any time the monotone condition is not satisfied. As soon as the condition is satisfied in the backward testing, the forward testing is resumed.

The program, input and output for this example are listed in FIGURE 23.

```
                                                            23 JULY,63 PAGE    1
     PROB C906  M.J. ROMANELLI  45107    EXAMPLE 9
     BLOC(X1-X44)X'1-X'44)
     SYN (F1=X25)
START   READ(44)NOS.AT(X1)
        MOVE(44)NOS.FROM(X1)TO(X'1)
        SET(I=0)
TSTFWD  IF(X1,I<=X2,I)GOTO(CI)
        T=X1,I% X1,I=X2,I% X2,I=T
        T=F1,I% F1,I=F1,(I+1)% F1,(I+1)=T% J=I
CJ      IF-INT(J=0)GOTO(TSTFWD)% INC(J=J-1)
        IF(X1,J<=X2,J)GOTO(TSTFWD)%
        T=X1,J% X1,J=X2,J% X2,J=T
        T=F1,J% F1,J=F1,(J+1)% F1,(J+1)=T% GOTO(CJ)
CI      COUNT(19)IN(I)GOTO(TSTFWD)%
        PRINT<          ORIGINAL                 REARRANGED>
        PRINT<      XI          FI          XI          FI>% SET(I=0)
WORK    PRINT(X'1,I)X'25,I)X1,I)F1,I)% COUNT(20)IN(I)GOTO(WORK)
        GOTO(N.PROB)
     END GOTO(START)
 93000000 00 95000000 00 98000000 00 10000000 01 12000000 01 12500000 01
 17000000 01 17500000 01 18000000 01 18500000 01 19000000 01 19300000 01
 13000000 01 13500000 01 14000000 01 15000000 01 16000000 01 16500000 01
 19800000 01 20000000 01
 59783000 00 58168000 00 55702000 00 54030000 00 36236000 00 31532000 00
-12884000 00-17825000 00-22720000 00-27559000 00-32329000 00-35153000 00
 26750000 00 21901000 00 16997000 00 70740000-01-29200000-01-79120000-01
-39788000 00-41615000 00
```

```
MAY.23,63  BRLESC  FORAST F62
     PROB C906  M.J. ROMANELLI  45107    EXAMPLE 9                          (

       ORIGINAL                 REARRANGED                                0000001
    XI          FI          XI          FI                                0000002
93000000    59783000    93000000    59783000                              0000003
95000000    58168000    95000000    58168000                              0000004
98000000    55702000    98000000    55702000                              0000005
10000000  1 54030000    10000000  1 54030000                              0000006
12000000  1 36236000    12000000  1 36236000                              0000007
12500000  1 31532000    12500000  1 31532000                              0000008
17000000  1-12884000    13000000  1 26750000                              0000009
17500000  1-17825000    13500000  1 21901000                              0000010
18000000  1-22720000    14000000  1 16997000                              0000011
18500000  1-27559000    15000000  1 70740000-01                           0000012
19000000  1-32329000    16000000  1-29200000-01                           0000013
19300000  1-35153000    16500000  1-79120000-01                           0000014
13000000  1 26750000    17000000  1-12884000                              0000015
13500000  1 21901000    17500000  1-17825000                              0000016
14000000  1 16997000    18000000  1-22720000                              0000017
15000000  1 70740000-01 18500000  1-27559000                              0000018
16000000  1-29200000-01 19000000  1-32329000                              0000019
16500000  1-79120000-01 19300000  1-35153000                              0000020
19800000  1-39788000    19800000  1-39788000                              0000021
20000000  1-41615000    20000000  1-41615000                              0000022
```

FIGURE 23.

The program contains two new concepts which have not been discussed. Note first the name

F1,(I + 1)

on line 9. This is used as a reference for $f(X_{i+2})$. The above is an illustration of an indexed name of the form

A,(B ± n)

where A is a defined base name, B is the name of an index and n is an integer. The resulting name is obtained by increasing (or decreasing) the existing value of B by n and adding this result to the base name A. That is, the parenthesized expression is evaluated and the resulting integer is added to A to obtain the resulting name. If for example, the existing value of I is 7, then

F1,(I + 1) represents the name F9
F1,(I - 3) represents the name F5.

Next, on line 10 is an illustration of the INC statement of the form

INC(A = A ± n)

where A is the name of an index and n is an explicit integer. This statement is generally used to increase (or decrease) index values by constant amounts. (More general means for operating on integer values will be illustrated in later examples) For our example we wrote

INC(J = J - 1)

which told the computer to dimish the value of J by one.

EXAMPLE 10.

This example illustrates the LIST order-type word which is used for two purposes:

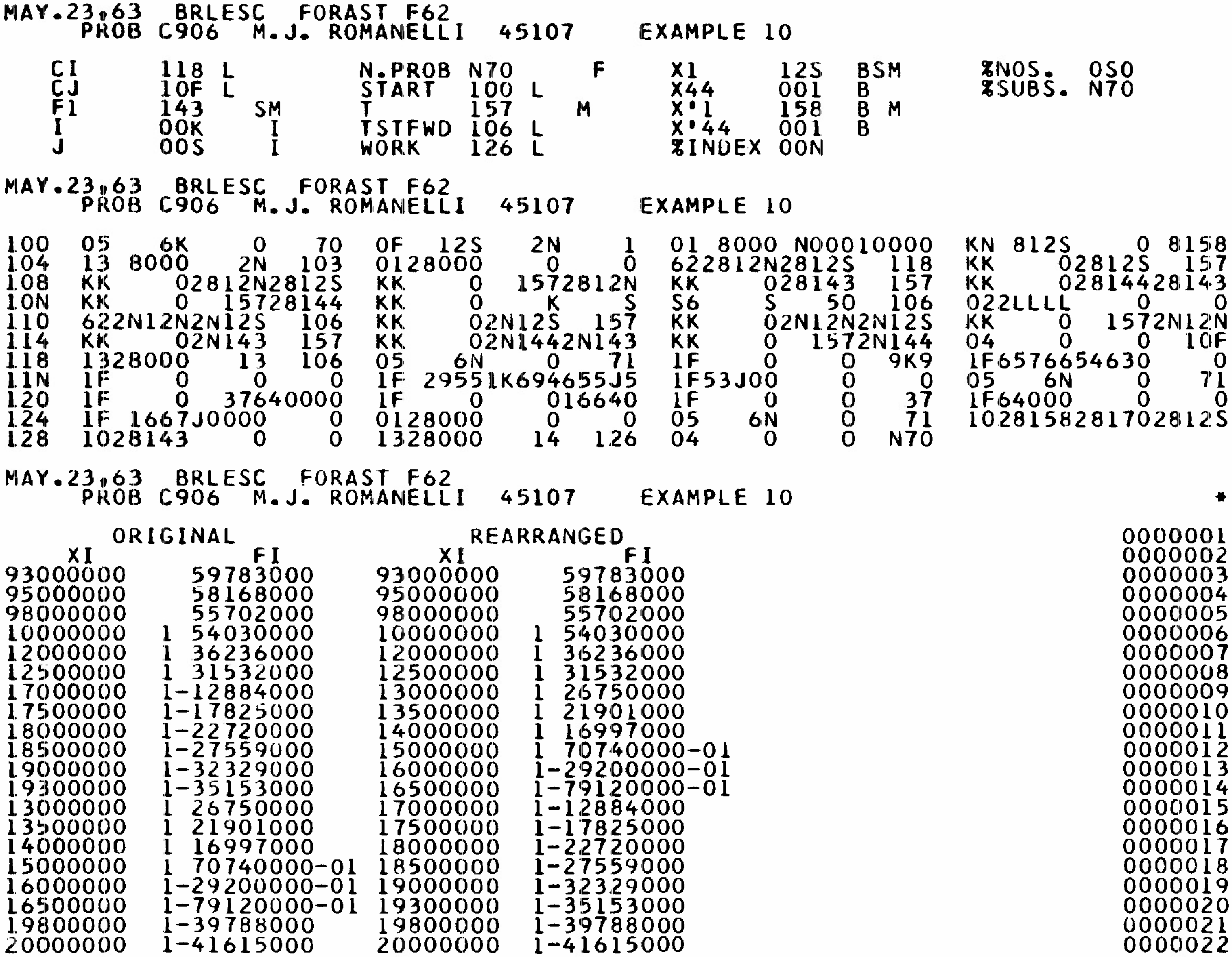
a.) It is used by the programmer to obtain a listing which shows the one to one correspondence between the symbolic names in his program and unique integers. The unique integers represent the absolute machine names of the internal storage units. Hence this listing shows the programmer the explicit storage assignment by the FORAST compiler. Since the compiler does not check for all possible violations and conflicts, a study of the listing can reveal conflicts of storage assignment or other programming errors.

b.) It is used by the professional programmer to obtain the absolute machine language corresponding to his symbolic FORAST language. We will indicate how this is obtained and list the absolute machine language of EXAMPLE 10; however, we leave the interpretation to the professional programmer.

During the transformation from the symbolic FORAST language to the absolute machine language, indications of errors will be printed to inform the programmer of the errors. Further, the listing (dictionary) referred to under a.) above will be produced and no attempt will be made to execute the program. The novice will find on many occassions that even though no errors were detected during compilation, his program does not run correctly. In such a case, no dictionary is obtained and to begin to determine a source of error (or errors) it is advisable to obtain the dictionary. To obtain the dictionary, we simply record LIST in columns 7 through 10 of a card and place this card in front of the END card. This indicates that the dictionary is desired and will be produced whether or not any errors are detected during compilation. (When the absolute machine language is desired in addition to the dictionary, S.CODE is recorded beginning in column 11 of the LIST card referred to above).

We have inserted a LIST card in the program of EXAMPLE 10. The results are listed in FIGURE 24.

```
                                                                       23 JULY,63 PAGE    1
         PROB C906   M.J. ROMANELLI   45107     EXAMPLE 10
         BLOC(X1-X44)X'1-X'44)
         SYN  (F1=X25)
START         READ(44)NOS.AT(X1)
              MOVE(44)NOS.FROM(X1)TO(X'1)
              SET(I=0)
TSTFWD        IF(X1,I<=X2,I)GOTO(CI)
              T=X1,I% X1,I=X2,I% X2,I=T
              T=F1,I% F1,I=F1,(I+1)% F1,(I+1)=T% J=I
CJ            IF-INT(J=0)GOTO(TSTFWD)% INC(J=J-1)
              IF(X1,J<=X2,J)GOTO(TSTFWD)%
              T=X1,J% X1,J=X2,J% X2,J=T
              T=F1,J% F1,J=F1,(J+1)% F1,(J+1)=T% GOTO(CJ)
CI            COUNT(19)IN(I)GOTO(TSTFWD)%
              PRINT<          ORIGINAL                    REARRANGED>
              PRINT<      XI             FI             XI             FI>% SET(I=0)
WORK          PRINT(X'1,I)X'25,I)X1,I)F1,I)% COUNT(20)IN(1)GOTO(WORK)
              GOTO(N.PROB)
         LIST S.CODE
         END GOTO(START)
 93000000 00 95000000 00 98000000 00 10000000 01 12000000 01 12500000 01
 17000000 01 17500000 01 18000000 01 18500000 01 19000000 01 19300000 01
 13000000 01 13500000 01 14000000 01 15000000 01 16000000 01 16500000 01
 19800000 01 20000000 01
 59783000 00 58168000 00 55702000 00 54030000 00 36236000 00 31532000 00
-12884000 00-17825000 00-22720000 00-27559000 00-32329000 00-35153000 00
 26750000 00 21901000 00 16997000 00 70740000-01-29200000-01-79120000-01
-39788000 00-41615000 00
```



```
MAY.23,63  BRLESC  FORAST F62
     PROB C906  M.J. ROMANELLI  45107   EXAMPLE 10

   CI     118 L       N.PROB N70     F   X1     12S  BSM     %NOS.  0S0
   CJ     10F L       START  100 L       X44    001  B       %SUBS. N70
   FI     143   SM    T      157    M    X'1    158  B M
   I      00K    I    TSTFWD 106 L       X'44   001  B
   J      00S    I    WORK   126 L       %INDEX 00N

MAY.23,63  BRLESC  FORAST F62
     PROB C906  M.J. ROMANELLI  45107   EXAMPLE 10

100  05   6K    0   70 0F  12S   2N    1 01 8000 N00010000 KN 812S    0 8158
104  13 8000   2N  103 0128000    0    0 622812N2812S  118 KK    02812S  157
108  KK    02812N2812S KK    0  1572812N KK    028143  157 KK    02814428143
10N  KK    0  15728144 KK    0    K    S S6    S   50  106 022LLLL    0    0
110  622N12N2N12S  106 KK    02N12S  157 KK    02N12N2N12S KK    0  1572N12N
114  KK    02N143  157 KK    02N1442N143 KK    0  1572N144 04    0    0  10F
118  1328000   13  106 05   6N    0   71 1F    0    0  9K9 1F6576654630    0
11N  1F    0    0    0 1F 29551K694655J5 1F53J00    0    0 05   6N    0   71
120  1F    0  37640000 1F    0    016640 1F    0    0   37 1F64000    0    0
124  1F 1667J0000    0 0128000    0    0 05   6N    0   71 1028158281702812S
128  1028143    0    0 1328000   14  126 04    0    0  N70

MAY.23,63  BRLESC  FORAST F62
     PROB C906  M.J. ROMANELLI  45107   EXAMPLE 10                          *

       ORIGINAL              REARRANGED                              0000001
    XI          FI         XI          FI                            0000002
93000000    59783000   93000000    59783000                          0000003
95000000    58168000   95000000    58168000                          0000004
98000000    55702000   98000000    55702000                          0000005
10000000  1 54030000   10000000  1 54030000                          0000006
12000000  1 36236000   12000000  1 36236000                          0000007
12500000  1 31532000   12500000  1 31532000                          0000008
17000000  1-12884000   13000000  1 26750000                          0000009
17500000  1-17825000   13500000  1 21901000                          0000010
18000000  1-22720000   14000000  1 16997000                          0000011
18500000  1-27559000   15000000  1 70740000-01                       0000012
19000000  1-32329000   16000000  1-29200000-01                       0000013
19300000  1-35153000   16500000  1-79120000-01                       0000014
13000000  1 26750000   17000000  1-12884000                          0000015
13500000  1 21901000   17500000  1-17825000                          0000016
14000000  1 16997000   18000000  1-22720000                          0000017
15000000  1 70740000-01 18500000  1-27559000                         0000018
16000000  1-29200000-01 19000000  1-32329000                         0000019
16500000  1-79120000-01 19300000  1-35153000                         0000020
19800000  1-39788000   19800000  1-39788000                          0000021
20000000  1-41615000   20000000  1-41615000                          0000022
```

FIGURE 24.

Note first that all of the symbolic names used in the program are listed in "alphabetical" order. (Numerals in general are "less than" any alphabetic character and appear in the "alphabetical sequence" before any alphabetic character). Immediately to the right of a symbolic name is the unique integer assigned by the compiler. This integer is recorded in the sexadecimal number system, (base 16). The characters in this system which correspond to the decimal ten, eleven,...,fifteen are respectively K,S,N,J,F and L. Hence the integer listed corresponding to the symbolic name

X1 is 12S ,

which represents the decimal integer

$$1 \text{ x } (16)^2 + 2\text{x}(16)^1 + 11\text{x}(16)^0 = 256 + 32 + 11 = 299.$$

The integer (absolute machine name) is generally followed by one or more of the alphabetic characters, A,B,F,I,L,M,R,S or U. Only a few of the meanings are listed below, the meanings of each are given in [1.] .

B indicates a BLOC name and opposite the end name of the BLOC is an integer which represents the uniform spacing between the elements of the BLOC;

F indicates the name of a function;

I indicates that the name is an index name;

L indicates a name that was recorded in the LOCATION columns, usually a statement name;

S indicates that the name was defined in a SYN statement.

U indicates that the name appeared only once in the program and hence this single reference may indicate a programming or punching error.

As an illustration of an undetected error, suppose on line 7 of EXAMPLE 10 one wrote GOTO(C1) and on line 14 in the LOCATION column one wrote CI. Note that

$$C1 \neq CI$$

and both C1 and CI are distinct and valid names. Both would appear in the dictionary followed by the letter U to indicate single references. The obvious error is that C1 and CI were not designed as distinct references, and either the 1 should be changed to I or the I to 1. The last three entries in the listing

% INDEX
% NOS.
% SUBS.

correspond to the extent of the storage used for indices, constants and functions. The integer opposite % INDEX corresponds to the integer that would be assigned to the next index encountered. Similarly, the integer opposite % NOS. corresponds to the integer that would be assigned to the next constant encountered. Finally, the functions (subroutines) are allocated space at the end of internal storage (largest integers) and hence the next function encountered would be assigned space just above the integer opposite % SUBS.

EXAMPLE 11.

This example introduces the CLEAR and SETEA statements and illustrates convenient variations of the COUNT, READ and BLOC statements.

GIVEN: X_i and $f(X_i)$ recorded on cards in standard form, $i = 1,2,\ldots,20$

Assume the X_i are in monotone sequence.

REQUIRED: Compute and print a table of forward differences,

$$\Delta_i^j = \Delta_i^{j-1} - \Delta_{i-1}^{j-1}$$

where

$$j = 1,\ 2,\ 3,\ 4$$

$$i = 1,\ 2,\ 3,\ \ldots\ ,\ 20$$

$$\Delta^o = f$$

and all undefined Δ's = 0, i.e.,

$$\Delta_1^j = 0 \quad \text{for} \quad j = 1,\ 2,\ 3,\ 4$$

$$\Delta_2^j = 0 \quad \text{for} \quad j = 2,\ 3,\ 4,\ \text{etc.}$$

Print the table in the following form:

X,I	F,I	1DEL,I	2DEL,I	3DEL,I	4DEL,I
X_1	$f(X_1)$	0	0	0	0
X_2	$f(X_2)$	Δ_2^1	0	0	0
X_3	$f(X_3)$	Δ_3^1	Δ_3^2	0	0
X_4	$f(X_4)$	Δ_4^1	Δ_4^2	Δ_4^3	0
X_5	$f(X_5)$	Δ_5^1	Δ_5^2	Δ_5^3	Δ_5^4
⋮	⋮	⋮	⋮	⋮	⋮

There are many ways that one can obtain the solution for this example. We chose the method indicated in the flow-chart of FIGURE 25 to illustrate a variation of the BLOC statement which permits us to conveniently index and reference elements in a two dimensional array. Illustrated below is a symbolic form of the desired two dimensional table.

E	1	2	3	4	5	6
1	X_1	f_1	0	0	0	0
2	X_2	f_2	Δ_2^1	0	0	0
3	X_3	f_3	Δ_3^1	Δ_3^2	0	0
4	X_4	f_4	Δ_4^1	Δ_4^2	Δ_4^3	0
5	X_5	f_5	Δ_5^1	Δ_5^2	Δ_5^3	Δ_5^4
$\vdots$						
20	X_{20}	f_6	Δ_{20}^1	Δ_{20}^2	Δ_{20}^3	Δ_{20}^4

We have labelled the rows of the table 1 through 20 and the columns 1 through 6. Note that the entries in the first two columns correspond to the given data, the X_i and $f(X_i)$.

To reserve internal storage space for this array we write the BLOC statement

BLOC(E1,1 - E20,6)

The first name in the enclosed parenthesis, E1,1 defines the name of the first element in the array; in particular it defines the name of the element in the first row and first column. The last name in the enclosed parentheses, E20,6 defines the last element in the array; in particular E20,6 defines the element in the 20th row and 6th column. Hence, space is reserved for the 120 elements in the 20 by 6 array. Having defined the array with the BLOC statement, subsequent references to any element in the array can be made by writing E followed by the explicit integers which define the row and column, the row and column integers must be separated by a comma. For example, we can refer to $f(X_5)$ by writing E5,3 and similarly we can refer to X_3 by writing E3,1. Further, any of these references may be used as a base name in a general index name. For example,

E1,1,J

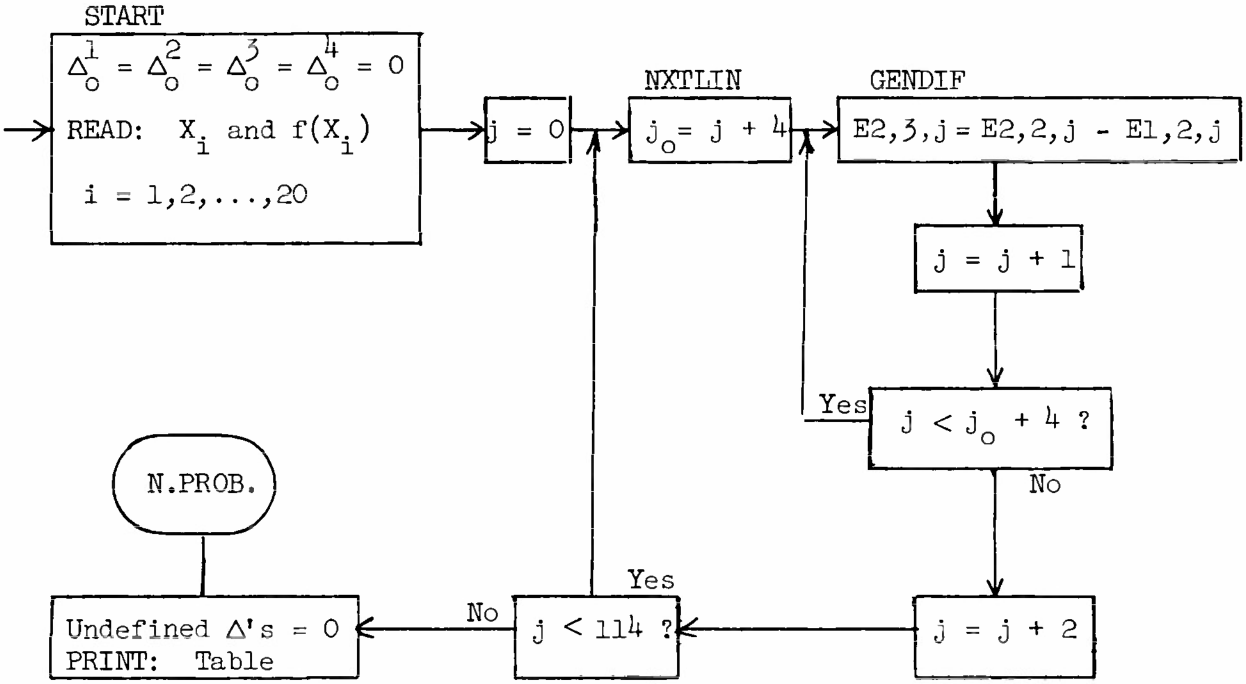
is a valid name and the particular element referenced depends on the existing value of J.

If J = 1, E1,1,J is equivalent to the name E1,2 and refers to element f_1.

If J = 7, E1,1,J is equivalent to the name E2,2 and refers to element f_2.

If J = 4, E2,3,J is equivalent to the name E3,1 and refers to element x_3.

The programmer must bear in mind that successive elements in a row are spaced one unit apart whereas successive elements in the columns are spaced six unit apart. Hence, to advance references from one element to another the index values must be manipulated accordingly. The flow-chart for this example is shown in FIGURE 25.



FLOW CHART FOR EXAMPLE 11.

FIGURE 25.

Note first the indication that the elements Δ_o^1, Δ_o^2, Δ_o^3 and Δ_o^4 are to be defined as zero since they correspond to undefined differences. To instruct the computer to do this we use the convenient CLEAR statement which is of the form

CLEAR(A)NOS.AT(B)%

where A is the name of an index or an explicit integer, and B is the name of the first of the consecutive quantities that are to be cleared, i.e., their numerical values are set to zero. If A is the name of an index, then the current value of that index specifies the number of quantities that are to be cleared. If A is an integer then the integer defines the number of quantities that are to be cleared. It is assumed that the quantities are uniformly spaced in internal storage and in the absence of a specified spacing it is taken to be unity. If a spacing other than unity is desired it is specified by following the name B with a slash, /, followed by the integer which defines the spacing. For example, if we want to clear all of theelements in the 4th column we would write

```
CLEAR(20)NOS.AT(E1,4/6)%
```

For our example we write

```
CLEAR(4)NOS.AT(E1,3)%
```

Next, to read and record the given X_i and $f(X_i)$ in th first two columns of the reserved array, we use a variation of the READ statement where we can specify that the uniform spacing of the quantities to be recorded is not unity. In particular, the uniform spacing is six, hence we write

```
READ(20)NOS.AT(E1,1/6)%
```

which provides for the reading and recording of the twenty X_i's in the reserved area. Similarly, we write

```
READ(20)NOS.AT(E1,2/6)%
```

which provides for the reading and recording of the twenty $f(X_i$'s) in the reserved area.

Now, assuming that the given data have been read and recorded in the reserved area, we plan to generalize the difference definition so that by generating prescribed values of an index we will instruct the computer to compute and record the differences for a given line. Then we will instruct the computer to repeat the process for subsequent lines.

To generalize, note that each difference is defined as the difference of two neighboring quantities in the array, namely the preceding row element minus its preceding column element. In particular,

$$E2,3 = E2,2 - E1,2$$
$$E3,3 = E3,2 - E2,2$$
$$E4,3 = E4,2 - E3,2$$

and in general we can define

$$E2,3,J = E2,2,j - E1,2,J$$

to obtain all of the desired differences provided that J takes on prescribed values.

First, if J took on values 0,1,2, and 3 we would obtain results for the second row; however, only the first difference Δ_2^1, is defined.

Next, if J took on values 6,7,8 and 9 we would obtain results for the third row; however, only the first two differences of.this row are defined.

Similarly, if J took on values 12,13,14 and 15 we would obtain results for the 4th row; here the first three differences are defined.

Finally, if J took on values 18,19,20 and 21 we would obtain the differences for the 5th row. In row 5 and subsequent rows all differences are defined!

For convenience, we will permit the computer to compute and record the incorrect differences in the 2nd, 3rd and 4th rows to maintain generality. Prior to printing the desired table, we will instruct the computer to replace the few "incorrect" differences with their correct zero values.

To achieve the generality, note that the index j is set to zero, and for j = 0,1,2 and 3 the general definition

$$E2,3,J = E2,2,J - E1,2,J$$

will provide the four entries for the second row. Prior to entering the general definition, we set another index, j_o to have value j + 4 so that we can use a COUNT statement to automatically advance j by unity and terminate the computations after the four differences for a given row have been computed and recorded. The COUNT statement will have the form

```
COUNT(JO)IN(J)GOTO(NXTLIN)
```

After the differences for a given row have been computed and recorded, we instruct the computer to determine if all rows have been completed. Again we can use a COUNT statement for this purpose since we know the precise number of rows that we want processed. Further, if all rows have not been processed, the existing value of J should be advanced by 2 to commence the computations for the next row. Again, the COUNT statement is convenient for both purposes; the determination as to whether all rows have been processed; and the automatic increase of the index j by 2. The COUNT statement which will achieve both is written in the form

```
COUNT(114/2)IN(J)GOTO(NXTLIN)%
```

This variation of the COUNT statement specifies that the current value of the index J is to be advanced by 2, then determine if this new value of J < 114 If J <114, go to next line, otherwise continue with the statement immediately following this COUNT statement. Note that the specification of the amount by which the index is to be advanced is specified after the slash following the maximum count, 114. (The student should verify that the maximum count, 114, will indeed provide for the processing of the 20th row and terminate the computation of the general differences. He should also verify that maximum counts of 109, 110, 111, 112 or 113 would achieve the same result).

Note that after all rows have been processed we instruct the computer to replace the few incorrect undefined differences with their correct zero values and then print the desired table.

We stated previously that at NXTLIN we would instruct the computer to let another index, j_o,take on a value equal to $j + 4$. Here we use the convenient SETEA statement which has the general form

$$\text{SETEA}(A = B, C \pm n)$$

where:

A and C are index names;

B is any name;

n is an explicit integer.

The result of the general SETEA statement is that

Index A takes on a value equal to the "name" B + the existing "value" of C ± n.

(Note, not the "value" of B and not the "name" C).

For our example we wrote

SETEA(JO = , J + 4)

A more general means for operating on integers is provided by prefixing formulas with the name INT. Prefixing formulas with INT indicates a departure from standard floating-point operations and in particular indicates that integer arithmetic is to be used to obtain the result. For example,

INT(A = B + 3 - C * D/E)

is valid provided that B,C,D and E are names of existing integers and intermediate operations do not yield results which exceed the restricted range of permissible integer representation. (For compatibility on both ORDVAC and BRLESC, index integer values should not exceed 2^{19} in magnitude, non-index integers are permitted and for compatibility on both ORDVAC and BRLESC should not exceed 2^{39} in magnitude). A further note of caution to the novice, internal representation of integers necessitates special rules for integer operations which do not yield exact integer results. If, for example, in the above INT statement, existing values of D and E are 1 and 3 respectively, then D/E = 1/3 and the computers will yield 1 as the result for D/E. In general, if the result of integer division is a positive mixed number, (i.e., an integer part and a fractional part), the integer result assigned is one greater than the integer part of the mixed number. If the result of integer division is a negative mixed number, the result assigned is the signed integer part of the mixed number.

The program, input and output for EXAMPLE 11 are listed in FIGURE 26. Note on line 8 of the program the statement

```
E2,4 = E2,5 = E2,6 = E3,5 = E3,6 = E4,6 = 0 %
```

This is an example of assigning a numerical result to many distinct quantities. A result may be assigned to as many as 15 distinct quantities where the name of each is separated by the equality symbol. We could not conveniently use the CLEAR statement in this case since all of the elements were not uniformly spaced in internal storage.

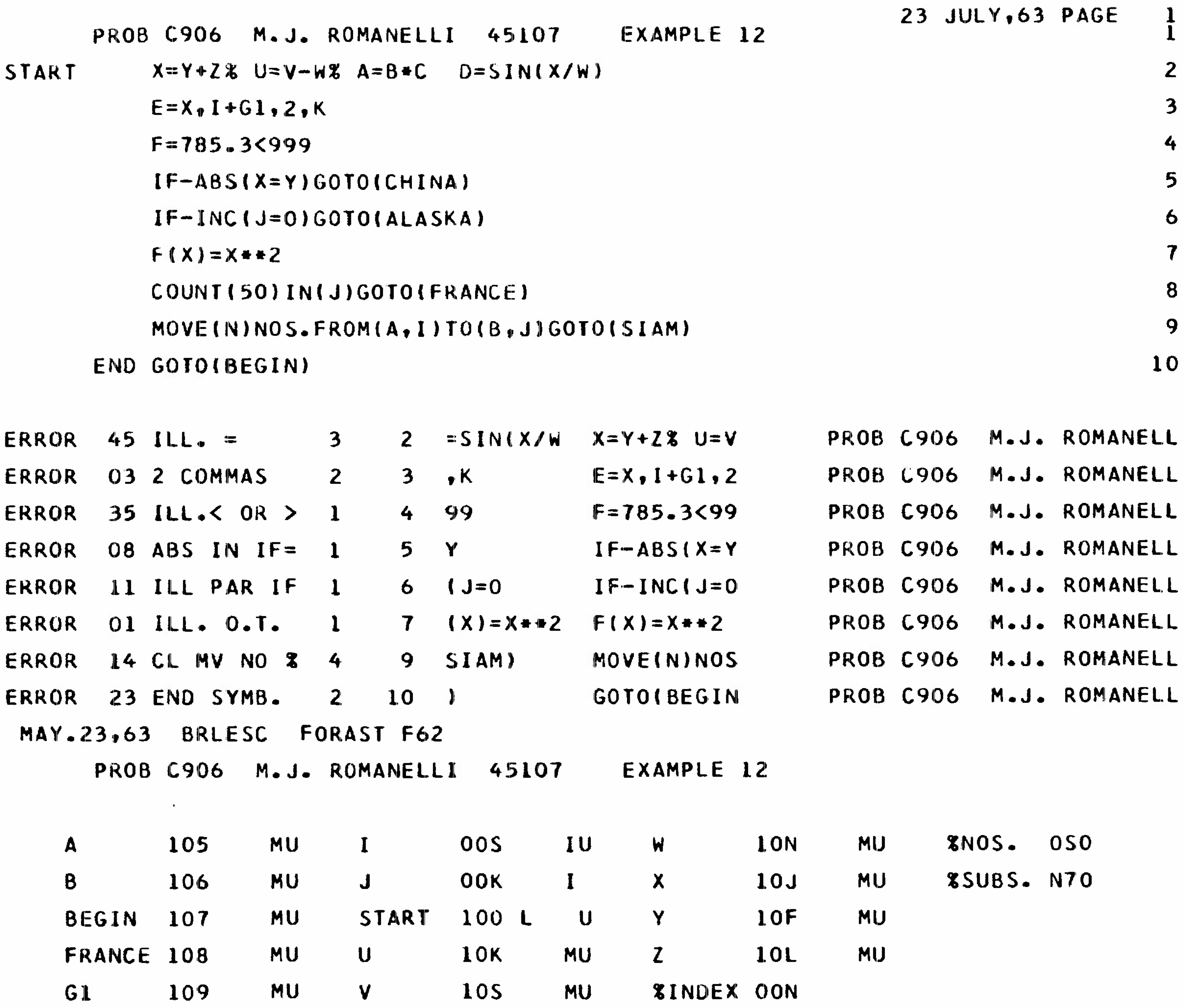
```
MAY.23,63   BRLESC   FORAST F62
     PROB C906   M.J. ROMANELLI   45107     EXAMPLE 11                      *

    X,I               F,I         1DEL,I      2DEL,I      3DEL,I      4DEL,I     0000001
93000000      59783000    00000000    00000000    00000000    00000000    0000002
95000000      58168000   -16150000-01 00000000    00000000    00000000    0000003
98000000      55702000   -24660000-01-85100000-02 00000000    00000000    0000004
10000000    1 54030000   -16720000-01 79400000-02 16450000-01 00000000    0000005
12000000    1 36236000   -17794000   -16122000   -16916000   -18561000    0000006
12500000    1 31532000   -47040000-01 13090000    29212000    46128000    0000007
13000000    1 26750000   -47820000-01-78000000-03-13168000   -42380000    0000008
13500000    1 21901000   -48490000-01-67000000-03 11000000-03 13179000    0000009
14000000    1 16997000   -49040000-01-55000000-03 12000000-03 10000000-04 0000010
15000000    1 70740000-01-99230000-01-50190000-01-49640000-01-49760000-01 0000011
16000000    1-29200000-01-99940000-01-71000000-03 49480000-01 99120000-01 0000012
16500000    1-79120000-01-49920000-01 50020000-01 50730000-01 12500000-02 0000013
17000000    1-12884000   -49720000-01 20000000-03-49820000-01-10055000    0000014
17500000    1-17825000   -49410000-01 31000000-03 11000000-03 49930000-01 0000015
18000000    1-22720000   -48950000-01 46000000-03 15000000-03 40000000-04 0000016
18500000    1-27559000   -48390000-01 56000000-03 10000000-03-50000000-04 0000017
19000000    1-32329000   -47700000-01 69000000-03 13000000-03 30000000-04 0000018
19300000    1-35153000   -28240000-01 19460000-01 18770000-01 18640000-01 0000019
19800000    1-39788000   -46350000-01-18110000-01-37570000-01-56340000-01 0000020
20000000    1-41615000   -18270000-01 28080000-01 46190000-01 83760000-01 0000021
```

FIGURE 26.

EXAMPLE 12.

Specific indications of errors detected during compilation of a program are recorded on an output device to inform the programmer of the particular errors detected. To illustrate the particular information obtained pertaining to detected errors, we wrote a short program which contains several programming errors. We submitted the program to the computer, obtained and listed the results so that one can correlate the information produced with the particular statements of the program submitted.

The program and output obtained are listed in FIGURE 27 below.



```
                                                                 23 JULY,63 PAGE     1
        PROB C906   M.J. ROMANELLI   45107     EXAMPLE 12                            1
START       X=Y+Z% U=V-W% A=B*C  D=SIN(X/W)                                          2
            E=X,I+G1,2,K                                                             3
            F=785.3<999                                                              4
            IF-ABS(X=Y)GOTO(CHINA)                                                   5
            IF-INC(J=0)GOTO(ALASKA)                                                  6
            F(X)=X**2                                                                7
            COUNT(50)IN(J)GOTO(FRANCE)                                               8
            MOVE(N)NOS.FROM(A,I)TO(B,J)GOTO(SIAM)                                    9
        END GOTO(BEGIN)                                                             10

ERROR  45 ILL. =        3    2   =SIN(X/W   X=Y+Z% U=V     PROB C906  M.J. ROMANELL
ERROR  03 2 COMMAS      2    3   ,K         E=X,I+G1,2     PROB C906  M.J. ROMANELL
ERROR  35 ILL.< OR >    1    4   99         F=785.3<99     PROB C906  M.J. ROMANELL
ERROR  08 ABS IN IF=    1    5   Y          IF-ABS(X=Y     PROB C906  M.J. ROMANELL
ERROR  11 ILL PAR IF    1    6   (J=0       IF-INC(J=0     PROB C906  M.J. ROMANELL
ERROR  01 ILL. O.T.     1    7   (X)=X**2   F(X)=X**2      PROB C906  M.J. ROMANELL
ERROR  14 CL MV NO %    4    9   SIAM)      MOVE(N)NOS     PROB C906  M.J. ROMANELL
ERROR  23 END SYMB.     2   10   )          GOTO(BEGIN     PROB C906  M.J. ROMANELL
 MAY.23,63  BRLESC  FORAST F62
        PROB C906   M.J. ROMANELLI   45107     EXAMPLE 12

     A       105    MU    I      00S    IU    W      10N    MU    %NOS.   0S0
     B       106    MU    J      00K    I     X      10J    MU    %SUBS.  N70
     BEGIN   107    MU    START  100 L   U    Y      10F    MU
     FRANCE  108    MU    U      10K    MU    Z      10L    MU
     G1      109    MU    V      10S    MU    %INDEX 00N
```

FIGURE 27.

Programs in general contain many statements and consequently many cards. Since errors are detected during compilation, of immediate concern to the programmer is the recognition of the errors. In particular, which card or cards contain the errors? Identifying the cards which contain the errors presents no problem to the compiler; however, it is the responsibility of the programmer to use the means provided for this purpose. This requires the programmer to uniquely identify each card of his program so that the compiler will provide the unique identification. In particular, the programmer should use the identification columns provided on the standard programming sheets and as suggested previously, simple numerical ordering will suffice to quickly identify the card or cards which contain errors. In general, when the compiler detects an error, it will record a line of information on the output device, (cards or teletype). A portion of the information recorded is the identification of the card that contains the error. Hence if the programmer uses the simple numerical ordering scheme, the compiler will record on the output device the identifying number (or that number + 1) that appeared on the card containing the error. Other relevant information is recorded pertaining to the error; however, identification of "the" card is most important. The results shown in FIGURE 27 correspond to the compilation errors detected on BRLESC . Similar but less detailed results are obtained from ORDVAC. Tables which classify the types of errors are given on pages 78 through 82 of [1] . The essential difference between the ORDVAC and BRLESC results is the amount of detail information recorded pertaining to the error. ORDVAC results indicate the classification type of error, the card and field of the card that contains the error. BRLESC prints the word ERROR, the first and subsequent characters of the field containing the error, the characters recorded in columns 11 through 20 of the card in addition to the classification, card and field identification. BRLESC also prints the first 30 characters that were recorded on the PROB card.

When an error on a card is detected, the remainder of that card is ignored. Further, if the error was detected near the end of a card the next card may also be ignored, hence, all errors may not be detected in a single compilation. The programmer generally uses a conventional process of elimination of errors; i.e., he submits a program, corrects the detected errors, re-submits his program, corrects and re-submits until no further compilation errors are detected. Depending on the nature and complexity of the program, several compilations may be necessary to detect all of the errors.

Note in FIGURE 27, indication of errors on cards 2,3,4,5,6,7,9 and 10. The student should study, identify and correct each of the errors indicated.

EXAMPLE 13.

The fact that the compiler does not detect any errors in a given program does not infer that there are not errors in the program. Another type of error which may be detected can occur during the execution of the compiled program. For example, one may write a valid statement

Z = SQRT(X)

and through an error in input or otherwise the existing value of X may become less than zero. Similarly one may write

Z = ARCSIN(X)

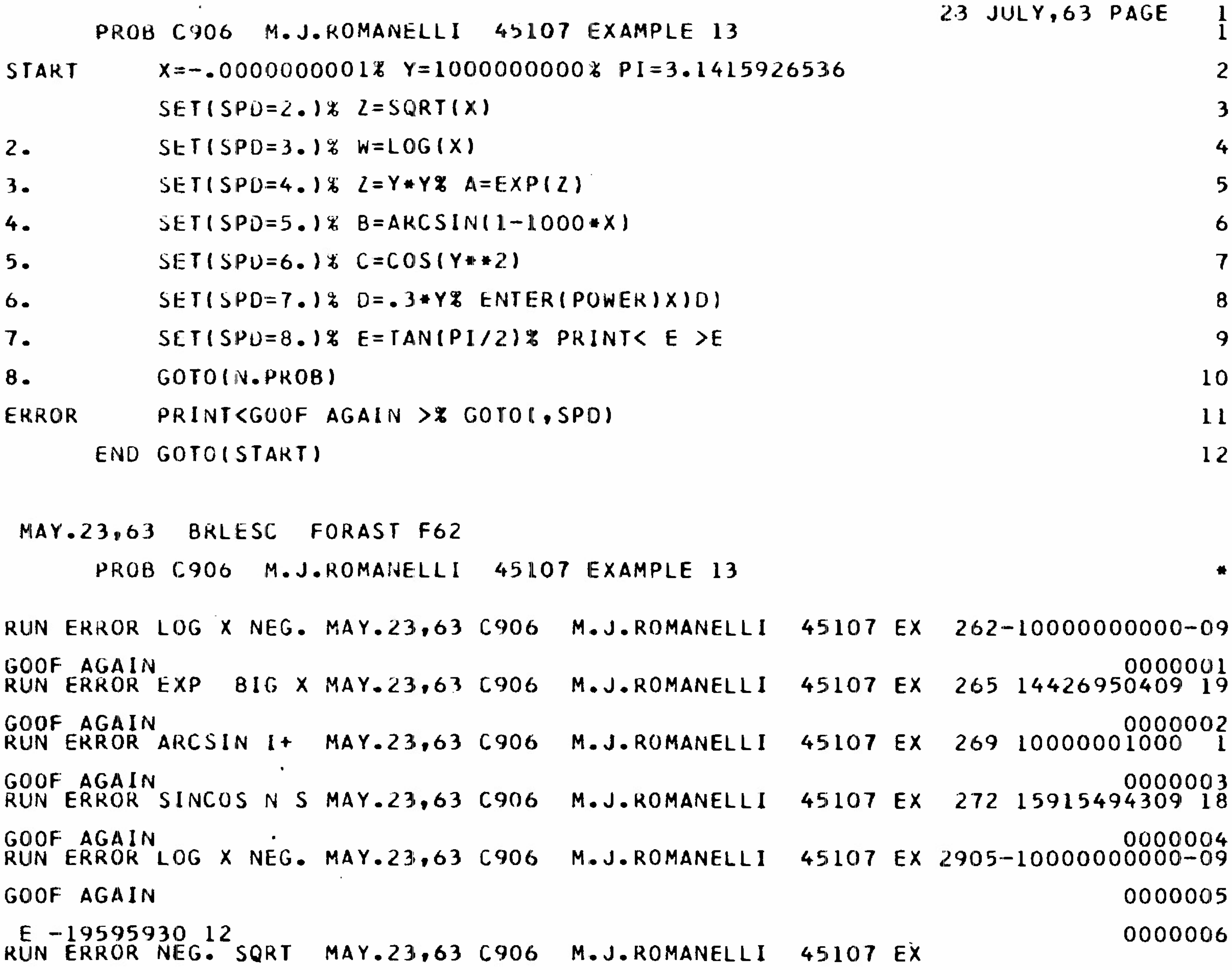
and at some point in the computation X may attain an existing value much greater than one in magnitude due to an error in input or otherwise.

Errors which occur during execution are not as easily identified as those detected during compilation. Here again, the responsibility of correlating the error with the program lies with the programmer. Many of the function subroutines (the functions of single and multiple arguments) include checks for errors. If errors are detected during execution, information is recorded on an output device and the computer will stop unless the programmer provides for continuation. If for example, the programmer wrote twenty distinct statements involving the SQRT function, relevant information is recorded on the output device pertaining to the error in the SQRT argument; however, identification as to which of the twenty caused the error is the responsibility of the programmer.

To provide for continuation after an error is detected and relevant information printed, the programmer must include in his program the location name, ERROR. At the place called ERROR in his program, the programmer may provide for whatever action he desires. He may for example, at ERROR, instruct the computer to print the twenty arguments of the twenty square roots so that he can then ascertain which of the twenty caused the error. At ERROR one could write instructions which test each of the twenty arguments to determine and identify which of the twenty was less than zero and provide for corrective action and continuation without stopping the execution of the program.

In the example which follows we will illustrate the use of ERROR so that the computer will not stop after the first error encountered. In particular, at ERROR, we will instruct the computer to print GOOF AGAIN and continue with the next statement in the program. In this way we can illustrate the pertinent information recorded corresponding to several types of errors.

The program and output are listed in FIGURE 28.



```
                                                                23 JULY,63 PAGE    1
      PROB C906  M.J.ROMANELLI  45107 EXAMPLE 13
START      X=-.0000000001% Y=1000000000% PI=3.1415926536
           SET(SPD=2.)% Z=SQRT(X)
2.         SET(SPD=3.)% W=LOG(X)
3.         SET(SPD=4.)% Z=Y*Y% A=EXP(Z)
4.         SET(SPD=5.)% B=ARCSIN(1-1000*X)
5.         SET(SPD=6.)% C=COS(Y**2)
6.         SET(SPD=7.)% D=.3*Y% ENTER(POWER)X)D)
7.         SET(SPD=8.)% E=TAN(PI/2)% PRINT< E >E
8.         GOTO(N.PROB)
ERROR      PRINT<GOOF AGAIN >% GOTO(,SPD)
      END  GOTO(START)

 MAY.23,63  BRLESC  FORAST F62
      PROB C906  M.J.ROMANELLI  45107 EXAMPLE 13                                       *

RUN ERROR LOG X NEG. MAY.23,63 C906  M.J.ROMANELLI  45107 EX  262-1000000000-09

GOOF AGAIN                                                             0000001
RUN ERROR EXP  BIG X MAY.23,63 C906  M.J.ROMANELLI  45107 EX  265 14426950409 19

GOOF AGAIN                                                             0000002
RUN ERROR ARCSIN I+  MAY.23,63 C906  M.J.ROMANELLI  45107 EX  269 10000001000  1

GOOF AGAIN                                                             0000003
RUN ERROR SINCOS N S MAY.23,63 C906  M.J.ROMANELLI  45107 EX  272 15915494309 18

GOOF AGAIN                                                             0000004
RUN ERROR LOG X NEG. MAY.23,63 C906  M.J.ROMANELLI  45107 EX 2905-1000000000-09

GOOF AGAIN                                                             0000005

 E -19595930 12                                                        0000006
RUN ERROR NEG. SQRT  MAY.23,63 C906  M.J.ROMANELLI  45107 EX
```

FIGURE 28.

EXAMPLE 14.

In EXAMPLE 11, the program was designed to compute and print a table of differences corresponding to the given X_i and $f(X_i)$ where $i = 1,2,\ldots,20$; i.e., it was designed for a table with precisely twenty "rows". Further, internal storage space was reserved for the complete table of differences. In this example, we will design another program for a solution to the problem of EXAMPLE 11, to introduce two new concepts: the program will be designed for a table of "variable" length, $i \leq 1000$; internal storage space will be reserved for the given data and only two rows of differences.

To provide for reading and recording a "variable" amount of data, the READ statement specifies the maximum number of quantities to be read and recorded and the variable amount of data is terminated with a blank card. The computers consider a READ statement fulfilled whenever the specified number of quantities have been read and recorded or whenever a blank card is encountered. (If a blank card is encountered as the first card, it is ignored). Corresponding to any READ statement, the precise number of quantities and the precise number of cards that were read and recorded is made available to the programmer. In particular, after a READ statement has been executed, the number of quantities that were read and recorded is available in integer form in index 09. Similarly, the number of cards that were read and recorded is available in integer form in index 08. Each subsequent READ statement erases the previous integers in 08 and 09 and consequently the existing integers in 08 and 09 at any given time correspond to the latest READ statement that was executed. (Blank cards are not included in the integer counts).

To instruct the computer to read and record the given X_i, $i = 1,2,\ldots,NX$ where $NX \leq 1000$, we write

```
READ(1000)NOS.AT(X1)%  NX = 09
```

The above assumes that the X_i are in standard form, six per card. If $NX < 996$, then the X_i are terminated with a blank card. If for example, there were 997 X's,

X_{997} would be recorded in the first field of the 167th card, and even though the remainder of this card were blank the computer would substitute zeros for the values of X_{998}, X_{999} and X_{1000} ! The integers in 09 and 08 would be respectively 1000 and 167. The statement

```
NX = 09
```

tells the computer to assign a "value" to NX equal to the existing integer in 09. Since "09" is an absolute machine name, the integer 9 is not the value assigned to NX, rather the "value" in index 09 is assigned as the existing value of NX. We wrote the NX = 09 statement immediately following the READ statement so that we will have in NX the number of X's read. After the $f(X_i)$ are read and recorded, we can instruct the computer to check if the number of X's and f's read are identical. To read and record the $f(X_i)$ we write

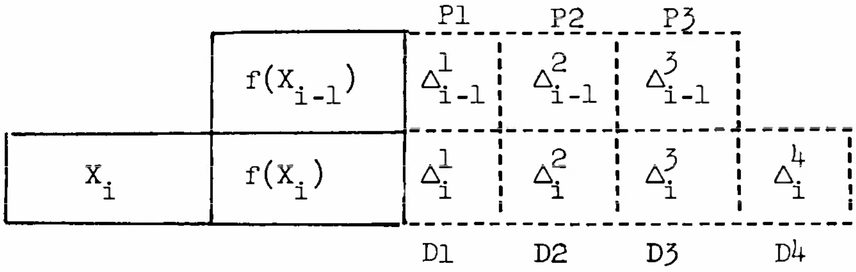
```
READ(1000)NOS.AT(F1) %   NF = 09
```

To include the program check we write

```
IF-INT(NX = NF)GOTO(WORK)%
```

This conditional statement is immediately followed by a PRINT statement which identifies and prints the number of X's and the number of f's in the event there is a discrepancy.

To determine the internal storage space required for the four differences of a given row, we illustrate schematically below the quantities required for the computation and printing. We plan to compute a row of differences and print that row immediately.



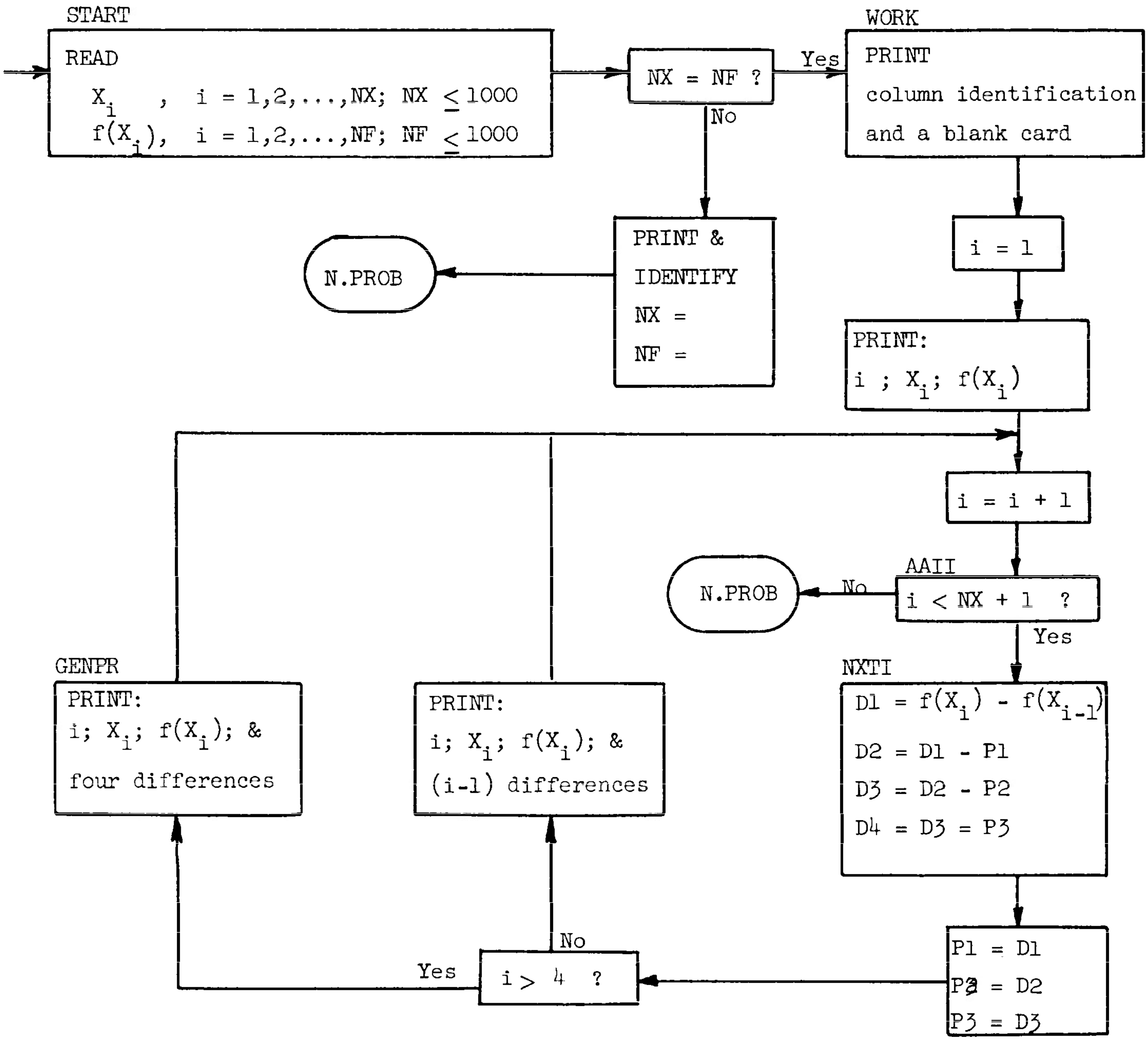
If we let D1,D2,D3 and D4 be the names of the four differences in the ith row and P1,P2,P3 the names of the required differences in the (i-1)st row, we can write

```
D1 = F,I - F,(I-1)
D2 = D1 - P1
D3 = D2 - P2
D4 = D3 - P3
```

where we prepare in advance for the computations in the next row by writing

```
P1 = D1
P2 = D2
P3 = D3
```

so that these quantities are available for the computations of the next row of differences. (We will make special provisions for printing rows with undefined differences). We have used the above notation in the flow-chart illustrated in FIGURE 29.



FLOW CHART FOR EXAMPLE 14.

FIGURE 29.

```
                                                            23 JULY,63 PAGE    1
        PROB C906  M.J.ROMANELLI  45107 EXAMPLE 14.
        BLOC(X-X1000)F-F1000)D1-D4
START       READ(1000)NOS.AT(X1)% NX=09% READ(1000)NOS.AT(F1)% NF=09
            IF-INT(NX=NF) GOTO(WORK)
            PRINT-FORMAT(SEE1.)-<NUMBER OF X,S = >NX
            PRINT-FORMAT(SEE1.)-<NUMBER OF F,S = >NF% GOTO(N.PROB)
WORK        PRINT<  I      X,I       F,I      1DELI     2DELI     3DELI>
        CONT<     4DELI>% ENTER(PRINTB)% SET(I=1)%
            PRINT-FORMAT(SEE1.)-(I)X,I)F,I
AATI        COUNT(NX+1)IN(I)GOTO(N.LINE)% GOTO(N.PROB)
N.LINE      D1=F,I-F,(I-1)% D2=D1-P1% D3=D2-P2% D4=D3-P3
            P1=D1% P2=D2% P3=D3%  IF-INT(I>4) GOTO(GENPR)
            PRINT-FORMAT(SEE1.)-(I)X,I)F,I)(I-1)NOS.AT(D1)% GOTO(AATI)
GENPR       PRINT-FORMAT(SEE1.)-(I)X,I)F,I) (4)NOS.AT(D1)% GOTO(AATI)
SEE1.   FORM(4-5)3-2)1-1)12-1-7)3-2)1-6)2
        END GOTO(START)
 93000000 00 95000000 00 98000000 00 10000000 01 12000000 01 12500000 01
 13000000 01 13500000 01 14000000 01 15000000 01 16000000 01 16500000 01
 17000000 01 17500000 01 18000000 01 18500000 01 19000000 01 19300000 01
 19800000 01 20000000 01

 59783000 00 58168000 00 55702000 00 54030000 00 36236000 00 31532000 00
 26750000 00 21901000 00 16997000 00 70740000-01-29200000-01-79120000-01
-12884000 00-17825000 00-22720000 00-27559000 00-32329000 00-35153000 00
-39788000 00-41615000 00
```

```
MAY.23,63  BRLESC  FORAST F62
     PROB C906  M.J.ROMANELLI  45107 EXAMPLE 14.                                  *

 I       X,I       F,I      1DELI      2DELI      3DELI      4DELI                0000001


   1    .9300     .5978
   2    .9500     .5817   - .0161
   3    .9800     .5570   - .0247   - .0085
   4   1.0000     .5403   - .0167     .0079      .0165
   5   1.2000     .3624   - .1779   - .1612   - .1692   - .1856
   6   1.2500     .3153   - .0470     .1309      .2921      .4613
   7   1.3000     .2675   - .0478   - .0008   - .1317   - .4238
   8   1.3500     .2190   - .0485   - .0007     .0001      .1318
   9   1.4000     .1700   - .0490   - .0005     .0001      .0000
  10   1.5000     .0707   - .0992   - .0502   - .0496   - .0498
  11   1.6000   - .0292   - .0999   - .0007     .0495      .0991
  12   1.6500   - .0791   - .0499     .0500      .0507      .0013
  13   1.7000   - .1288   - .0497     .0002   - .0498   - .1006
  14   1.7500   - .1783   - .0494     .0003      .0001      .0499
  15   1.8000   - .2272   - .0489     .0005      .0002      .0000
  16   1.8500   - .2756   - .0484     .0006      .0001   - .0001
  17   1.9000   - .3233   - .0477     .0007      .0001      .0000
  18   1.9300   - .3515   - .0282     .0195      .0188      .0186
  19   1.9800   - .3979   - .0464   - .0181   - .0376   - .0563
  20   2.0000   - .4162   - .0183     .0281      .0462      .0838
  21    .0000     .0000     .4161      .4344      .4063      .3602
  22    .0000     .0000     .0000   - .4162   - .8506   -1.2569
  23    .0000     .0000     .0000      .0000      .4161     1.2667
  24    .0000     .0000     .0000      .0000      .0000   - .4162
```

FIGURE 30.

Note that we have obtained four extraneous lines in the table, i.e., correspond to I = 21,22,23 and 24. To eliminate these, we could have used a second terminating condition, namely:

```
IF (X1,I = F1,I = 0)GOTO(N.PROB) .
```

This should be inserted after the PRINT statement on line 9 or on a card immediately preceding card 10.

EXAMPLE 15.

This example illustrates several of the convenient ENTER statements. They provide a means of evaluating functions which may require more than one argument and for which more than one result may be produced. As in the class of single valued functions of one argument, most of these functions require floating-point arguments and yield floating-point results. Exceptions will be explicitly stated in the descriptions which follow. Arguments for the trigonometric functions and results for the inverse trigonometric functions are in radian units.

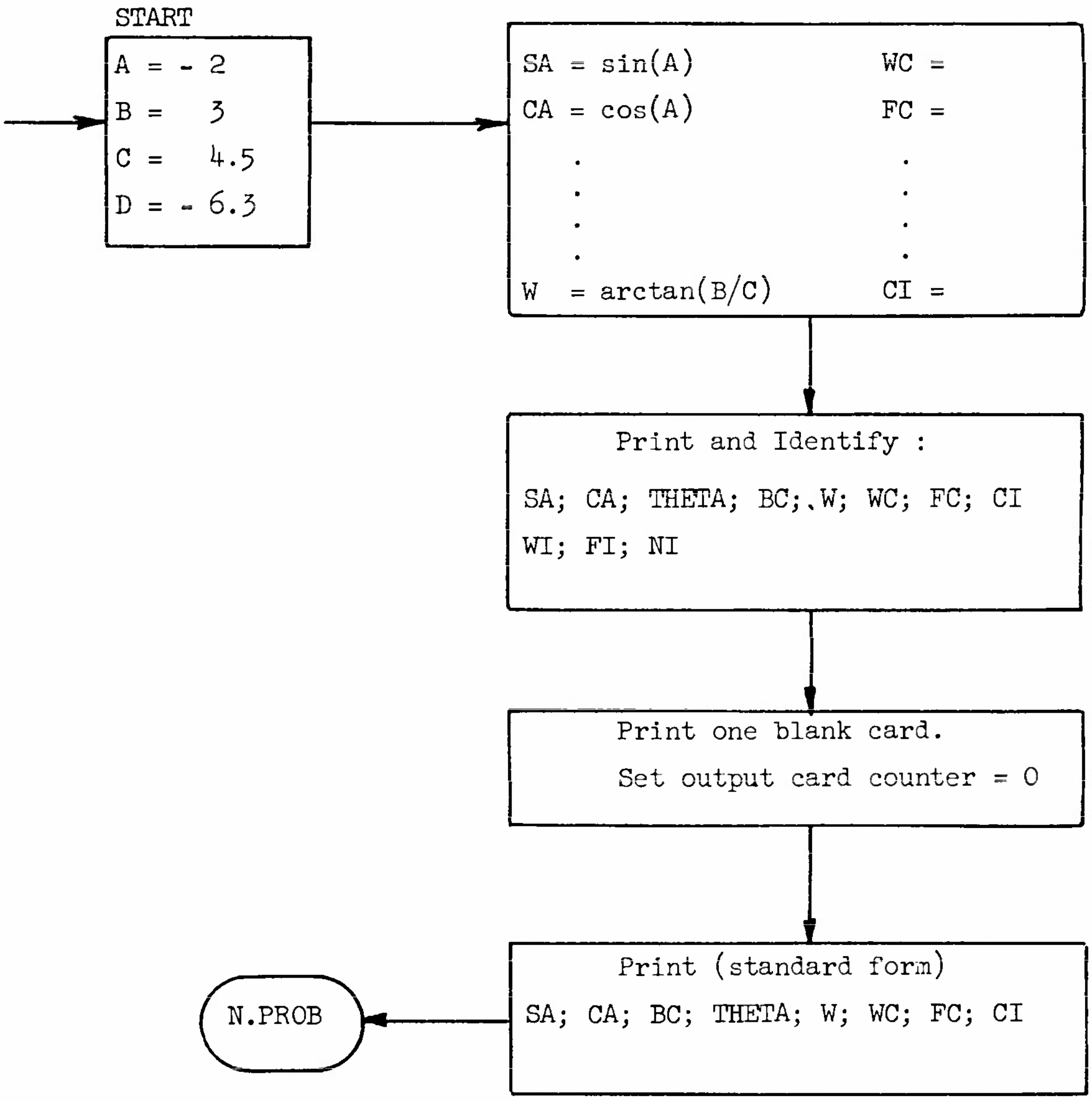
GIVEN:

A = - 2
B = 3
C = 4.5
D = - 6.3

REQUIRED: Compute, print and identify the following:

SA = sin(A)	WC = whole part of C	floating-pt representations
CA = cos(A)	FC = fractional part of C	
BC = B^C	WI = integer form of C	
θ = arcsin(SA)	FI = integer form of FC	integer representations
and arccos(CA)	NI = integer form of D	
W = arctan(B/C)	CI = floating-point of NI	floating-pt representation

The flow chart for this example is shown in FIGURE 31.



FLOW CHART FOR EXAMPLE 15

FIGURE 31.

The statement

```
ENTER(SINCOS)A)SA)CA)
```

tells the computer to compute both the sine and cosine of A and to assign the results to SA and CA respectively.

The statement

```
ENTER(POWER)B)C)BC
```

tells the computer to raise the quantity B to the C power and to assign the result to BC. Since the logarithm function is used to obtain this result, $B > 0$.

The statement

```
ENTER(ARCSC)SA)CA)THETA
```

tells the computer to determine the angle THETA whose arcsin and arccos are respectively SA and CA.

$$(-\pi \leq \text{THETA} \leq \pi) .$$

The statement

```
ENTER(ARTAN)B)C)W
```

tells the computer to determine the angle W whose arctan is B/C. $(-\pi \leq W \leq \pi)$.

The statement

```
ENTER(WH.FRA)C)WC)FC
```

tells the computer to separate the floating-point quantity C into its whole and fractional parts, assigning the whole part to WC and the fractional part to FC. Both parts are recorded in floating-point form.

The statement

```
ENTER(CVFTOI)C)WI
```

tells the computer to convert the floating-point quantity C to integer form, assigning the result to WI. (For rounding purposes, the quantity 10^{-4} is first added to |C| before the conversion is effected).

The statement

```
ENTER(CVITOF)NI)CI
```

tells the computer to convert the integer quantity NI to floating-point form, assigning the result to CI.

The statement

```
ENTER(PRINT B)
```

tells the computer to print one blank card.

The statement

```
ENTER(ZEROCC)
```

tells the computer to set the card counter equal to zero. When printing standard card formats, the successive cards produced contain serial identification in columns 77 through 80, (output card count). The serial count commences at one with the first output card produced and is advanced by one with each successive card produced until either the count reaches 9999 or when restored to zero by the above ENTER statement.

Non-standard formats may also include the output card count, however, it must be specified in the format. We used a special format for the integer quantities that are to be printed. The format

```
K    FORM    (4-4)4-4)4-4)2
```

is of the form

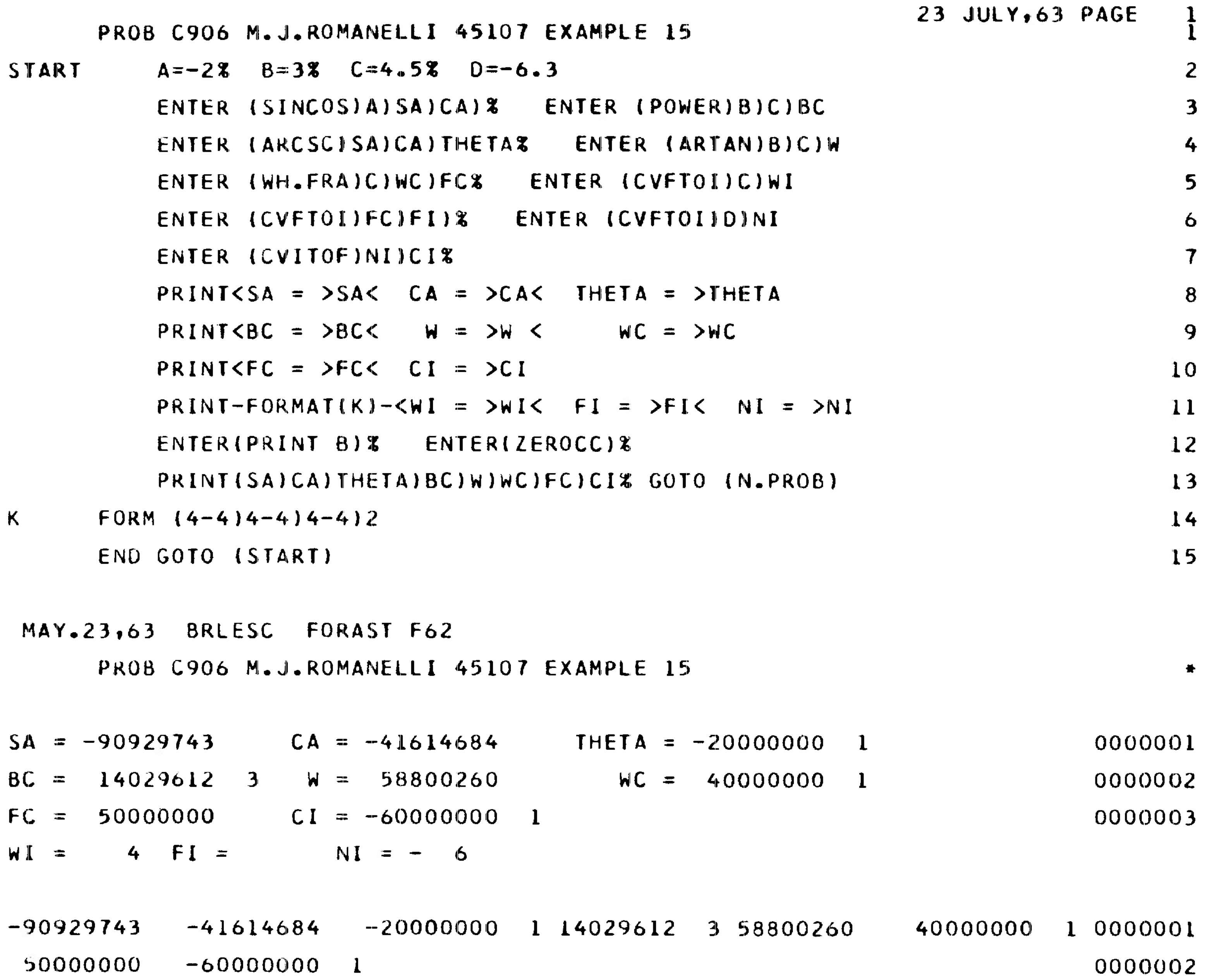
```
(T-L)T-L)T-L)
```

where in each case the length is 4 and the type is 4. The 4 type tells the computer that the quantity exists internally as an integer and is to be represented in integer form in the four columns allocated, the units position on the card corresponding to the fourth column. The characters specified within the < and > of the PRINT statements supercede the column allocation specified in the format; i.e., corresponding to the statement

PRINT-FORMAT(K) - <WIb=b > WI < bbFI >----

WI is first recorded in columns 1 and 2, then a blank in column 3, the equality in column 4, a blank in column 5 and then the integer in columns 6,7,8 and 9 with insignificant zeroes suppressed, (i.e., replaced by blanks). Next, blanks are recorded in columns 10 and 11, FI recorded in 12 and 13, etc.

The program, input and output for this example are listed in FIGURE 32.



```
                                                                    23 JULY,63 PAGE    1
      PROB C906 M.J.ROMANELLI 45107 EXAMPLE 15
START      A=-2%  B=3%  C=4.5%  D=-6.3
           ENTER (SINCOS)A)SA)CA)%   ENTER (POWER)B)C)BC
           ENTER (ARCSC)SA)CA)THETA%   ENTER (ARTAN)B)C)W
           ENTER (WH.FRA)C)WC)FC%   ENTER (CVFTOI)C)WI
           ENTER (CVFTOI)FC)FI)%   ENTER (CVFTOI)D)NI
           ENTER (CVITOF)NI)CI%
           PRINT<SA = >SA<  CA = >CA<  THETA = >THETA
           PRINT<BC = >BC<   W = >W <     WC = >WC
           PRINT<FC = >FC<  CI = >CI
           PRINT-FORMAT(K)-<WI = >WI<  FI = >FI<  NI = >NI
           ENTER(PRINT B)%   ENTER(ZEROCC)%
           PRINT(SA)CA)THETA)BC)W)WC)FC)CI% GOTO (N.PROB)
K      FORM (4-4)4-4)4-4)2
      END GOTO (START)
```

```
 MAY.23,63  BRLESC  FORAST F62
      PROB C906 M.J.ROMANELLI 45107 EXAMPLE 15                                           *

SA = -90929743      CA = -41614684      THETA = -20000000  1                      0000001
BC =  14029612  3    W =  58800260         WC =  40000000  1                      0000002
FC =  50000000      CI = -60000000  1                                              0000003
WI =    4  FI =        NI = -  6

-90929743   -41614684   -20000000  1 14029612  3 58800260    40000000  1 0000001
 50000000   -60000000  1                                                  0000002
```

FIGURE 32.

EXAMPLE 16.

In many problems, the given functions are not always known analytically and in lieu of explicit analytic definitions discrete tabular data is given. Similarly, as illustrated in the previous examples, many solutions obtained from the computers are expressed in tabular form. Hence, in many problems conventional interpolation is often desired. This example is designed to illustrate how the ENTER statement provides for interpolation. The statement has the general form

```
ENTER(D.D.IN)X)FX)X1)F1)tpt)n)ix)if)%
             1  2  3  4  5  6  7  8
```

where D.D.IN is an abbreviation for divided difference interpolation. As many as 8 parameters may be specified to describe the particular interpolation desired. It is assumed that a table of discrete values, X_i and $f(X_i)$, is recorded in internal storage, where i = 1,2,...,tpt and the X_i are in monotone increasing or monotone decreasing sequence. * It is further assumed that the tabular values are uniformly spaced in internal storage. (This does not mean that $X_{i+1} = X_i + \Delta X$, the uniformity refers to the space allocation in internal storage). Further, the uniform spacing of the $f(X_i)$ may indeed be different from the uniform spacing of the independent variable, X_i. In general, if we want the computer to perform particular interpolations, we must specify the pertinent information. The parameters which follow the ENTER(D.D.IN) tell the computer the specific information that is required for the interpolation. The meanings for the parameters are as follows:

X is the name of the argument, i.e., we want the interpolated value for f(X);

FX is the name of the result, i.e., the result obtained is assigned as the existing value of FX and hence future references to this

* Since divided differences are used, $X_i \neq X_{i+1}$, for all i.

result are made by referring to the name FX;

X1 is the name of the first entry in the table of the independent variable;

F1 is the name of the first functional value in the table;

tpt is an integer which specifies the number of points in the table, i.e., the maximum value of i;

n is an integer which specifies the number of points to be used in the interpolation, i.e., if $n = 2$, linear interpolation is desired, if $n = 3$, parabolic interpolation is desired, etc., $n \leq 18$;

ix is an integer which specifies the uniform spacing of the X_i in internal storage;

if is an integer which specifies the uniform spacing of the $f(X_i)$ in internal storage.

Note that the dependent and independent tabular data need not have the same uniform spacing. (The last three parameters, n, ix and if may be omitted <u>only if</u> ix = if = 1 and n = 5). In general, the lower case letters listed in the string of parameters of ENTER statements refer to integer values. As such, the integers may be expressed explicitly or may be expressed as the existing values of indices provided the index name is preceded by the mandatory comma. If for example, the total number of points in a table is 92, one may literally write 92 in the parenthesis corresponding to tpt. As an alternative, if one used the name J as an index with existing value 92, for example, SET(J = 92), one may write ,J in the parentheses corresponding to tpt.

GIVEN:

x_j and $f(x_j)$, $(j = 1,2,\ldots,10)$. y_i, $i = 1,2,3,4,5$.

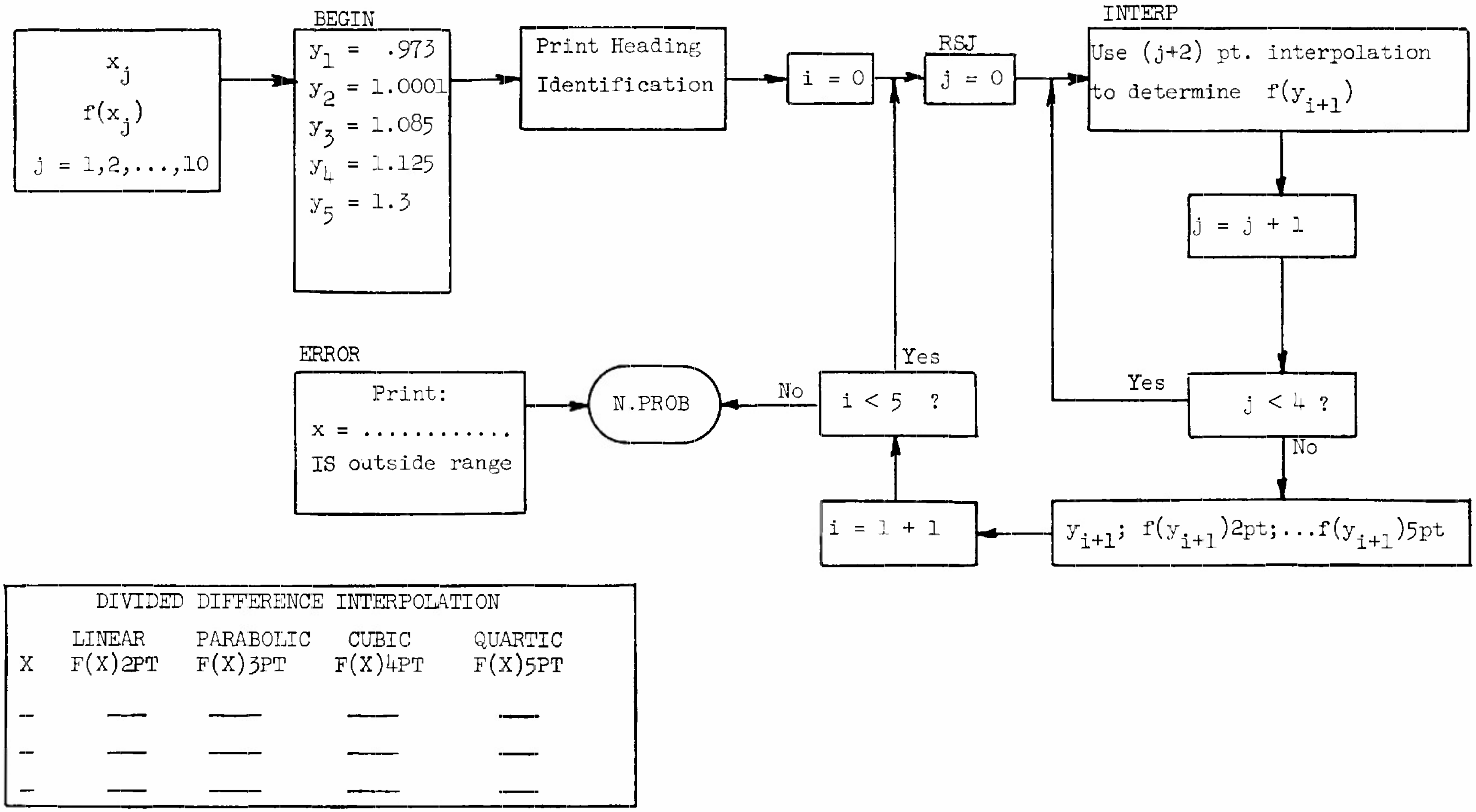
$x_1 < y_i < x_{10}$ for $i = 1,2,3,4$, $y_5 > x_{10}$

REQUIRED:

For each y_i, determine $f(y_i)$ using 2,3,4 and 5 point interpolation.

Print and Identify: x; f(x) 2 pt; f(x) 3 pt; f(x) 4pt; and f(x) 5 pt. where x corresponds to the y_i.

The flow-chart for this example is shown in FIGURE 33.



FLOW CHART FOR EXAMPLE 16.

FIGURE 33.

One other general remark concerning the interpolation function is that, extrapolation will be performed provided that the point is not "too far" outside the given range. Indeed "too far" means that the point must lie within "one interval" at either end of the table. Explicitly this means that extrapolation for the point X will be performed provided

$$X_{tpt} < X < X_{tpt} + (X_{tpt} - X_{tpt-1})$$

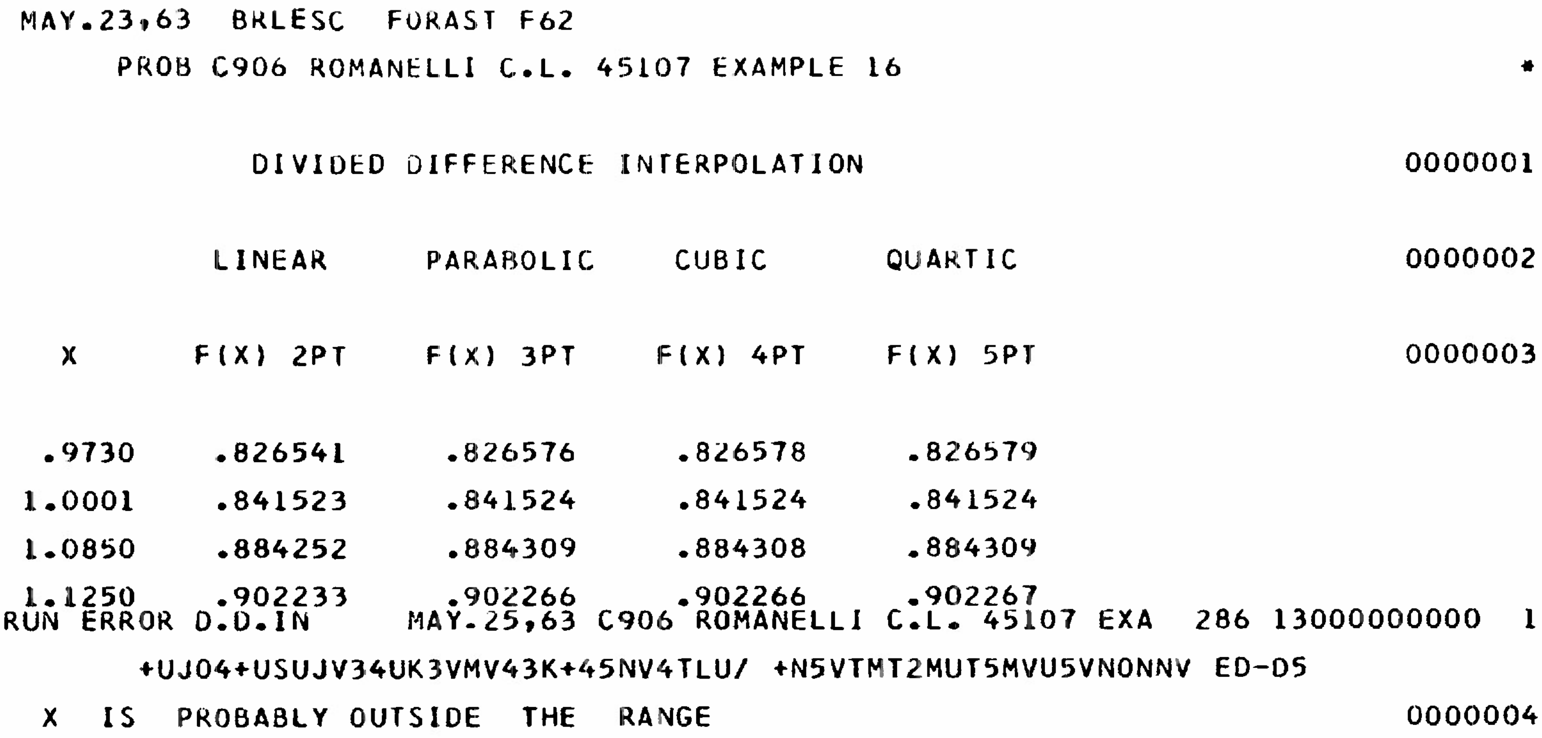
or at the other end,

$$X_1 - (X_2 - X_1) < X < X1$$

for monotone increasing X_i. Similar conditions hold for extrapolations for monotone decreasing X_i.

The program, input and output for this example is listed in FIGURE 34.

```
                                                   23 JULY,63 PAGE
          PROB C906 ROMANELLI C.L. 45107 EXAMPLE 16
          BLOC (Y1-Y5)IF2-IF5)X1-X20
          SYN (F1=X2)
BEGIN        Y1=.973% Y2=1.0001% Y3=1.085% Y4=1.125% Y5=1.3%
             PRINT<                DIVIDED DIFFERENCE INTERPOLATION>
             ENTER(PRINT B)
             PRINT<              LINEAR      PARABOLIC     CUBIC       QUARTIC>
             ENTER(PRINT B)
             PRINT<    X       F(X) 2PT     F(X) 3PT     F(X) 4PT     F(X) 5PT>
             ENTER(PRINT B)% SET(I=0)
RSJ          SET(J=0)
INTERP       ENTER(D.D.IN)Y1,I)IF2,J)X1)F1)10),J+2)2)2)
             COUNT(4)IN(J)GOTO(INTERP)
             PRINT-FORMAT(H)-(Y1,I)(4)NOS.AT(IF2)
             COUNT(5)IN(I)GOTO(RSJ)% GOTO(N.PROB)
H         FORM(12-1-7)3-2)1-1)12-1-9)3-3)1-4)2
X1        DEC (.95).81342).96).81919).98).83050)1).84147)1.02).85211
          CONT(1.05).86742)1.06).87236)1.09).88663)1.11).89570)1.13).90441
ERROR        PRINT<  X  IS  PROBABLY OUTSIDE  THE  RANGE >% GOTO(N.PROB)%
          END GOTO(BEGIN)
```



```
MAY.23,63  BRLESC  FORAST F62
     PROB C906 ROMANELLI C.L. 45107 EXAMPLE 16                              *

            DIVIDED DIFFERENCE INTERPOLATION                              0000001

          LINEAR      PARABOLIC    CUBIC      QUARTIC                     0000002

  X      F(X) 2PT    F(X) 3PT    F(X) 4PT    F(X) 5PT                     0000003

 .9730    .826541     .826576     .826578     .826579
1.0001    .841523     .841524     .841524     .841524
1.0850    .884252     .884309     .884308     .884309
1.1250    .902233     .902266     .902266     .902267
RUN ERROR D.D.IN     MAY.25,63 C906 ROMANELLI C.L. 45107 EXA   286 13000000000   1
      +UJ04+USUJV34UK3VMV43K+45NV4TLU/ +N5VTMT2MUT5MVU5VNONNV ED-D5
 X   IS   PROBABLY OUTSIDE  THE   RANGE                                   0000004
```

FIGURE 34.

A study of the program for this example will illustrate some new concepts. First, we could have used the general READ statement to record the given tabular data in internal storage. However, to illustrate another means of recording data in internal storage we have used the

DEC order-type word.

Note you will find on line 17, X1 in the LOCATION columns and DEC in the order-type columns. Beginning in column 11 we have written the numerical values that correspond to X1, f(X1), X2, f(X2), etc. Each numerical value is separated by a parenthesis. These numerical quantities are recorded in consecutive spaces in internal storage. The first numerical value, .95, is associated with the name X1 recorded in the location columns. This fact alone does not establish X2, X3, etc. as valid names. To associate these quantities with particular names for future references we have used BLOC and SYN statements. In the BLOC statement we wrote (X1 - X20) to reserve space and establish X names for these twenty quantities. In the SYN statement we wrote (F1 = X2) indicating that the second numerical quantity, .81342 was to be associated with either name, X2 or F1. Note, also, that under this ordering and arrangement, the X's are two units apart in internal storage. Likewise the corresponding functional values are two units apart in internal storage. These concepts permit us to establish the data and names required for the interpolation. We will make reference to X1, and F1 and specify that the tabular data is two units apart both for the X values and F values.

We used names IF2 - IF5 for the interpolated functional values. We have also used a non-standard format

H FORM(12-1-7)3-2)1-1)12-1-9)3-3)1-4)2

for the entries in the output table. The form (12-1-7) specifies that the numerical value of Y1,I exists in internal storage in floating-point form, the external form is to be printed with a decimal point after one digit and 7 columns are to be used for entire external representation. The form (3-2) indicates that 2 blank spaces are to precede the next quantity printed.

The form (1-1) specifies that the preceding forms are to be used just 1 time. This was inserted since we will indicate repetitions of another form and repetitions commence with the form immediately following the previous repetition form. In the above example, the repetition from (1-1) applies to the (12-1-7)3-2 whereas the (1-4) repetition form applies to the (12-1-9)3-3).

The form (12-1-9) is similar to the initial form (12-1-7), the only difference is in the total number of columns to be allocated for the quantity.

The form (3-3) indicates 3 spaces.

The form (1-4) indicates that the total form (12-1-9)3-3) is to be used for 4 consecutive quantities, namely the four interpolated functional values, IF2, IF3, IF4 and IF5.

The form)2 terminates the format definition.

EXAMPLE 17.

This example illustrates how the ENTER statement is used to solve a system of n linear equations in n unknowns. In matrix notation, the system is expressed as

$$AX = B$$

or

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ \vdots & & & & \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{pmatrix} \begin{pmatrix} X_1 \\ X_2 \\ \vdots \\ X_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

The problem in general is to determine the $X_1, X_2, \ldots, X_n$ which satisfy the above system when the elements of A and B are given. To use the convenient ENTER statement to solve such a system, the given elements must be recorded in consecutive spaces of internal storage by <u>rows</u>. That is, the prescribed order is as follows:

$$a_{11}, a_{12}, a_{13}, \ldots, a_{1n}, b_1, a_{21}, a_{22}, \ldots, a_{2n}, b_2, \ldots\ldots, a_{nn}, b_n \ .$$

One should reserve spaces for the n x (n+1) consecutive given elements, the A and B, by using the BLOC statement. Here, one may define a linear BLOC, say A1 - A$\{$n x (n+1)$\}$ or a two dimensional array, A1,1 - A$\{$n , n+1$\}$. For example, for a system of four equations in four unknowns one may choose the linear bloc, A1 - A20, or the two dimensional array A1,1 - A4,5. In either case, one must be consistent when writing the ENTER statement which refers to the given elements. The ENTER statement has the general form

ENTER(S.N.E.)A)n)C)D

where:

S.N.E. is simply an abbreviation for Solve Normal Equations;

A is the name of the first given element, (a_{11});

n is an integer (or name of an integer) which specifies the number of unknowns;

C is optional and if specified is the name where the result of the first unknown is to be recorded, i.e., generally the name X1 where the solution is to be recorded. The successive results are recorded in consecutive spaces in internal storage. The identical results are available elsewhere, these are explained below.

D is the name of the determinant of A This name is optional, and if specified, the value of the determinant of A is calculated and recorded at the place called D.

In the process of obtaining the solution, the computer destroys the given A and B. In particular, the original A is replaced by the inverse of A, and B is replaced by the solution, X, which satisfies the given system. Hence one need not specify C since one can make references to the solution by referring to the names of the original B's. However, if one wants to specify D and not C then the ENTER statement must be written in the form

```
ENTER(S.N.E.)A)n))D
```

To illustrate, we chose the simple system of four equations in four unknowns with solution X1 = 1, X2 = 2, X3 = 3, X4 = 4.

GIVEN:

$$\begin{array}{ccccccccc} X1 & + & X2 & + & X3 & + & X4 & = & 10 \\ X1 & + & 2X2 & + & X3 & + & X4 & = & 12 \\ X1 & + & X2 & + & 3X3 & + & X4 & = & 16 \\ X1 & + & X2 & + & X3 & + & 4X4 & = & 22 \end{array}$$

REQUIRED:

Determine, identify and print the X1,X2,X3 and X4 which satisfy the above equations.

Note that the given elements in the prescribed order by rows are:

1)1)1)1)10)1)2)1)1)12)1)1)3)1)16)1)1)1)4)22 .

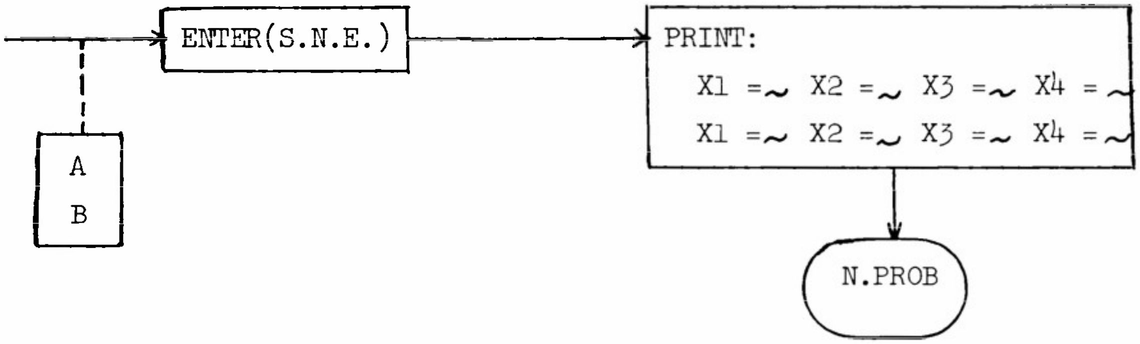
To reserve space in internal storage for the given elements and the solution, we write a BLOC statement

BLOC(A1,1 - A4,5)X1 - X4)

To record the given elements in the reserved space we use the convenient DEC order-type word and identify the first element by writing A1,1 in the LOCATION space.

A1,1 DEC (1)1)1)1)10)1)2)1)1)12)1)1)3)1)16)1)1)1)4)22

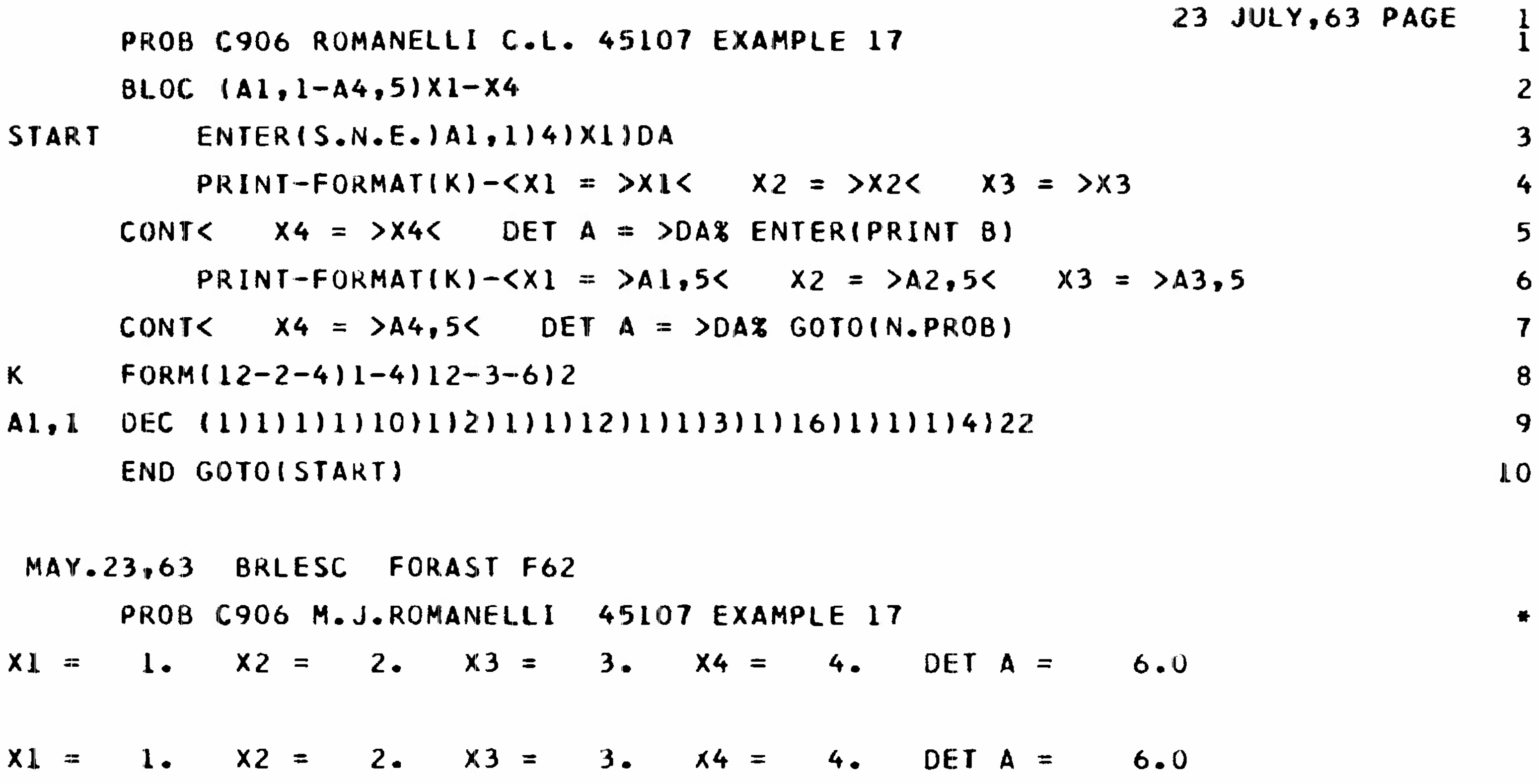
The BLOC statement above provides for reserving the consecutive space for the given elements and solution. The "A1,1 DEC" provides for recording the given system in internal storage in the reserved space. The flow chart for this example is given below.



FLOW CHART - EXAMPLE 17

FIGURE 34.

Note that at START it is assumed that the given A and B are recorded internally in the prescribed order. This is indicated on the flow chart just prior to START with the box containing A and B appended with a broken line. Note also that we will print and identify the solution twice. This is done to emphasize that if one specifies a place for the solution (as we do in our program) then the solution results are available in two places. The two prints correspond to the solution references in the two distinct places. The program and output for this example are listed in FIGURE 35.



```
                                                           23 JULY,63 PAGE    1
      PROB C906 ROMANELLI C.L. 45107 EXAMPLE 17
      BLOC (A1,1-A4,5)X1-X4
START     ENTER(S.N.E.)A1,1)4)X1)DA
          PRINT-FORMAT(K)-<X1 = >X1<   X2 = >X2<   X3 = >X3
      CONT<   X4 = >X4<   DET A = >DA% ENTER(PRINT B)
          PRINT-FORMAT(K)-<X1 = >A1,5<   X2 = >A2,5<   X3 = >A3,5
      CONT<   X4 = >A4,5<   DET A = >DA% GOTO(N.PROB)
K     FORM(12-2-4)1-4)12-3-6)2
A1,1  DEC (1)1)1)1)10)1)2)1)1)12)1)1)3)1)16)1)1)1)4)22
      END GOTO(START)

 MAY.23,63  BRLESC  FORAST F62
      PROB C906 M.J.ROMANELLI  45107 EXAMPLE 17                              *
X1 =   1.   X2 =   2.   X3 =   3.   X4 =   4.   DET A =   6.0


X1 =   1.   X2 =   2.   X3 =   3.   X4 =   4.   DET A =   6.0
```

FIGURE 35.

EXAMPLE 18.

In this example we will illustrate another means of obtaining the solution to the problem of EXAMPLE 17. Since the A matrix (the coefficient matrix) is symmetric, less storage space is required and computation time is proportionally diminished. For low order systems, the space and time savings may not be appreciable; however, for high order systems, both space and time savings may be appreciable.

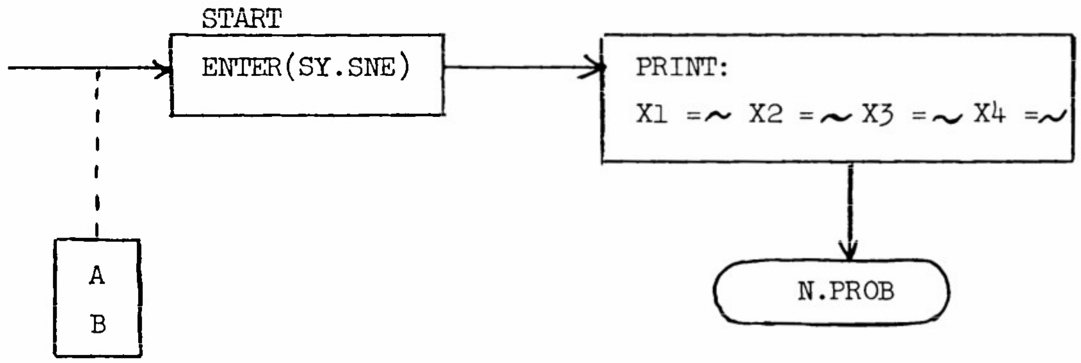
Similar to the general case, the symmetric case also requires that the given elements be recorded internally in consecutive spaces by rows. In particular the required order is as follows:

$$a_{11}, a_{12}, \ldots, a_{1n}, b_1, a_{22}, a_{23}, \ldots, a_{2n}, b_2, \ldots, b_{n-1}, a_{nn}, b_n \ .$$

The symmetry condition is specified in both the BLOC statement and the ENTER statement as follows:

```
BLOC(A1,1 - A4,5/SY.)X1 - X4

ENTER(SY.SNE)A1,1)4)X1)D
```

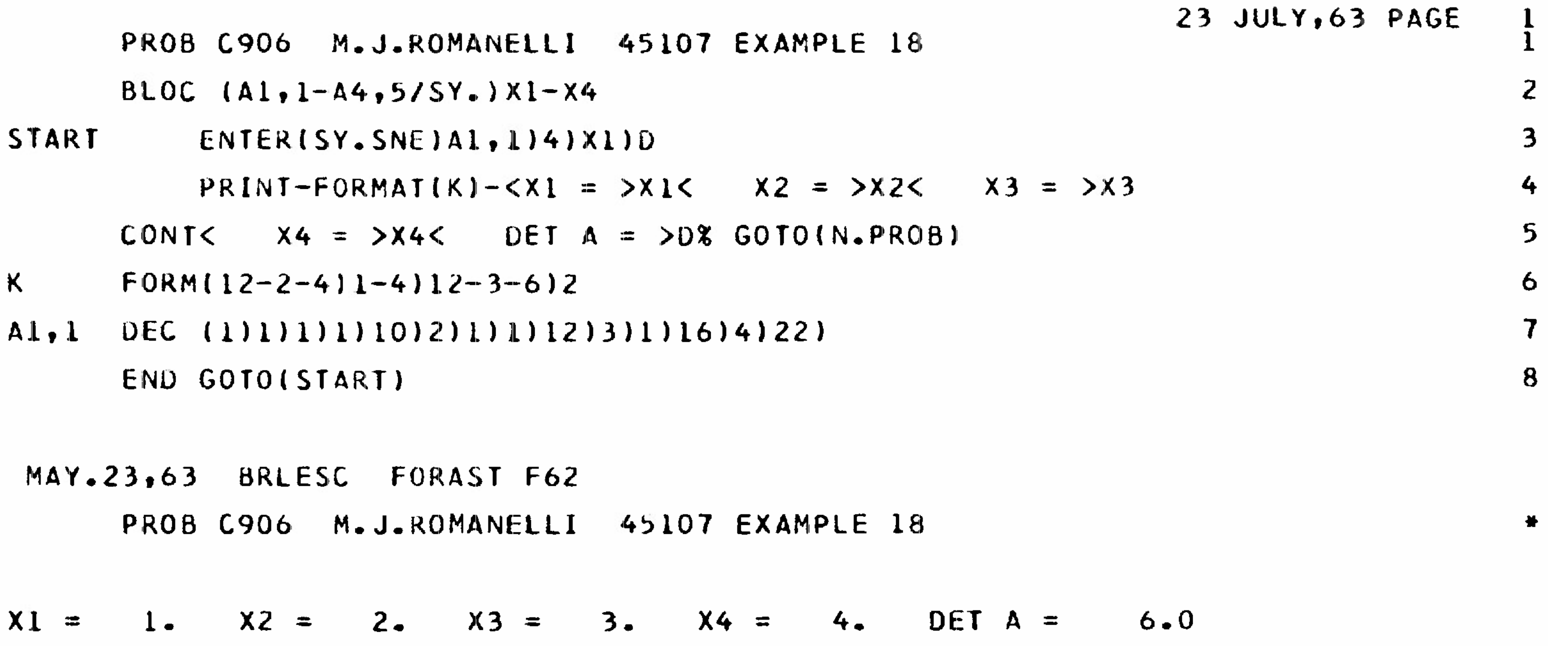
Again, as in the general case, in obtaining the solution, the original A matrix is replaced by the inverse of A; however, the solution is recorded only in the specified place X1, and is not recorded in the space previously occupied by the augmented vector as in the general S.N.E. The flow chart for this symmetric case is similar to that of the general case.



FLOW CHART FOR EXAMPLE 18.

FIGURE 36.

The program and results for this example are listed in FIGURE 37.



```
                                                                    23 JULY,63 PAGE    1
        PROB C906  M.J.ROMANELLI  45107 EXAMPLE 18
        BLOC (A1,1-A4,5/SY.)X1-X4
START       ENTER(SY.SNE)A1,1)4)X1)D
            PRINT-FORMAT(K)-<X1 = >X1<   X2 = >X2<   X3 = >X3
        CONT<   X4 = >X4<   DET A = >D% GOTO(N.PROB)
K       FORM(12-2-4)1-4)12-3-6)2
A1,1    DEC (1)1)1)1)10)2)1)1)12)3)1)16)4)22)
        END GOTO(START)

 MAY.23,63  BRLESC  FORAST F62
        PROB C906  M.J.ROMANELLI  45107 EXAMPLE 18                                     *

X1 =   1.   X2 =   2.   X3 =   3.   X4 =   4.   DET A =   6.0
```

FIGURE 37.

EXAMPLE 19.

This example illustrates the ENTER statement which applies Simpson's rule to approximate the definite integral

$$\int_A^B f(x)dx \ .$$

The statement has the general form

ENTER(S.INTE)F)X)FX)I)A)B)E

where:

S.INTE is an abbreviation for Simpson integration;

F is the location name where the integrand f(x) is explicitly defined;

Since the process is designed to provide the approximation for arbitrary f(x), it is the responsibility of the programmer to define the f(x) of interest. Further, the statement or statements which define f(x) must be terminated with GOTO(S.I.FF). This enables the computer to evaluate and weight the integrand as many times as is necessary.

FX is the name of the integrand, f(x);

I is the name of the resulting approximation;

A is the name of the lower limit, B the upper limit;

E is the name of a "relative error bound;

The program is designed to obtain the approximation with a minimum number of evaluations of the integrand. Successive approximations are obtained using successively smaller ΔX until

$$\left| \frac{I(\Delta X) - I(\frac{\Delta X}{2})}{I(\Delta X)} \right| < E$$

or until

$$\left|\frac{B - A}{\frac{\Delta X}{2}}\right| \geq \quad 1023.$$

When the latter occurs, a "run error" print includes the numerical value of

$$\left|\frac{I(\Delta X) - I(\frac{\Delta X}{2})}{I(\Delta X)}\right| \quad .$$

(For approximations which tend to zero, defining the original integral as a sum of integrals may suffice to produce a satisfactory approximation).

GIVEN:

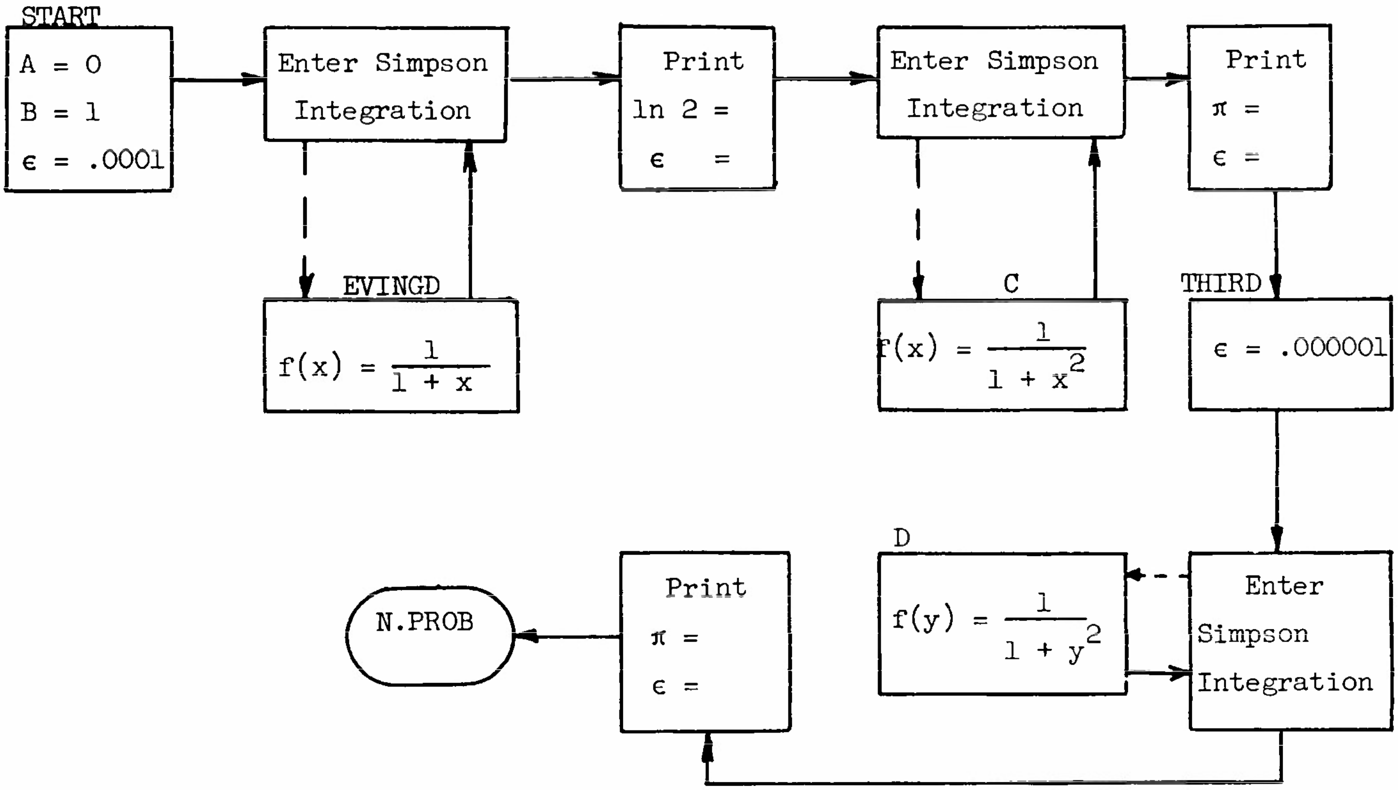
$$\ell n(2) = \int_0^1 \frac{dx}{1 + x} \qquad \pi = 4 \int_0^1 \frac{dx}{1 + x^2}$$

REQUIRED:

Use Simpson Method of Numerical Integration to obtain approximations for:

LN(2) with an associated E = .0001 ;
π " " " E = .0001 ;
π " " " E = .000001 ;

PRINT AND IDENTIFY RESULTS.



FLOW CHART FOR EXAMPLE 19.

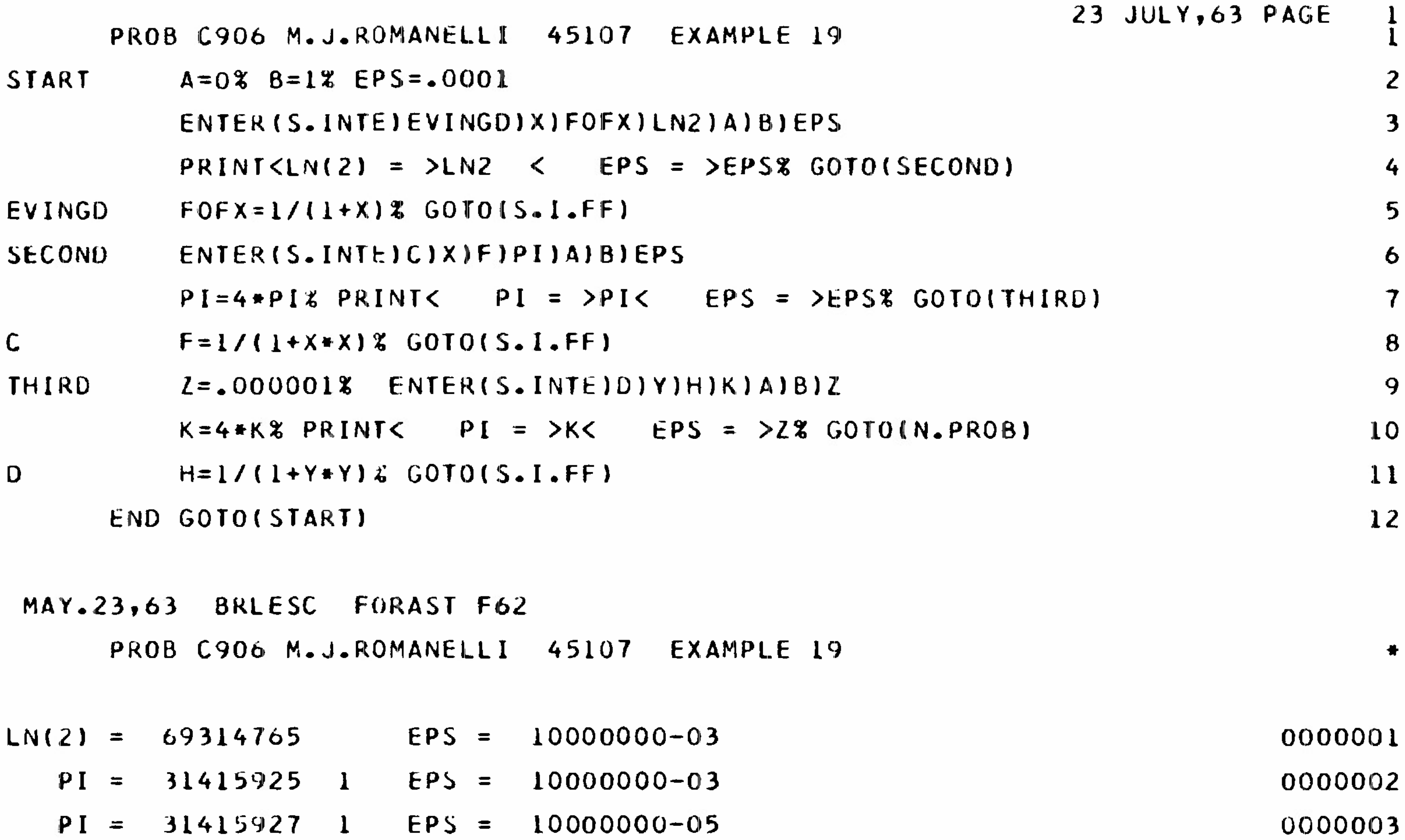
FIGURE 38.

Note the "broken lines" to the evaluation of the integrands and the "solid" line from the evaluation back to the Simpson integration. We used the "broken lines" because the programmer does not write an explicit statement which says GOTO(EVINGD). Recall that we specify this connection in the ENTER statement.

We used the solid line returning from the evaluation since the programmer does write GOTO(S.I.FF).

Note the solid lines from the "integration boxes" to the "print boxes" which are used to indicate that the computers go to the statement immediately following the ENTER statement after the ENTER statement is completely executed. Hence, evaluation of an integrand associated with an ENTER statement should not immediately follow the ENTER statement.

The program and output for this example are listed in FIGURE 39.



```
                                                           23 JULY,63 PAGE    1
        PROB C906 M.J.ROMANELLI  45107  EXAMPLE 19
START       A=0% B=1% EPS=.0001
            ENTER(S.INTE)EVINGD)X)FOFX)LN2)A)B)EPS
            PRINT<LN(2) = >LN2  <   EPS = >EPS% GOTO(SECOND)
EVINGD      FOFX=1/(1+X)% GOTO(S.I.FF)
SECOND      ENTER(S.INTE)C)X)F)PI)A)B)EPS
            PI=4*PI% PRINT<   PI = >PI<   EPS = >EPS% GOTO(THIRD)
C           F=1/(1+X*X)% GOTO(S.I.FF)
THIRD       Z=.000001%  ENTER(S.INTE)D)Y)H)K)A)B)Z
            K=4*K% PRINT<   PI = >K<   EPS = >Z% GOTO(N.PROB)
D           H=1/(1+Y*Y)% GOTO(S.I.FF)
        END GOTO(START)

 MAY.23,63  BRLESC  FORAST F62
        PROB C906 M.J.ROMANELLI  45107  EXAMPLE 19                                *

LN(2) =  69314765       EPS =  10000000-03
   PI =  31415925  1    EPS =  10000000-03
   PI =  31415927  1    EPS =  10000000-05
```

FIGURE 39.

EXAMPLE 20.

This example illustrates how the ENTER statement is used to obtain a numerical solution of a system of first-order, ordinary differential equations,

$$\frac{dy_i}{dt} = y_i' = f_i(t,y_1(t),\ y_2(t),\ldots,y_n(t))$$

with initial conditions

$$y_i(t_o) = y_{io} \quad , \qquad i = 0,1,2,\ldots,\mathbf{N}\ .$$

The Runge-Kutta-Gill method is applied to obtain an approximate solution. The statement has the general form:

ENTER(R.K.G)Δt)n)EVDS)Y)Y')Q

where:

R.K.G. is an abbreviation for Runge-Kutta Gill;

Δt is the name of the incremental value of the independent variable; The numerical solution is determined at discrete values of the independent variable, t. i.e., when initial conditions are specified corresponding to $t = t_o$, the solution, $y_i(t)$, is determined at $t = t_o + \Delta t$. The computer replaces the given values, $y_i(t_o)$, with the new values, $y_i(t_o + \Delta t)$. These new values then serve as the given values for determining the solution at the "next" discrete t, namely at $t_o + 2\Delta t$. Each subsequent entry produces the solution at the next t;

n is an integer (or name of an integer) which specifies the number of equations in the system, i.e., $n = N + 1$;

The method requires a "zeroth" equation

$$y'_o = \frac{dt}{dt} = 1$$

which is treated in the same manner as the other equations.

This provides for adjusting the value of the independent variable as required by the method.

EVDS is the name of the location where the derivatives are explicitly defined;

Similar to the Simpson integration of the previous example, this statement is designed for arbitrary f_i, hence, again it is the responsibility of the programmer to write the statements which evaluate the f_i of interest. These statements must be designed to record the evaluated derivatives in consecutive places called Y'. Further, this sequence of statements must be terminated with GOTO(R.K.GD). Specifying the name EVDS in the ENTER statement and terminating this evaluation of the derivatives with GOTO(R.K.GD) enables the computer to evaluate and weight the derivatives as many times as required by the method.

Y is the name of the first of the N+1 functional values (or equivalently, the name of the "zeroth" functional value);

To avoid confusion in (n) and (N+1) and the inclusion of the independent variable as one of the (N+1) functional values and derivatives it is suggested that the system be considered and identified as follows:

$$\frac{dy}{dt} = y' = 1$$

$$\frac{dy_1}{dt} = y_1' = f_1$$

$$\frac{dy_2}{dt} = y_2' = f_2$$

$$\vdots$$

$$\frac{dy_n}{dt} = y_n' = f_n$$

To identify and reserve space for the required quantities it is suggested that the programmer use the following or equivalent BLOC statement

BLOC(Y - Y n)Y' - Y' n)Q - Q n)

As stated above, Y is the name of the first of the (N+1) consecutive functional values. These correspond to the names of the given conditions and correspondingly to the names where the resulting values will be recorded.

Y' is the name of the first of the (N+1) consecutive derivative values; It is in these places that the EVDS statements instruct the computer to record the derivative values. As identified above, $y' = 1$ corresponds to $\frac{dt}{dt} = 1$.

Q is the name of the first of (N+1) quantities that are required for intermediate values in the computations.

For subsequent steps, i.e., for solutions at $t_o + k\Delta t$ where $k = 1,2,3,\ldots,$ etc., it is not necessary to re-specify the parameters as in the original ENTER statement. Indeed, for subsequent steps one need only write GOTO(R.K.G1).

GIVEN:

$$\frac{dy}{dt} = y' = 1 \qquad \text{where} \quad y = t, \text{ and at} \quad t = t_o = 0$$

$$\frac{dy_1}{dt} = y_1' = (a/b)y_2(t) \qquad y(t_o) = 0$$

$$\frac{dy_2}{dt} = y_2' = (-b/a)y_1(t) \qquad y_1(t_o) = 0$$

$$\frac{dy_3}{dt} = y_3' = (d)y_3(t) \qquad y_2(t_o) = b$$

$$y_3(t_o) = c$$

where a, b, c, and d are constants

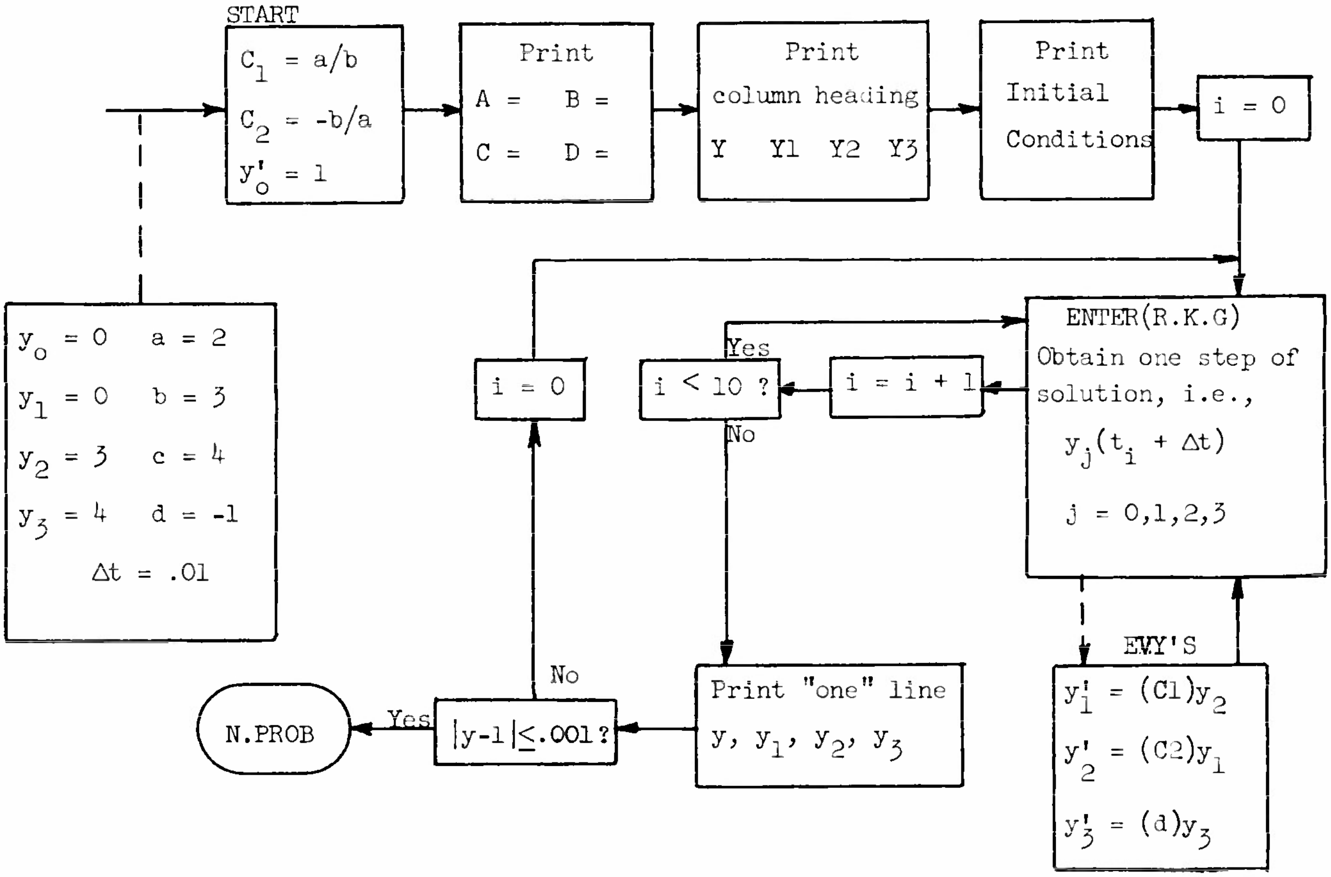
REQUIRED: For $a = 2$, $b = 3$, $c = 4$, $d = -1$,

Determine, print (and identify)

y, $y_1(t)$, $y_2(t)$, $y_3(t)$ for $t = 0, .1, .2, \ldots, 1.0$.

Use a computation step-size $\Delta t = .01$

The flow chart for this example is given in FIGURE 40 below; the program and output in FIGURE 4.



FLOW CHART FOR EXAMPLE 20.

FIGURE 40.

EXAMPLE 21.

This example is given to illustrate addition, subtraction, multiplication and inversion of matrices. We use ENTER statements for multiplication and inversion whereas we use general formulas and COUNT statements for addition and subtraction. The illustrations given in this example do not include permissible options such as: references to symmetric matrices; references to the transpose of a matrix; references to internally stored matrices whose elements are uniformly spaced different from unity; or "accummulative multiplication", i.e., adding the matrix C to the product matrix AB and recording the result at C. For ease in reference, it is convenient to identify matrices as two dimensional arrays. This is readily achieved through the use of BLOC statements.

GIVEN:

A, B and C, where:

A is a (2 x 2) matrix;
B is a (2 x 2) matrix;
C is a (2 x 3) matrix.

REQUIRED:

Print and identify the following matrices:

1. A
2. B
3. C
4. A^{-1}
5. $(A^{-1})^{-1}$
6. D = B x C
7. E = D + C
8. $H = A - (A^{-1})^{-1}$
9. $I = A^{-1}A$

The flow chart for this example is given in FIGURE 42.

The program, input and output are listed in FIGURE 43.

```
MAY.23,63  BRLESC  FORAST F62
     PROB C906 M.J.ROMANELLI  45107 EXAMPLE 21                                  *

DETA = -30000000  1   DETA' = -33333333                                  0000001

    A MATRIX              B  MATRIX                  C MATRIX            0000002
  1.0000   2.0000      3.0000   4.0000     7.0000   8.0000   9.0000
  2.0000   1.0000      5.0000   6.0000     3.0000   2.0000   1.0000
    A INVERSE            A  INV.  INV.               D = B*C             0000003
-  .3333    .6667      1.0000   2.0000    33.0000  32.0000  31.0000
   .6667 -  .3333      2.0000   1.0000    53.0000  52.0000  51.0000
    IDENTITY                NULL                     E = D+C             0000004
  1.0000 -  .0000       .0000 -  .0000    40.0000  40.0000  40.0000
   .0000   1.0000   -   .0000 -  .0000    56.0000  54.0000  52.0000
```

FIGURE 43.

To form a sum of matrices, (or difference), we write a general equation using an index, then by means of an initial setting of the index and an appropriate COUNT statement, the sum or difference is obtained. As illustrated on the flow chart and the program, for the sum, E = D + C, we initially set the index J = 0. The general equation is then expressed as:

```
SUMMAT        E1,1,J = D1,1,J + C1,1,J  %
```

So that J takes on the required integer values, we write

```
COUNT(6)IN(J)GOTO(SUMMAT)
```

Note that J was set equal to zero on line 4 and hence the first evaluation of the general definition corresponds to summing the first elements of the matrices. Note also that we set M = 2 and N = 3 on line 4. This was done only to illustrate later references to dimensions by name rather than by explicit integers.

Next, to obtain the inverse we could have written the appropriate ENTER statement immediately following line 5. However, as in the case of solving linear equations, the given matrix is replaced with the resulting inverse. Hence, since we want the original A matrix for future operations, we move A to F and retain the original at A. We will instruct the computer to invert F and hence after the inversion is completed, A^{-1} will be recorded at F. To obtain the inverse of F, we write

```
ENTER(MAT.INV)F1,1)2)DETA
```

where:

MAT.INV is an abbreviation for matrix inversion;

F1,1 is the name of the first element of the matrix to be inverted, (successive element in consecutive spaces by rows);

2 indicates the dimension, i.e., F is a 2 x 2 matrix;

DETA is optional and when specified as above is the name of the value of the determinant of the matrix to be inverted.

EXAMPLE 22.

In the next examples, several methods for determining approximations to real roots of continuous functions are illustrated. This example illustrates a "bisection" method.

GIVEN:

$f(X) = X^3 - X - 1$;

X_o and X_1 such that $f(X_o)f(X_1) < 0$;

$\epsilon > 0$

REQUIRED:

Determine $X_2, X_3, X_4, . . , X_j$, such that $|f(X_j)| < \epsilon$;

Print and identify, X_i and $f(X_i)$ for $i = 0,1,2,...,j$.

In the bisection method, the "next" approximation for the root is defined as

$$X = \frac{X_n + X_p}{2}$$

where $f(X_n) < 0$ and $f(X_p) > 0$. The function is evaluated for this mean X and a test is applied to determine if the magnitude of the function is less than the given ϵ. If the magnitude is less than ϵ, X is the desired root. If the magnitude is not less than ϵ, then the interval in which the root lies is diminished by replacing X_n or X_p by X. We replace X_n by X if $f(X) < 0$, or X_p by X if $f(X) > 0$. The next mean X is determined and the process continued. The flow chart for this example is given in FIGURE 44.

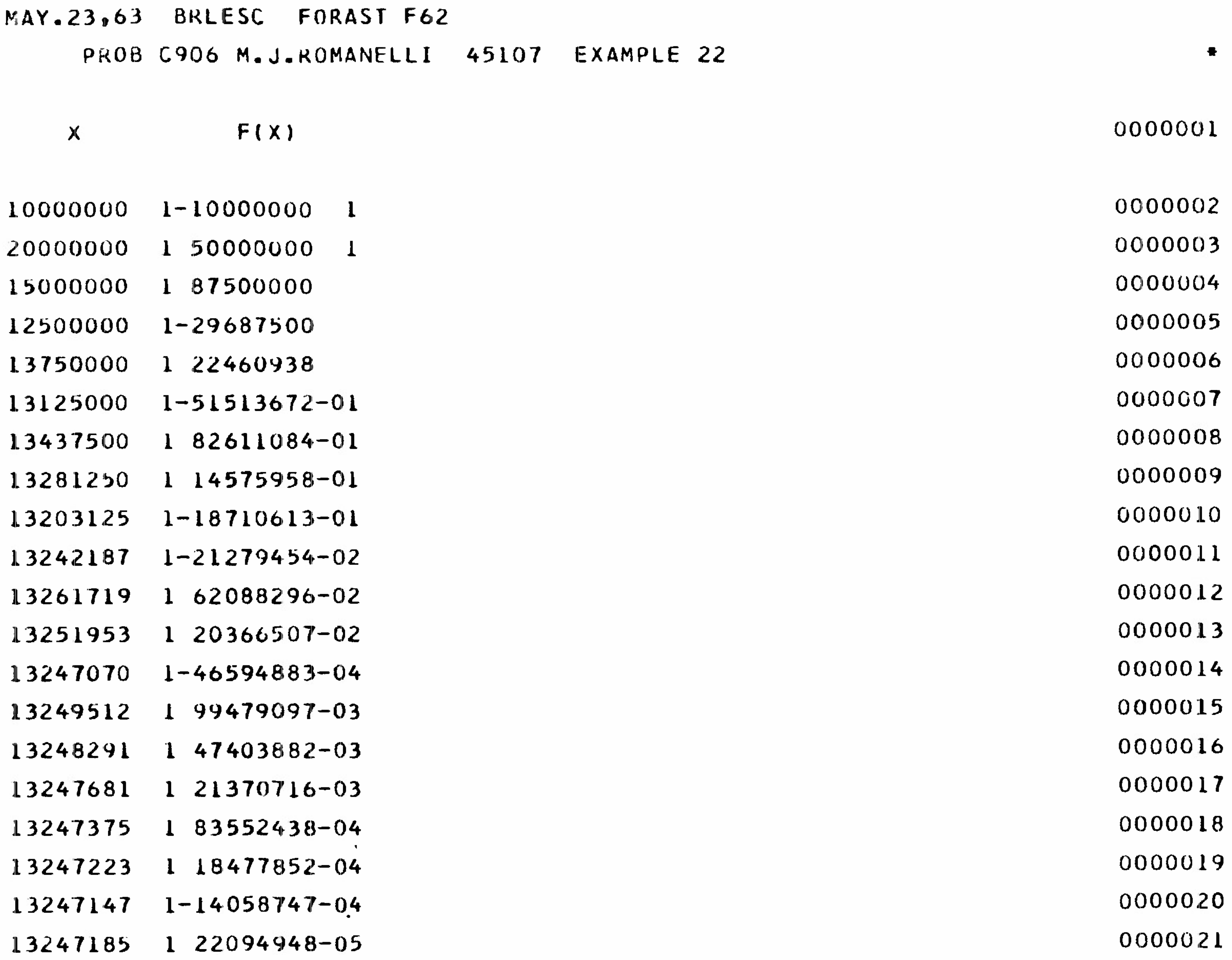
Since the process will require an indefinite number of evaluations of the function, we wrote the general definition for arbitrary X and terminated the general definition and printing of X and f(X) with a "variable" exit, E. Prior to each entrance to the evaluation, we establish values for the arbitrary X and the variable exit, E.

The function is first evaluated for $X = X_o$ with exit E = 1. At 1. we record $f_o = f(X_o)$ and establish either X_p or X_n depending on whether $f(X_o)$ was positive or negative.

The function is next evaluated for $X = X_1$ with exit E = 4. At 4. we record $f_1 = f(X_1)$ and establish either X_p or X_n, again depending on whether $f(X_1)$ was positive or negative.

At 6. the exit E is set equal to 7. for all future exits from the general function evaluation. Next we apply a test to determine if the initial condition is satisfied; i.e., if $f(X_o)f(X_1) < 0$. If the condition is not satisfied, this indication is printed and the computer is directed to the next problem. If the initial condition is satisfied, the initial X_n and X_p required for the general process are established and recorded. Hence, we direct the computer to the definition of the mean X and initiate the general process.

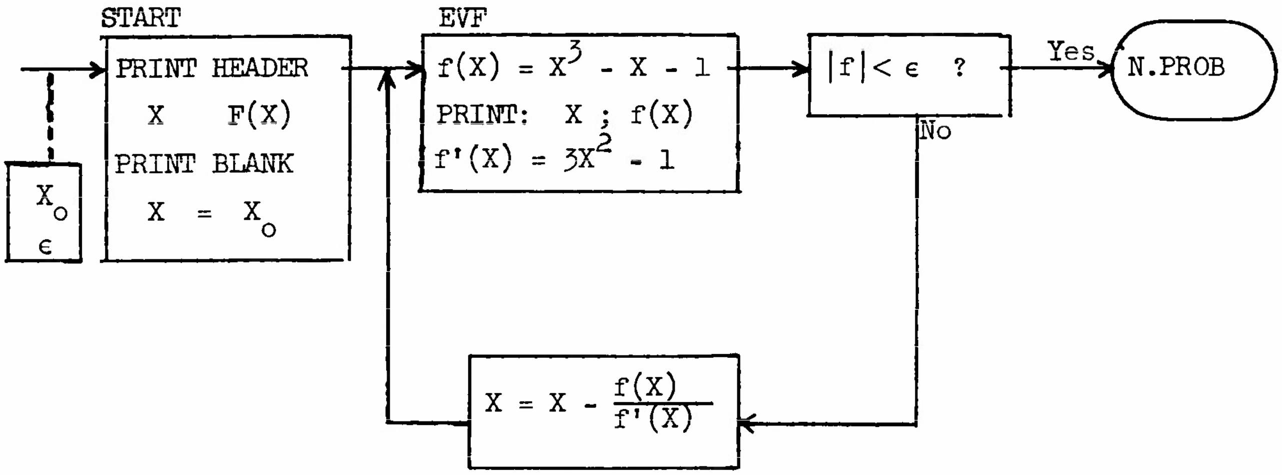
The program and results for this example are listed in FIGURE 45. Note that this program may be modified for other f(X) and corresponding initial conditions by replacing cards 3 and 15 accordingly.



```
MAY.23,63  BRLESC  FORAST F62
     PROB C906 M.J.ROMANELLI  45107  EXAMPLE 22                                        *

    X          F(X)                                                               0000001

10000000  1-10000000   1                                                          0000002
20000000  1 50000000   1                                                          0000003
15000000  1 87500000                                                              0000004
12500000  1-29687500                                                              0000005
13750000  1 22460938                                                              0000006
13125000  1-51513672-01                                                           0000007
13437500  1 82611084-01                                                           0000008
13281250  1 14575958-01                                                           0000009
13203125  1-18710613-01                                                           0000010
13242187  1-21279454-02                                                           0000011
13261719  1 62088296-02                                                           0000012
13251953  1 20366507-02                                                           0000013
13247070  1-46594883-04                                                           0000014
13249512  1 99479097-03                                                           0000015
13248291  1 47403882-03                                                           0000016
13247681  1 21370716-03                                                           0000017
13247375  1 83552438-04                                                           0000018
13247223  1 18477852-04                                                           0000019
13247147  1-14058747-04                                                           0000020
13247185  1 22094948-05                                                           0000021
```

FIGURE 45.

The flow chart for this example is given in FIGURE 46.



FLOW CHART FOR EXAMPLE 23.

FIGURE 46.

The program and results are listed in FIGURE 47. Note that this program may be modified for other f(X) by replacing cards 3 and 5 accordingly. Note also, this process may not converge, indeed, it may diverge if f'(X) tends to zero.

EXAMPLE 24.

This example illustrates the "Regula-Falsi" method for obtaining an approximation for a real root of a function, $f(X) = 0$. In this method, the next approximation for the root is defined by:

a.) X_n, such that $f(X_n) < 0$;

b.) X_p, such that $f(X_p) > 0$;

c.) $f(X_n)$

d.) $f(X_p)$

In general, the new approximation,

$$X = \frac{X_n f(X_p) - X_p f(X_n)}{f(X_p) - f(X_n)}$$

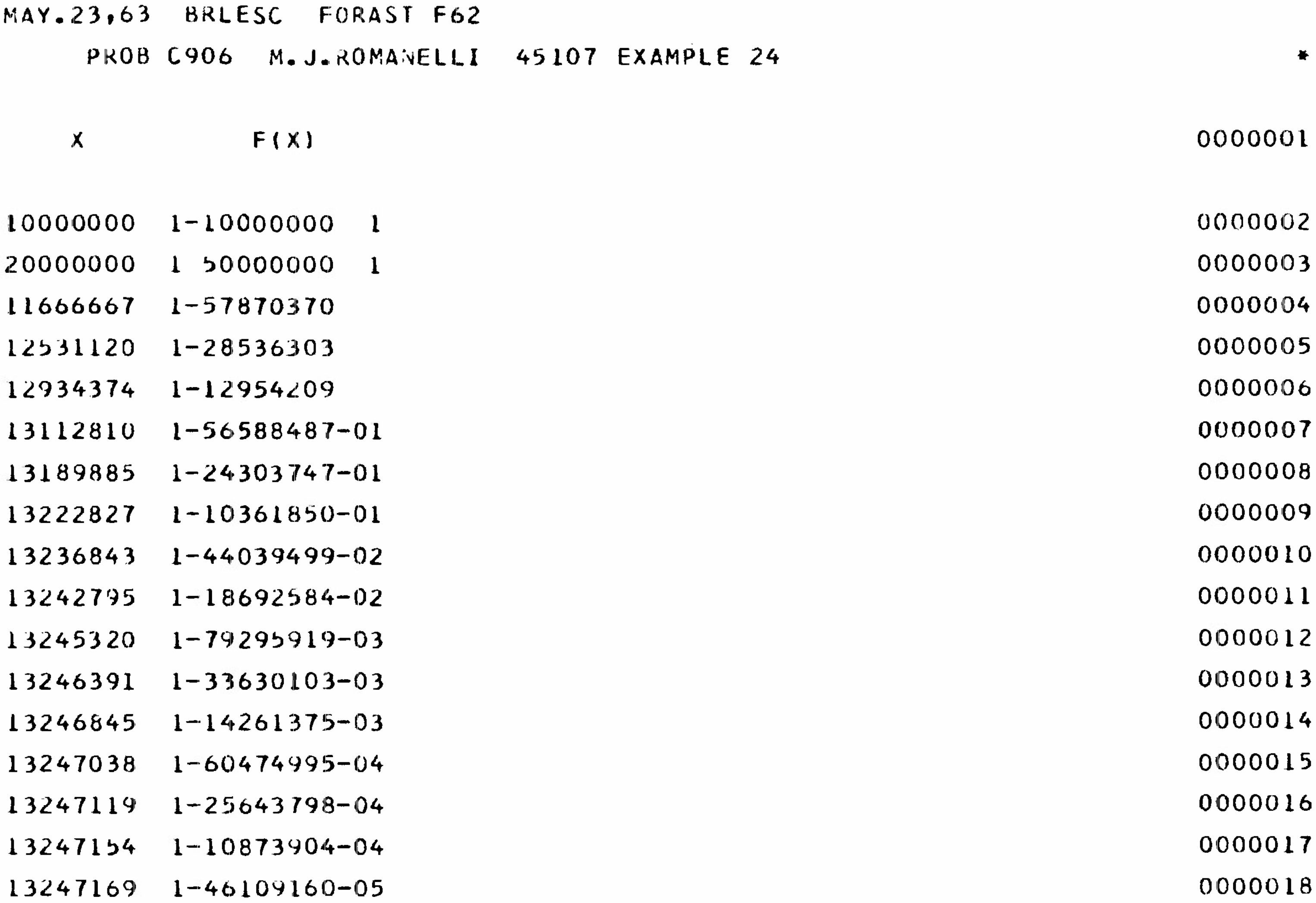
GIVEN:

X_o and X_1 such that $f(X_o)\, f(X_1) < 0$, $f(X) = X^3 - X - 1$, and $\epsilon > 0$

REQUIRED:

Determine X such that $f(X) < \epsilon$. Print and identify each X and corresponding $f(X)$

The flow chart for this example is given in FIGURE 48. Note the similarity to the bisection method flow-chart. The program and results are listed in FIGURE 49.



```
MAY.23,63  BRLESC  FORAST F62
     PROB C906  M.J.ROMANELLI  45107 EXAMPLE 24                                  *

    X          F(X)                                                      0000001

10000000  1-10000000   1                                                 0000002
20000000  1 50000000   1                                                 0000003
11666667  1-57870370                                                     0000004
12531120  1-28536303                                                     0000005
12934374  1-12954209                                                     0000006
13112810  1-56588487-01                                                  0000007
13189885  1-24303747-01                                                  0000008
13222827  1-10361850-01                                                  0000009
13236843  1-44039499-02                                                  0000010
13242795  1-18692584-02                                                  0000011
13245320  1-79295919-03                                                  0000012
13246391  1-33630103-03                                                  0000013
13246845  1-14261375-03                                                  0000014
13247038  1-60474995-04                                                  0000015
13247119  1-25643798-04                                                  0000016
13247154  1-10873904-04                                                  0000017
13247169  1-46109160-05                                                  0000018
```

FIGURE 49.

EXAMPLE 25.

This example illustrates a "constant secant" method for approximating a real root of a function, $f(X) = 0$. Again we assume that X_o and X_1 are given such that

$$f(X_o)\ f(X_1) < 0.$$

The slope

$$m = \frac{f(X_1) - f(X_o)}{X_1 - X_o}$$

is computed and the "next" approximation to the root is defined similar to the Newton-Raphson method, i.e.,

$$X_{i+1} = X_i - \frac{f(X_i)}{m}$$

GIVEN:

$f(X) = X^3 - X - 1$

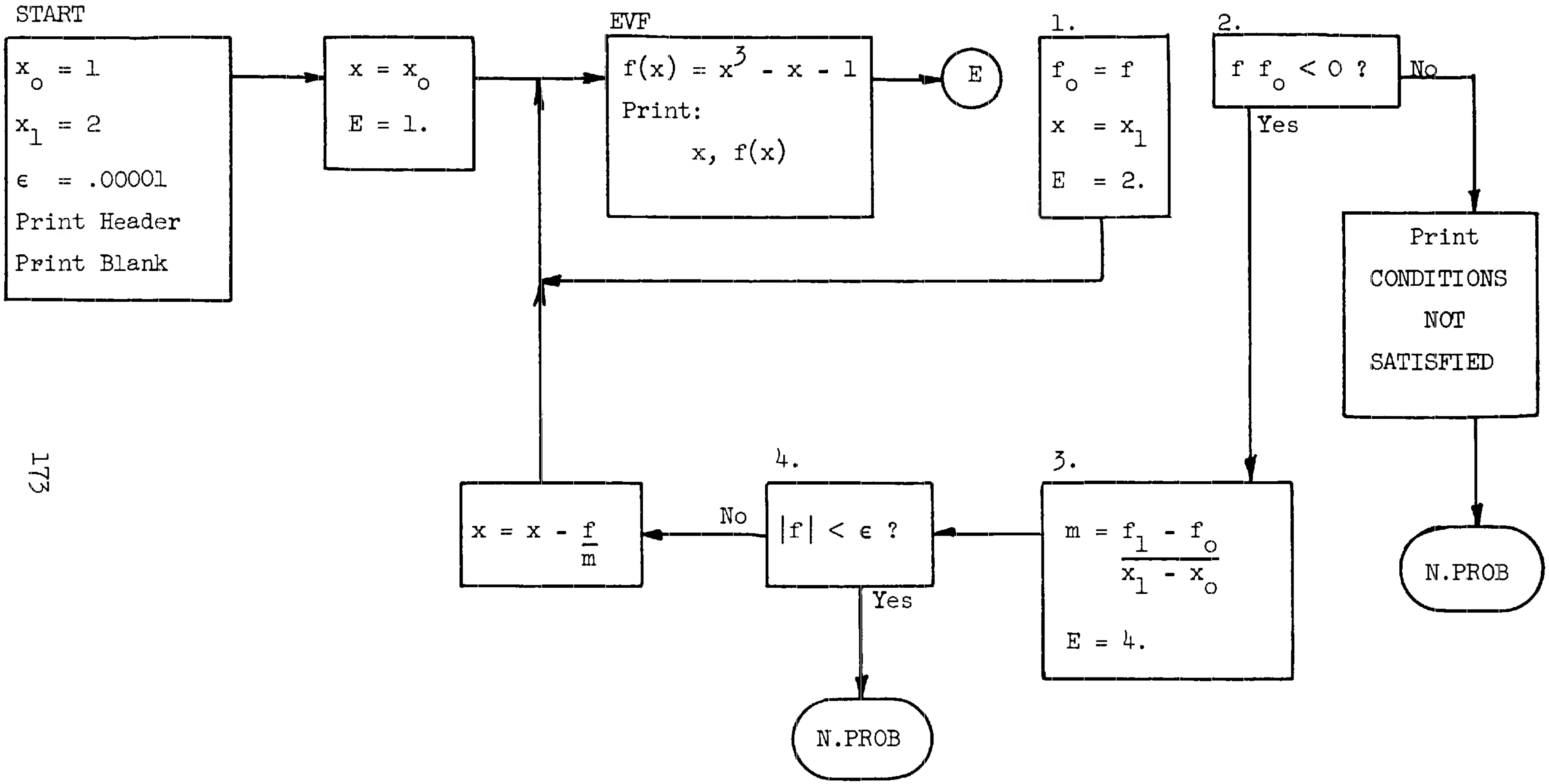
X_o and X_1 such that $f(X_o)\ f(X_1) < 0$; $\epsilon > 0$.

REQUIRED:

Determine $X_2, X_3, X_4, \ldots, X_j$, such that $|f(X_j)| < \epsilon$;

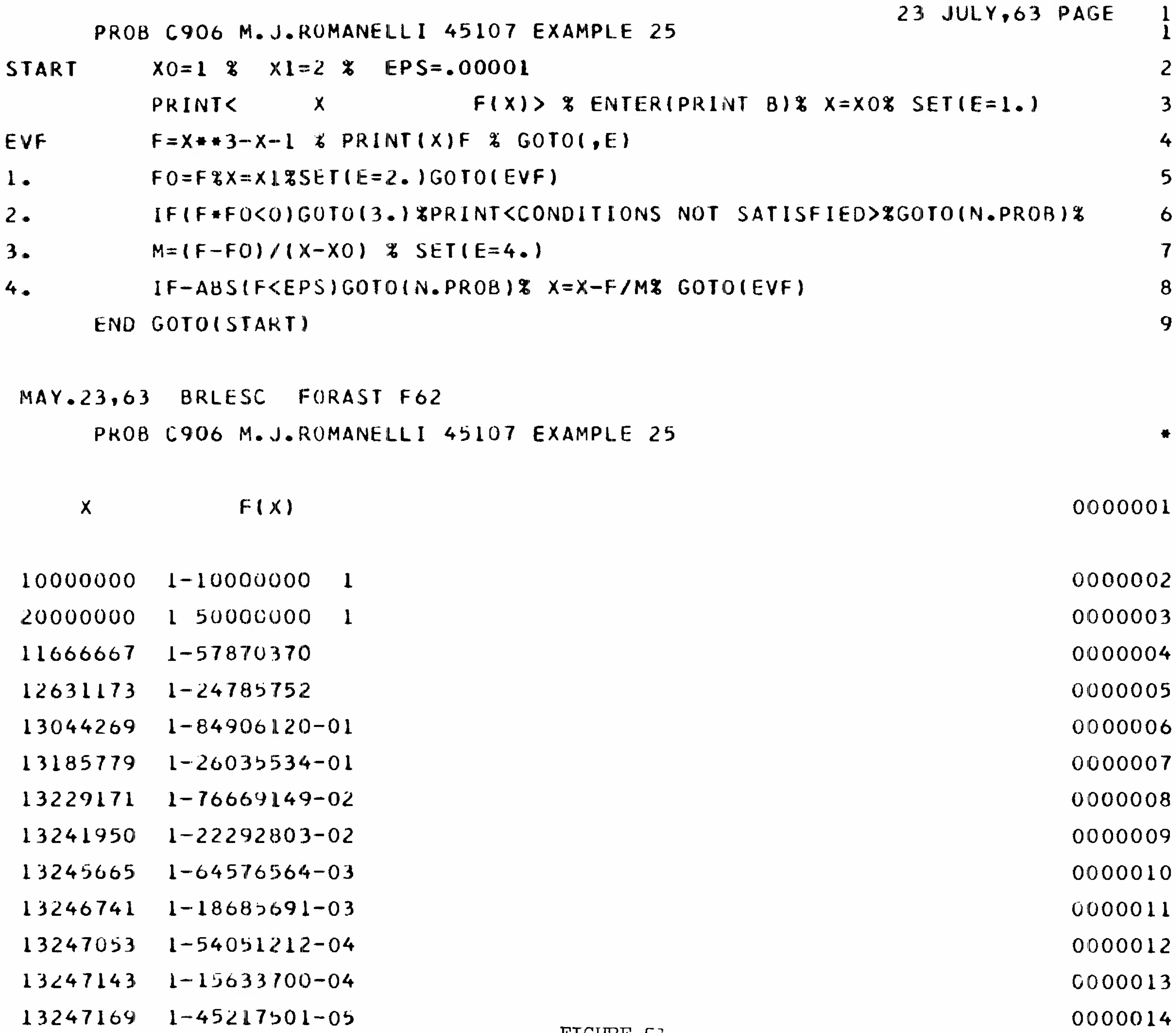
Print and identify X_i and $f(X_i)$, $i < 0,1,2,\ldots,j$.

The flow-chart for this method is shown in FIGURE 50. The program and output obtained are listed in FIGURE 51.



FLOW CHART FOR EXAMPLE 25.

FIGURE 50.



```
                                                               23 JULY,63 PAGE  1
        PROB C906 M.J.ROMANELLI 45107 EXAMPLE 25
START      X0=1 %  X1=2 %  EPS=.00001
           PRINT<     X          F(X)> % ENTER(PRINT B)% X=X0% SET(E=1.)
EVF        F=X**3-X-1 % PRINT(X)F % GOTO(,E)
1.         F0=F%X=X1%SET(E=2.)GOTO(EVF)
2.         IF(F*F0<0)GOTO(3.)%PRINT<CONDITIONS NOT SATISFIED>%GOTO(N.PROB)%
3.         M=(F-F0)/(X-X0) % SET(E=4.)
4.         IF-ABS(F<EPS)GOTO(N.PROB)% X=X-F/M% GOTO(EVF)
        END GOTO(START)


 MAY.23,63  BRLESC  FORAST F62
        PROB C906 M.J.ROMANELLI 45107 EXAMPLE 25                                  *


      X          F(X)                                                      0000001


 10000000  1-10000000  1                                                   0000002
 20000000  1 50000000  1                                                   0000003
 11666667  1-57870370                                                      0000004
 12631173  1-24785752                                                      0000005
 13044269  1-84906120-01                                                   0000006
 13185779  1-26035534-01                                                   0000007
 13229171  1-76669149-02                                                   0000008
 13241950  1-22292803-02                                                   0000009
 13245665  1-64576564-03                                                   0000010
 13246741  1-18685691-03                                                   0000011
 13247053  1-54051212-04                                                   0000012
 13247143  1-15633700-04                                                   0000013
 13247169  1-45217501-05                                                   0000014
```

FIGURE 51.

EXAMPLE 26.

This example illustrates another iterative method for determining an approximation to a real root of a function $f(X) = 0$. Here it is assumed that $f(X) = 0$ can be expressed in an equivalent form

$$X = F(X)$$

The general iteration is then expressed in the form

$$X_{i+1} = F(X_i)$$

where we assume an initial $X = X_o$ is given to start the iteration.

GIVEN:

$f(X) = X^3 - X - 1$;

X_o ;

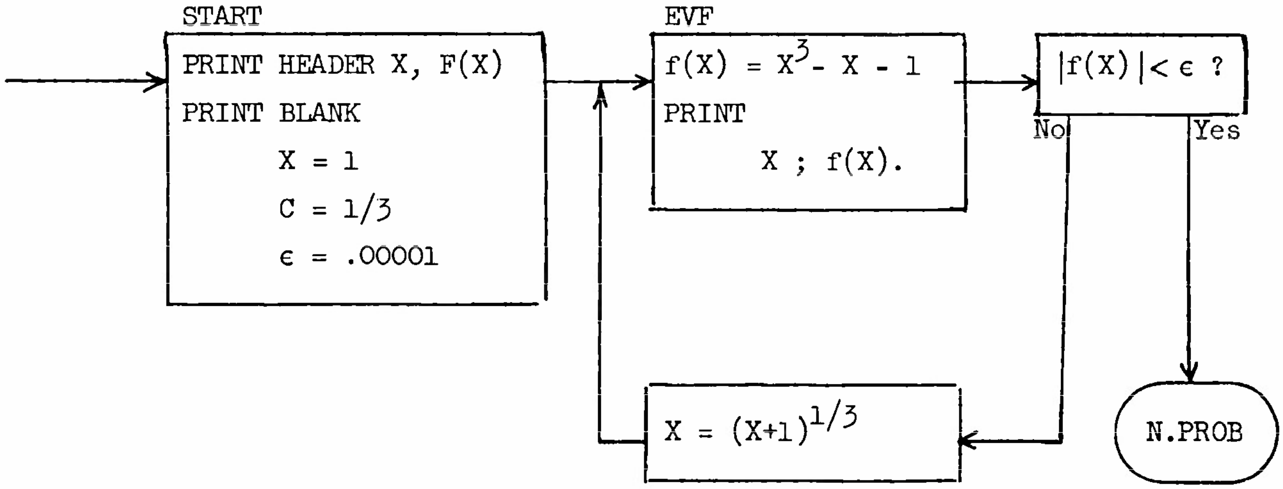
$\epsilon > 0$.

REQUIRED:

Use the iteration, $X_{i+1} = (X_i + 1)^{1/3}$ to determine X_j, such that $|f(X_j)| < \epsilon$.

Print and identify X_i and $f(X_i)$, $i = 0,1,2,\ldots,j$.

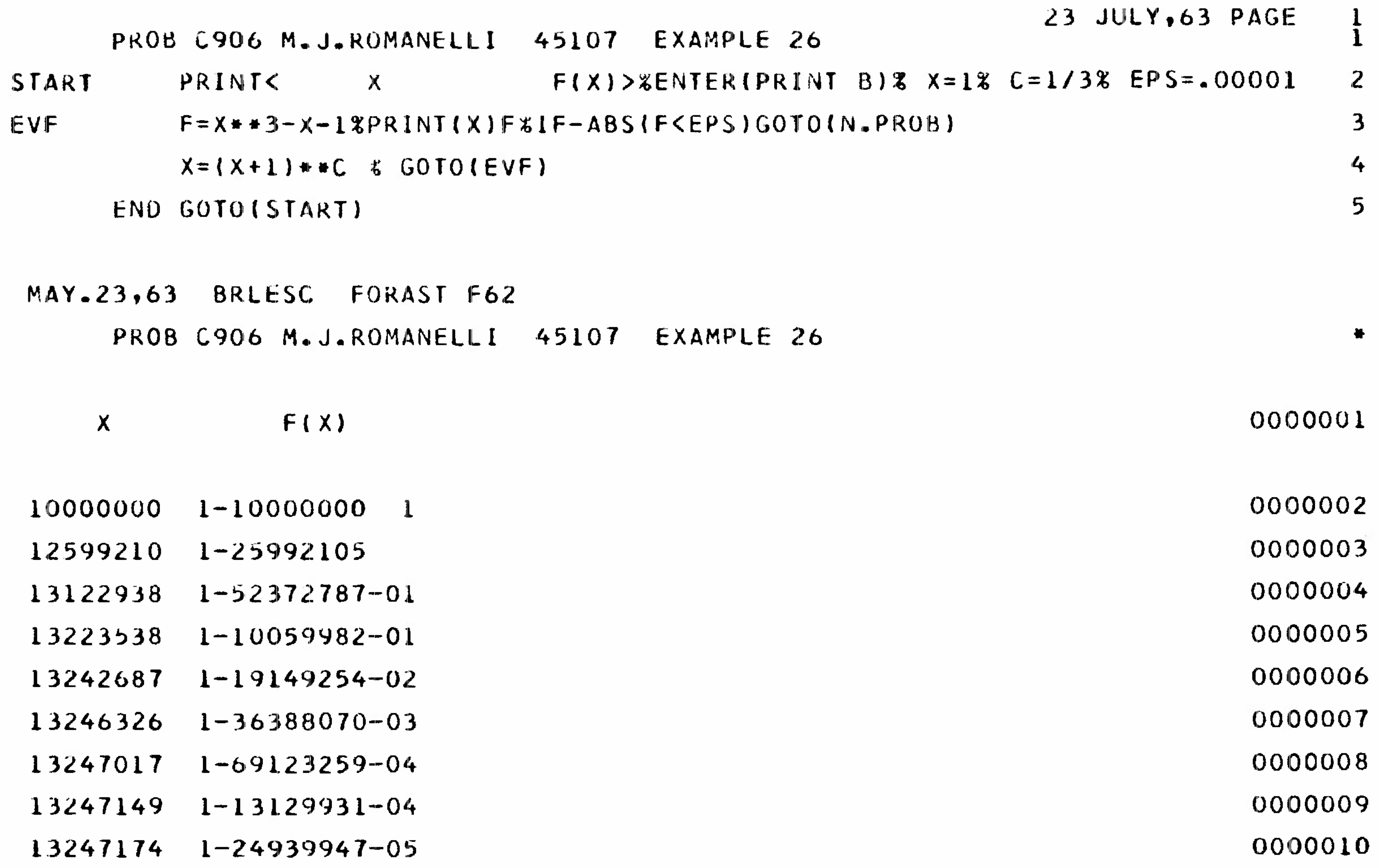
The flow-chart for this example is given in FIGURE 52.



FLOW CHART FOR EXAMPLE 26

FIGURE 52.

The program and output obtained are listed in FIGURE 53.



```
                                                                   23 JULY,63 PAGE    1
          PROB C906 M.J.ROMANELLI   45107   EXAMPLE 26                                1
START          PRINT<      X            F(X)>%ENTER(PRINT B)% X=1% C=1/3% EPS=.00001  2
EVF            F=X**3-X-1%PRINT(X)F%IF-ABS(F<EPS)GOTO(N.PROB)                         3
               X=(X+1)**C % GOTO(EVF)                                                 4
          END GOTO(START)                                                             5

 MAY.23,63   BRLESC   FORAST F62
          PROB C906 M.J.ROMANELLI   45107   EXAMPLE 26                                *


    X            F(X)                                                           0000001


 10000000   1-10000000   1                                                      0000002
 12599210   1-25992105                                                          0000003
 13122938   1-52372787-01                                                       0000004
 13223538   1-10059982-01                                                       0000005
 13242687   1-19149254-02                                                       0000006
 13246326   1-36388070-03                                                       0000007
 13247017   1-69123259-04                                                       0000008
 13247149   1-13129931-04                                                       0000009
 13247174   1-24939947-05                                                       0000010
```

FIGURE 53.

EXAMPLE 27.

This example illustrates a "least-squares" method for "fitting" a polynomial to given tabular data. Although there are "packaged" programs available for this purpose, this and the next two examples are given to illustrate how the ENTER statement is used to generate the normal equations required for least-squares solutions. To illustrate we consider the following problem:

GIVEN:

X_i, $y(X_i)$ for $i = 1,2,3, \ldots\ldots$

Assume this data on cards in standard form, one pair per card.

Since an indefinite number of pairs are given, we will terminate this data with two blank cards.

REQUIRED:

Determine, print and identify, C_1, C_2, and C_3 such that

$$S = \sum_i (\bar{y}(X_i) - y(X_i))^2 \text{ is a minimum,}$$

where

$$\bar{y}(X_i) = C_1 + C_2X_i + C_3X_i^{\,2} .$$

The necessary conditions for a minimum are:

$$\frac{\partial S}{\partial C_1} = 0 = \sum_i X_i^0 C_1 + \sum_i X_i^1 C_2 + \sum_i X_i^2 C_3 - \sum_i y(X_i)$$

$$\frac{\partial S}{\partial C_2} = 0 = \sum_i X_i C_i + \sum_i X_i^2 C_2 + \sum_i X_i^3 C_3 - \sum_i X_i y(X_i)$$

$$\frac{\partial S}{\partial C_3} = 0 = \sum_i X_i^2 C_1 + \sum_i X_i^3 C_2 + \sum_i X_i^4 C_3 - \sum_i X_i^2 y(X_i)$$

To obtain the solution, we must generate and solve the above system of linear equations in the three unknowns, C_1, C_2, and C_3. Note that the matrix of coefficients of the unknowns is symmetric; i.e.,

$$A = \begin{pmatrix} \sum_i X_i^0 & \sum X_i^1 & \sum X_i^2 \\ \sum X_i & \sum X_i^2 & \sum X_i^3 \\ \sum X_i^2 & \sum X_i^3 & \sum X_i^4 \end{pmatrix}$$

Note also that the system to be generated and solved is of the form illustrated in EXAMPLE 18, i.e., of the form

$$AC = b$$

where:

A is symmetric

C is the unknown vector, C_1, C_2, C_3 ;

and

$$b = \begin{pmatrix} \sum\limits_{i} y_i \\ \\ \sum\limits_{i} x_i y_i \\ \\ \sum\limits_{i} x_i^2 y_i \end{pmatrix}$$

Hence, the following elements must be generated and recorded internally in consecutive spaces by rows.

$\sum\limits_{i} x_i^0$	$\sum\limits_{i} x_i^1$	$\sum\limits_{i} x_i^2$	$\sum\limits_{i} y_i$
	$\sum\limits_{i} x_i^2$	$\sum\limits_{i} x_i^3$	$\sum\limits_{i} x_i y_i$
		$\sum x_i^4$	$\sum x_i^2 y_i$

To generate the above, the FORAST language includes a convenient ENTER statement of the form

```
ENTER(F.N.E.)A1,1)n)P1)W %
```

where:

F.N.E. is an abbreviation for, "Form Normal Equations" ;

A1,1 is the name of the firstelement in the first row of the symmetric system to be generated;
(space should be reserved for this system with a BLOC statement)

n is an integer (or name of an integer) which specifies the number of equations in the system to be generated;
(for our example, n = 3)

P_1 is the name of the first of (n+1) consecutive elements, in particular,

General form,

$$P_1 = \frac{\partial \bar{y}}{\partial C_1}\bigg| (X_i, y_i)$$

$$P_2 = \frac{\partial \bar{y}}{\partial C_2}\bigg| (X_i, y_i)$$

.

.

.

$$P_n = \frac{\partial \bar{y}}{\partial C_n}\bigg| (X_i, y_i)$$

$$P_{n+1} = y_i$$

For our example,

$$P_1 = 1$$

$$P_2 = X_i$$

$$P_3 = X_i^2$$

$$P_4 = y_i$$

where the notation, $\Big|_{(X_i,y_i)}$, denotes evaluation of the partial derivative at (X_i,y_i) ; (again, space should be reserved for these (n+1) quantities with a BLOC statement).

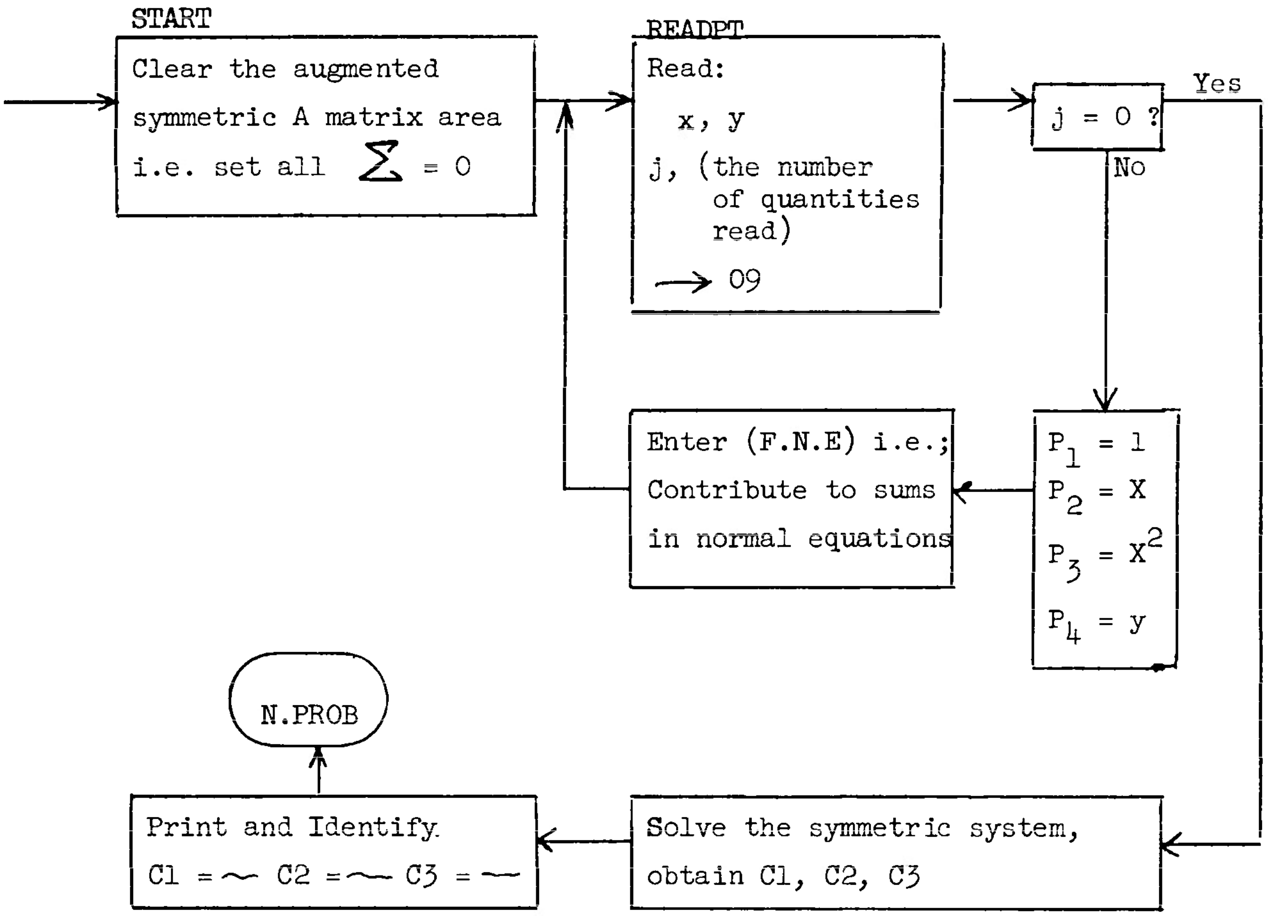
W is optional and if specified is the name of a weight value to be applied in the sums, if unspecified, the weight value is taken as unity.

Corresponding to a given point, $(X_i,y(X_i)$, we instruct the computer to compute and record the (n+1) quantities at P_1, P_2, ... , P_{n+1}. Then, the above ENTER statement tells the computer to make the necessary contributions to the sums in the normal equations. When all points have been processed in this manner, we instruct the computer to solve the symmetric system generated through the use of the ENTER statement

```
ENTER(SY.SNE)A1,1)3)C1 %
```

The solution is then available for printing or for any subsequent use desired.

The flow-chart for this example is given in FIGURE 54.


START
Clear the augmented symmetric A matrix area i.e. set all $\sum = 0$
READPT
Read:
x, y
j, (the number of quantities read)
→ 09
j = 0 ?
Yes
No
$P_1 = 1$
$P_2 = X$
$P_3 = X^2$
$P_4 = y$
Enter (F.N.E) i.e.; Contribute to sums in normal equations
N.PROB
Print and Identify.
C1 = ~ C2 = ~ C3 = ~
Solve the symmetric system, obtain C1, C2, C3

FLOW CHART FOR EXAMPLE 27.

FIGURE 54.

The program, input data and results obtained for this example are listed in FIGURE 55.

```
                                                            23 JULY,63 PAGE    1
          PROB C906 M.J.ROMANELLI   45107   EXAMPLE 27
          BLOC(A1,1-A3,4/SY.)P1-P4)C1-C3)
START        CLEAR(9)NOS.AT(A1,1)%
READPT       READ(X)Y% IF-INT(09=0)GOTO(SOL'N)
             P1=1% P2=X% P3=X*X% P4=Y%
             ENTER(F.N.E.)A1,1)3)P1% GOTO(READPT)
SOL'N        ENTER(SY.SNE)A1,1)3)C1%
             PRINT<C1 = >C1<  C2 = >C2<  C3 = >C3% GOTO(N.PROB)
          END GOTO(START)
 0.0           3.0000
  .1           2.9850
  .2           2.9402
  .3           2.8660
  .4           2.7632
  .5           2.6327
  .6           2.4760
  .7           2.2945
  .8           2.0901
  .9           1.8648
 1.0           1.6209


 MAY.23,63   BRLESC   FORAST F62
          PROB C906 M.J.ROMANELLI   45107   EXAMPLE 27                              *

C1 =   30077021   1   C2 = -10693520      C3 = -12891375   1                 0000001
```

FIGURE 55.

EXAMPLE 28.

In the previous example we obtained a "polynomial" fit to some tabular data. In this example we will include a computation and printing of the "residuals"; i.e., the difference between the given functional value and the approximating functional value at each point. We will also compute and print the "root mean-square" error.

First, since we need the given functional values to compute residuals, we will have to make these functional values available after the solution for the approximating function is obtained. (Recall that we discarded the data points as soon as their contribution to the normal equations was completed). Hence, to make the given data available for residual computations, we will first read and record all of the given data and retain it internally for as many future references as desired.

To provide for a maximum of say 500 points, we will reserve space for 1000 values of given data, hence we will use a BLOC statement

```
BLOC(D1 - D1000).
```

Further, since only two data values are recorded per card, we will specify a format which departs from standard and indicates the desired departure.

The READ statement to accomplish the reading and recording of a maximum of 1000 values, two per card has the form

```
READ-FORMAT(F/2)-(1000)NOS.AT(D1)
```

where the format F which specifies the particular form desired is:

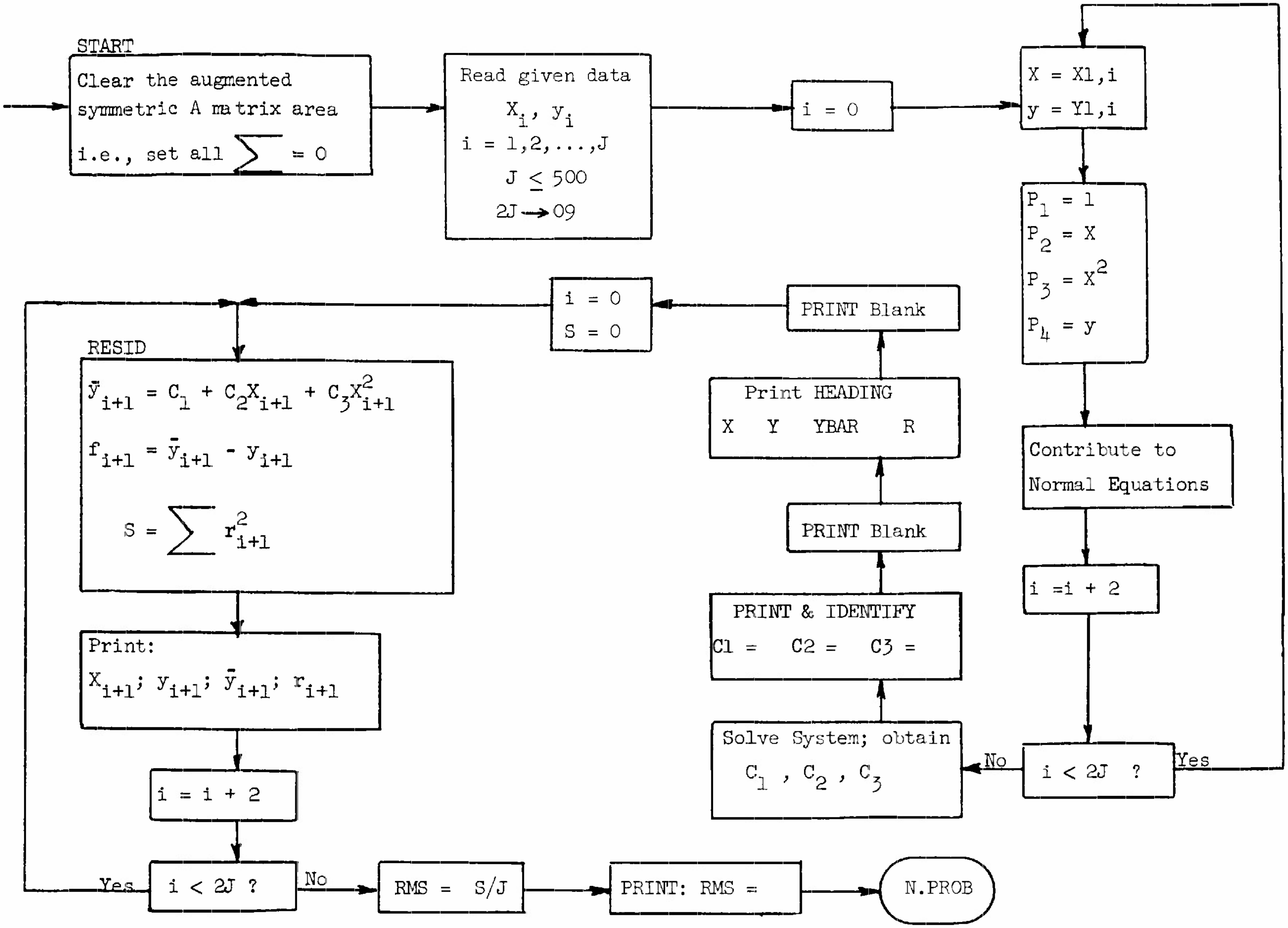
```
F    FORM    (10-12)10-12) 2  %
```

To identify and refer to the given X's and Y's we can write a SYN statement of the form

```
SYN(X1 = D1)Y1 = D2) %
```

Hence, to refer to the given data we may refer to the X's, Y's or D's bearing in mind that the X's are <u>two</u> units apart in internal stcrage, and likewise, the Y's are two units apart in internal storage.

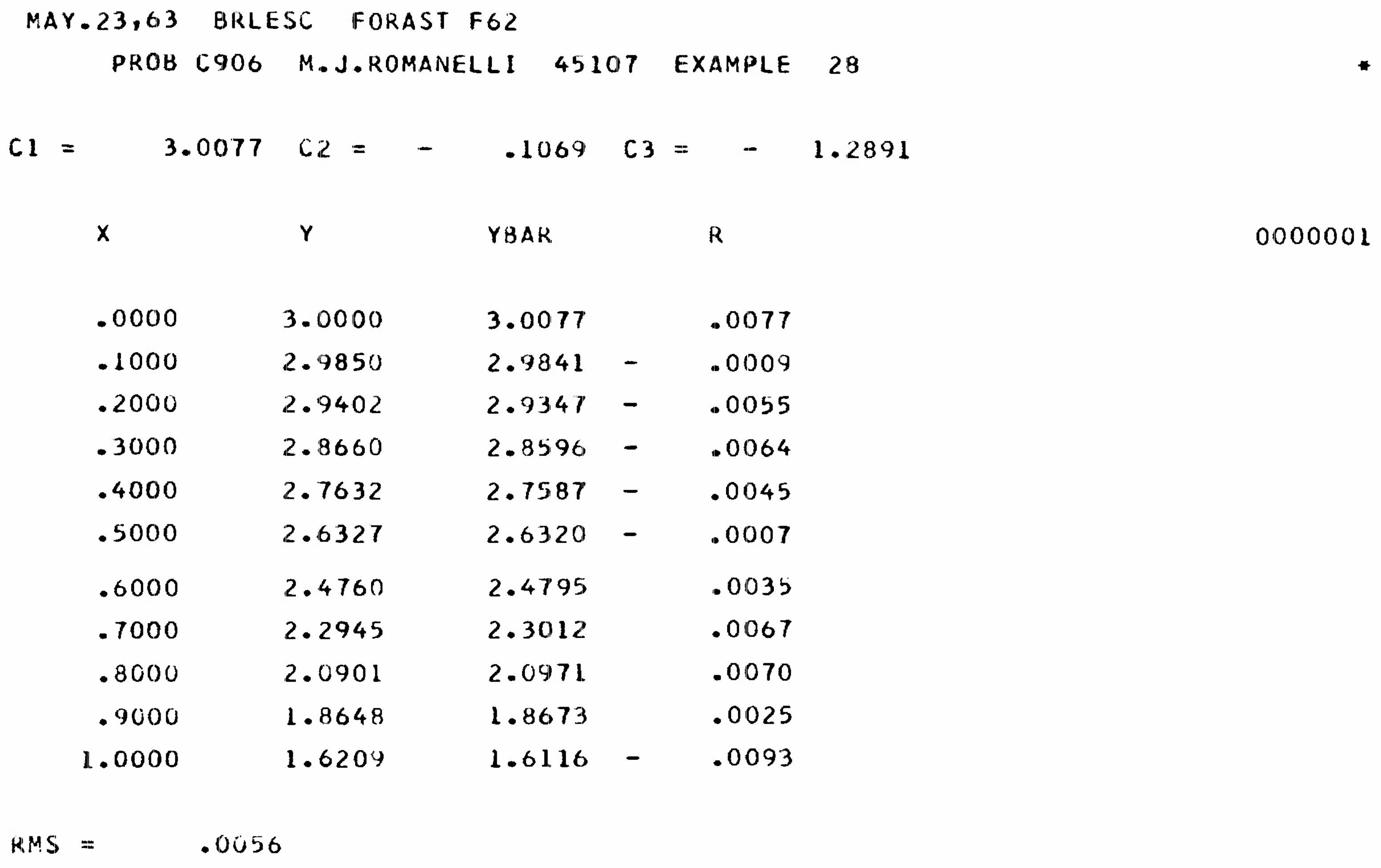
The flow chart for this example is given in FIGURE 56.


START
Clear the augmented symmetric A matrix area i.e., set all $\sum$ = 0
Read given data X_i, y_i $i = 1,2,\ldots,J$ $J \leq 500$ $2J \rightarrow 09$
i = 0
X = X1,i
y = Y1,i
$P_1 = 1$
$P_2 = X$
$P_3 = X^2$
$P_4 = y$
Contribute to Normal Equations
i =i + 2
i < 2J ?
Yes
No
Solve System; obtain C_1, C_2, C_3
PRINT & IDENTIFY
C1 = C2 = C3 =
PRINT Blank
Print HEADING
X Y YBAR R
PRINT Blank
i = 0
S = 0
RESID
$\bar{y}_{i+1} = C_1 + C_2X_{i+1} + C_3X^2_{i+1}$
$f_{i+1} = \bar{y}_{i+1} - y_{i+1}$
$S = \sum r^2_{i+1}$
Print:
X_{i+1}; y_{i+1}; $\bar{y}_{i+1}$; r_{i+1}
i = i + 2
i < 2J ?
Yes
No
RMS = S/J
PRINT: RMS =
N.PROB

FLOW CHART FOR EXAMPLE 28.

FIGURE 56.

```
                                                               23 JULY,63 PAGE    1
         PROB C906  M.J.ROMANELLI  45107  EXAMPLE  28
         BLOC(A1,1-A3,4/SY.)P1-P4)C1-C3)D1-D1000
         SYN (X1=D1)Y1=D2)2J=09)
START        CLEAR(9)NOS.AT(A1,1)% READ-FORMAT(F/2)-(1000)NOS.AT(D1)% SET(I=0)
NEXTPT       P1=1%  P2=X1,I%  P3=P2*P2%  P4=Y1,I
             ENTER(F.N.E.)A1,1)3)P1)% COUNT(2J/2)IN(I)GOTO(NEXTPT)
             ENTER(SY.SNE)A1,1)3)C1)%
             PRINT-FORMAT(F0)-<C1 = >C1<  C2 = >C2<  C3 = >C3% ENTER(PRINT B)
             PRINT<     X          Y          YBAR          R>% ENTER(PRINT B)
             SET(I=0)% S=0
RESID        YBAR=C1+X1,I(C2+C3*X1,I)%  R=YBAR-Y1,I%  S=S+R*R%
             PRINT-FORMAT(F0)-(X1,I)Y1,I)YBAR)R% COUNT(2J/2)IN(I)GOTO(RESID)
             ENTER(CVITOF)09)N% RMS=SQRT(2*S/N)% ENTER(PRINT B)
             PRINT-FORMAT(F0)-<RMS = >RMS% GOTO(N.PROB)
F        FORM(10-12)10-12)2
F0       FORM(12-4-10)3-2)1-4)2
         END GOTO(START)
 0.0        3.0000
  .1        2.9850
  .2        2.9402
  .3        2.8660
  .4        2.7632
  .5        2.6327
  .6        2.4760
  .7        2.2945
  .8        2.0901
  .9        1.8648
 1.0        1.6209
```



```
 MAY.23,63  BRLESC   FORAST F62
      PROB C906  M.J.ROMANELLI  45107  EXAMPLE  28                                      *

C1  =      3.0077  C2  =   -     .1069  C3  =   -    1.2891

      X             Y             YBAR            R                                0000001


     .0000        3.0000        3.0077          .0077
     .1000        2.9850        2.9841   -      .0009
     .2000        2.9402        2.9347   -      .0055
     .3000        2.8660        2.8596   -      .0064
     .4000        2.7632        2.7587   -      .0045
     .5000        2.6327        2.6320   -      .0007
     .6000        2.4760        2.4795          .0035
     .7000        2.2945        2.3012          .0067
     .8000        2.0901        2.0971          .0070
     .9000        1.8648        1.8673          .0025
    1.0000        1.6209        1.6116   -      .0093

RMS  =       .0056
```

FIGURE 57.

EXAMPLE 29.

In the previous example of "least-squares curve-fitting", the approximating function, (the polynomial), was linear in the unknown coefficients. This example illustrates "least-squares curve-fitting" where the approximating function is non-linear in the unknowns. The method illustrated is often referred to as the method of "differential corrections". To illustrate we consider the following problem:

GIVEN:

X_i, $y(X_i)$, $(i = 1,2,3,\ldots)$;

(Here we will assume the given data is recorded on cards in standard form with six values per card), X_1, y_1, X_2, y_2, etc.

A_o and B_o, initial estimates of the unknowns A and B in the approximating function

REQUIRED:

Determine, print and identify, A and B such that

$$S = \sum (\bar{y}(X_i) - y(X_i))^2 \text{ is a minimum,}$$

where

$$\bar{y}(X) = A(B^X) \; .$$

Compute, print and identify residuals and RMS as in the previous example.

The necessary conditions required to minimize S with respect to A and B would lead to a system of equations which is non-linear in the unknowns A and B. Hence, we first expand $\bar{y}(X,A,B)$ in a Taylor series about A_o, B_o. We obtain then

$$\bar{y}(X,A_o + \Delta A,\ B_o + \Delta B) \approx \bar{y}(X,A_o,B_o) + \frac{\partial\bar{y}}{\partial A}\Big|_o \Delta A + \frac{\partial\bar{y}}{\partial B}\Big|_o \Delta B = \bar{\bar{y}}(X,\Delta A,\Delta B)$$

an approximating function which is linear in the unknowns, ΔA and ΔB. (The notation $\Big|_o$ denotes evaluation of the partial derivative at A_o,B_o). Note that we have substituted $A_o + \Delta A$ for A, and $B_o + \Delta B$ for B. The necessary conditions for minimizing

$$\bar{S} = \sum \left[\bar{\bar{y}}(X_i) - y(X_i)\right]^2$$

with respect to ΔA and ΔB are:

$$\frac{\partial\bar{S}}{\partial\Delta A} = 0 = 2\sum_i \left[\qquad\qquad\right] \frac{\partial\bar{y}}{\partial A}\Big|_o$$

$$\frac{\partial\bar{S}}{\partial\Delta B} = 0 = 2\sum_i \left[\qquad\qquad\right] \frac{\partial\bar{y}}{\partial B}\Big|_o$$

or equivalently,

$$\sum_i \left(\frac{\partial\bar{y}}{\partial A}\Big|_o\right)^2 \Delta A + \sum_i \left(\frac{\partial\bar{y}}{\partial A}\Big|_o \frac{\partial\bar{y}}{\partial B}\Big|_o\right) \Delta B = \sum_i \left(y(X_i) - \bar{y}(X_i,A_o,B_o)\right)\frac{\partial\bar{y}}{\partial A}\Big|_o$$

$$\sum_i \left(\frac{\partial\bar{y}}{\partial A}\Big|_o \frac{\partial\bar{y}}{\partial B}\Big|_o\right) \Delta A + \sum_i \left(\frac{\partial\bar{y}}{\partial B}\Big|_o\right)^2 \Delta B = \sum_i \left(y(X_i) - \bar{y}(X_i,A_o,B_o)\right)\frac{\partial\bar{y}}{\partial B}\Big|_o$$

Note that the above system in matrix notation is of the form

$$CD = E$$

where:

$$C = \begin{pmatrix} \sum\limits_i \left(\frac{\partial\bar{y}}{\partial A}\Big|_o\right)^2 & \sum\limits_i \left(\frac{\partial\bar{y}}{\partial A}\Big|_o \frac{\partial\bar{y}}{\partial B}\Big|_o\right) \\ \sum\limits_i \left(\frac{\partial\bar{y}}{\partial A}\Big|_o \frac{\partial\bar{y}}{\partial B}\Big|_o\right)^2 & \sum\limits_i \left(\frac{\partial\bar{y}}{\partial B}\Big|_o\right)^2 \end{pmatrix}$$

the symmetric matrix of coefficients;

$$D = \begin{vmatrix} \Delta A \\ \Delta B \end{vmatrix} \quad , \text{ the unknown vector;}$$

$$E = \begin{pmatrix} \sum\limits_i (y(X_i) - \bar{y}(X_i, A_o, B_o))\frac{\partial\bar{y}}{\partial A}\Big|_o \\ \\ \sum (y(X_i) - \bar{y}(X_i, A_o, B_o))\frac{\partial\bar{y}}{\partial B}\Big|_o \end{pmatrix}$$

Here, as in the previous example, we can use the ENTER(F.N.E.) to form the normal equations by defining:

$$P_1 = \frac{\partial\bar{y}}{\partial A}\Big|_o = B_o^{X_i}$$

$$P_2 = \frac{\partial \bar{y}}{\partial B}\bigg|_o = A_oX_iB_o^{X_i - 1} = A_oX_iP_1/B_o$$

$$P_3 = y(X_i) - \bar{y}(X_i, A_o, B_o) = y(X_i) - A_oP_1$$

After forming the normal equations we can use the ENTER(SY.SNE) to obtain the corrections, ΔA and ΔB. We then form

$$A_1 = A_o + \Delta A$$

$$B_1 = B_o + \Delta B$$

which serve as "new" estimates for A and B for repeating the entire process. To terminate this correction process we will test two conditions:

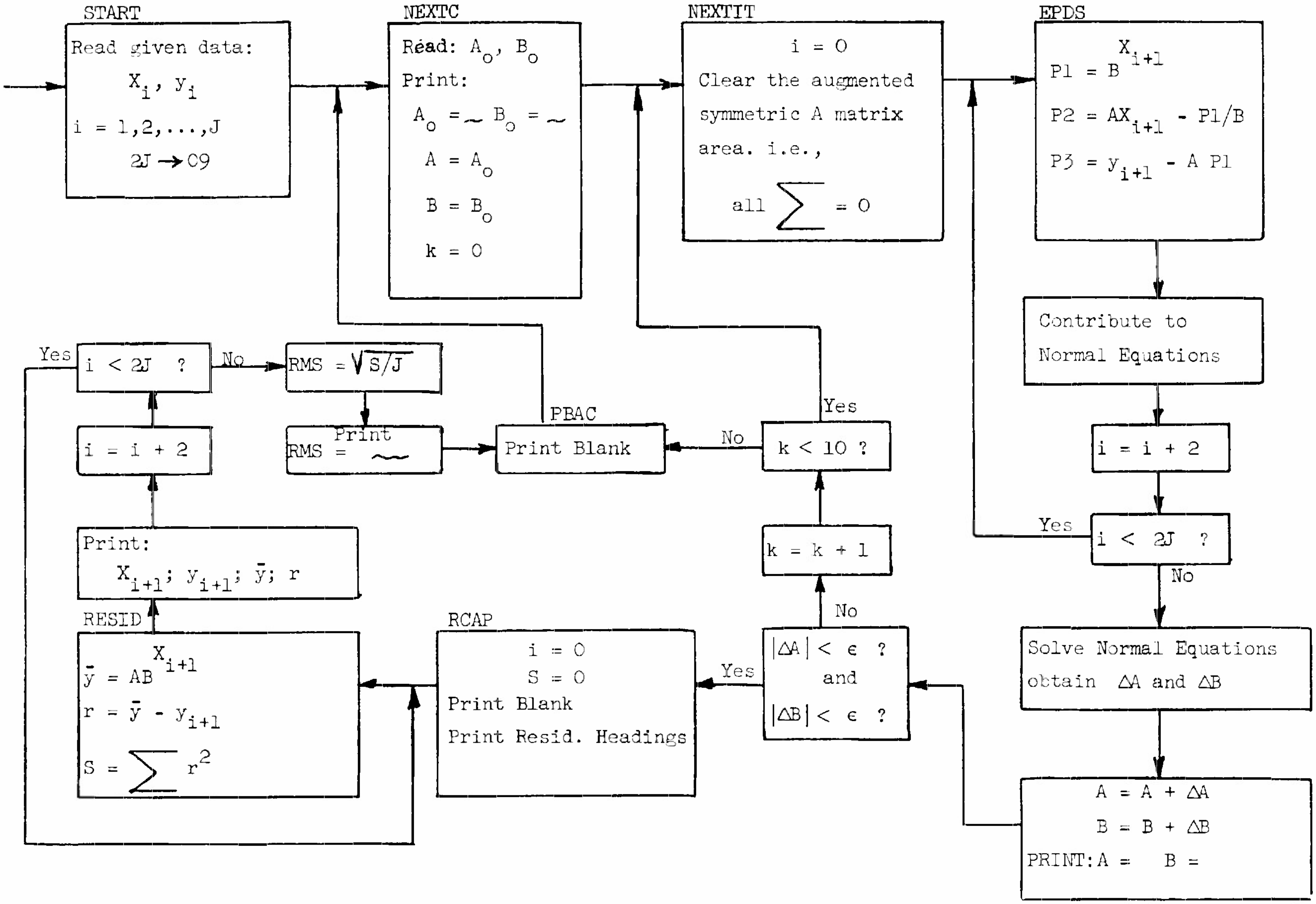
(1.) if both $|\Delta A| < \epsilon$ and $|\Delta B| < \epsilon$ we will consider the problem solved and direct the computer to the computation and printing of the residuals;

(2.) if after 10 corrections, (1.) above is not satisfied, we will assume that the process is diverging and no residuals will be computed.

In either case, (1.) or (2.), we will repeat the process for different initial estimates, A_o and B_o.

The flow chart for this exampleis given in FIGURE 58.

The program and results obtained are listed in FIGURE 59.



FLOW CHART FOR EXAMPLE 29.

FIGURE 58.

```
                                                              23 JULY,63 PAGE    1
      PROB C906 M.J.ROMANELLI  45107  EXAMPLE 29
      BLOC(A1,1-A2,3/SY.)P1-P3)C1-C2)D1-D1000)
      SYN (X1=D1)Y1=D2)
START      READ(1000)NOS.AT(D1)% 2J=09
NEXTC      READ(A0)B0)% PRINT-FORMAT(F)-<A0 = >A0<  B0 = >B0% ENTER(PRINT B)
           A=A0%   B=B0%   SET(K=0)
NEXTIT     SET(I=0)% CLEAR(5)NOS.AT(A1,1)
EPDS       P1=B**X1,I%   P2=A*X1,I*P1/B%   P3=Y1,I-A*P1
           ENTER(F.N.E.)A1,1)2)P1%  COUNT(2J/2)IN(I)GOTO(EPDS)
           ENTER(SY.SNE)A1,1)2)C1%  A=A+C1%  B=B+C2
           PRINT-FORMAT(F)-<A  = >A<  B  = >B
           IF-ABS(C1<.001)AND-ABS(C2<.001)GOTO(RCAP)
           COUNT(10)IN(K)GOTO(NEXTIT)
PBAC       ENTER(PRINT B)% GOTO(NEXTC)
RCAP       SET(I=0)% S=0% ENTER(PRINT B)%
           PRINT<      X            Y            YBAR            R>% ENTER(PRINT B)
RESID      YBAR=A*B**X1,I%  R=YBAR-Y1,I%  S=S+R*R
           PRINT-FORMAT(F)-(X1,I)Y1,I)YBAR)R
           COUNT(2J/2)IN(I)GOTO(RESID)% ENTER(CVITOF)2J)N
           RMS=SQRT(2*S/N)% PRINT-FORMAT(F)-<RMS = > RMS% GOTO(PBAC)
F     FORM(12-4-10)3-2)1-4)2
      END GOTO(START)
 0.0        3.005       0.5        4.223      1.0        5.983
 1.5        8.5         2.0        11.99      2.5        17.
-1.0        1.47       -0.5        2.180     -2.0        .747

 4.0        1.5
 1.0        1.0
```

```
MAY.23,63  BRLESC  FORAST F62
     PROB C906 M.J.ROMANELLI  45107  EXAMPLE 29                                *

A0 =       4.0000  B0 =        1.5000

A  =       3.0115  B  =        1.9496
A  =       2.9963  B  =        2.0025
A  =       2.9972  B  =        2.0016

     X            Y            YBAR            R                        0000004

     .0000      3.0050       2.9972   -     .0078
     .5000      4.2230       4.2404         .0174
    1.0000      5.9830       5.9993         .0163
    1.5000      8.5000       8.4878   -     .0122
    2.0000     11.9900      12.0085         .0185
    2.5000     17.0000      16.9895   -     .0105
-   1.0000      1.4700       1.4974         .0274
-    .5000      2.1800       2.1185   -     .0615
-   2.0000       .7470        .7481         .0011
RMS =      .0253

A0 =       1.0000  B0 =        1.0000

A  =       4.6050  B  =        4.4132
A  =       1.7647  B  =        3.8917
A  =       2.0481  B  =        2.5747
A  =       2.8476  B  =        1.9399
A  =       2.9959  B  =        2.0066
A  =       2.9972  B  =        2.0017
A  =       2.9972  B  =        2.0016

     X            Y            YBAR            R                        0000005

     .0000      3.0050       2.9972   -     .0078
     .5000      4.2230       4.2404         .0174
    1.0000      5.9830       5.9993         .0163
    1.5000      8.5000       8.4878   -     .0122
    2.0000     11.9900      12.0085         .0185
    2.5000     17.0000      16.9895   -     .0105
-   1.0000      1.4700       1.4974         .0274
-    .5000      2.1800       2.1185   -     .0615
-   2.0000       .7470        .7481         .0011
RMS =      .0253
```

FIGURE 59.

EXAMPLE 30.

Some problems require random numbers to simulate errors or other random processes. This example illustrates a "pseudo-random number generator. The numbers are called "pseudo-random" since their genesis is known and any set produced may easily be reproduced. Further, for large enough samples, they do satisfy conventional probability and statistical tests and are of practical use in various problems. In this example, we illustrate a generator of normally distributed pseudo-random numbers. Two ENTER statements are used for generating these numbers; they have the general form

ENTER(NRNOS1)A1)n)B1

ENTER(NRNOS2)A1)n)B1

where:

NRNOS is an abbreviation for Normal Random Numbers; The two distinct entrances, NRNOS1 and NRNOS2, are essential for reproducibility. The initial entry, NRNOS1, always yields the same set of numbers. Subsequent entries at NRNOS2 yield sets which differ from preceding sets.

A1 is the name of the first of n standard deviation values; It is assumed that they are recorded in consecutive locations in internal storage. For standard deviations σ_i, i = 1,2,...,n, the pseudo-random numbers produced, X_i, lie in the interval

$$-4\sigma_i < X_i < 4\sigma_i$$

n is an integer (or name of an integer) which defines the number of pseudo-random numbers desired;

B1 is the name of the first of the n pseudo-random numbers generated.

In general, the programmer should establish the σ_i values in A_i prior to the entry to the generator and should reserve space for these and the resulting values which will be recorded in B_i, $i = 1,2,\ldots,n$.

In the example which follows, we will generate one pseudo-random number with each entry to NRNOS. Further, we will specify a standard deviation $\sigma = 1$. To illustrate, we will generate sets of pseudo-random numbers with set (sample) size equal to N, where N = 200, 400, 600, 800, 1000, 2000, ... , 5000. Corresponding to each set N, we will determine, print and identify the following:

NN the number of negative numbers in the set;

NP the number of positive numbers in the set;

LX the largest number in the set;

SX the smallest number in the set;

M0 the number of X_i in the range, $-4 < X_i \leq -2.9999$;

M1 the number of X_i in the range, $-2.9999 < X_i \leq -1.9999$;

M2 the number of X_i in the range, $-1.9999 < X_i \leq -\ .9999$;

M3 the number of X_i in the range, $= .9999 < X_i < 0$;

P0 the number of X_i in the range, $0 \leqslant X_i < .9999$;

P1 the number of X_i in the range, $.9999 \leq X_i < 1.9999$;

P2 the number of X_i in the range, $1.9999 \leq X_i < 2.9999$;

P3 the number of X_i in the range, $2.9999 \leq X_i < 2.9999$;

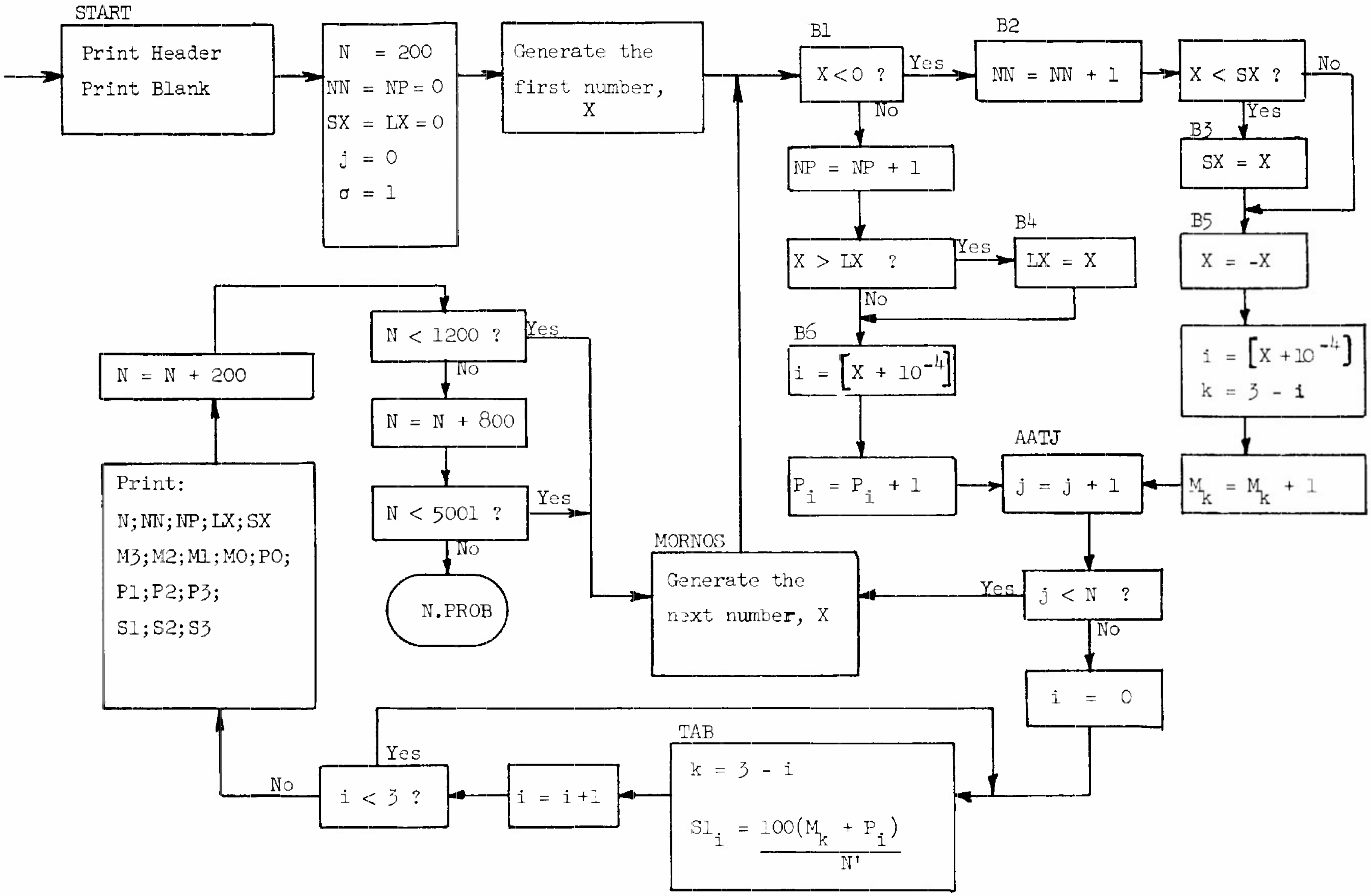
S1 the percentage of N which lie within 1 standard deviation of the mean;

S2 the percentage of N which lie between 1 and 2 standard deviation of the mean;

S3 the percentage of N which lie between 2 and 3 standard deviation of the mean.

The flow chart for this example is given in FIGURE 60.

The program, input and output are listed in FIGURE 61.



FLOW CHART FOR EXAMPLE 30

FIGURE 60.

```
                                                          23 JULY,63 PAGE    1
         PROB C906  M.J.ROMANELLI  45107  EXAMPLE 30
         BLOC (M-M10) %
         SYN  (P=M4)S1=M8) %
START       PRINT<     N      NN     NP    LX     SX    M3    M2    M1    MO>
         CONT <     PO     P1    P2    P3  S1  S2  S3>% ENTER(PRINT B)%
            SET(N=200)NN=0)NP=0)J=0% SX=LX=0% D=1
            ENTER (NRNOS1) D) 1)X % CLEAR(8) NOS. AT (M) %
  B1        IF (X<O) GO TO (B2) % INT (NP=NP+1) % IF (X>LX) GO TO (B4)
  B6        ENTER (CVFTOI)X) I % P,I=P,I+1 % GO TO (AATJ)
  B2        INT (NN=NN+1) % IF (X<SX) GO TO (B3)
  B5        X=-X% ENTER (CVFTOI)X)I)  % INT(K=3-I)%M, K=M, K+1% GO TO (AATJ)
  B3        SX=X % GO TO (B5)
  B4        LX=X % GO TO (B6)
 AATJ       COUNT(N)IN(J)GOTO(MORNOS)% ENTER(CVITOF)N)N'% SET(I=0)GOTO(TAB)%
MORNOS      ENTER (NRNOS2)D)1)X % GO TO (B1)
TAB         INT(K=3-I)% S1,I=100(M,K+P,I)/N'% COUNT(3)IN(I)GOTO(TAB)%
            PRINT-FORMAT (F)- (N)NN)NP)LX)SX)(11)NOS.AT(M)%
            COUNT (1200/200) IN (N) GO TO (MORNOS)
            COUNT (5001/300) IN (N) GO TO (MORNOS)%  GOTO(N.PROB)%
  F      FORM(4-6)1-3)3-1)12-1-5)1-2)3-1)11-3-4)1-3)3-1)11-4-5)1-2)3-1)11-3-4)
         CONT1-2)3-1)1-1)11-2-3)1-4)%
         END GOTO(START)
```

```
MAY.23,63  BRLESC  FORAST F62
     PROB C906  M.J.ROMANELLI  45107  EXAMPLE 30                             *

    N    NN    NP    LX    SX   M3   M2   M1    MO    PO   P1   P2  P3 S1 S2 S3

  200   110    90  3.21 -2.63  000  007  033  0070  0061  025  003  01 65 29 05
  400   205   195  3.21 -2.63  000  009  060  0136  0128  055  011  01 66 29 05
  600   292   308  3.21 -2.63  000  011  086  0195  0211  080  016  01 68 28 04
  800   398   402  3.21 -3.35  001  016  112  0269  0275  107  019  01 68 27 04
 1000   494   506  3.21 -3.35  001  019  148  0326  0348  134  023  01 67 28 04
 2000   977  1023  3.21 -3.92  003  038  274  0662  0716  265  041  01 69 27 04
 3000  1468  1532  3.21 -3.92  004  055  435  0974  1068  403  060  01 68 28 04
 4000  1997  2003  3.21 -3.92  005  073  575  1344  1396  523  083  01 68 27 04
 5000  2493  2507  3.24 -3.92  006  091  720  1676  1750  655  100  02 69 27 04
```

FIGURE 61.

CONCLUSIONS

Specific examples have been employed as a vehicle to enable the novice to program for the ORDVAC and BRLESC Scientific Computers and to obtain appropriate solutions. While the full generality of the FORAST language has not been presented, a sufficient varity of approaches are made available to indicate the flexibility and application of this method. As sophistication is developed, both mathematically and in programming ability, the student should refer to [1.] for a more complete discussion of FORAST. Additions to the language subsequent to the publishing of reference [1.] are found in Appendix A of this report. Detailed information on available subroutines for plotting results can be found in reference [2.] .

ACKNOWLEDGEMENTS

Acknowledgement is made to Dr. B. Garfinkel and Lt. H.B. Tingey for their critical and constructive review of the text.

MICHAEL J. ROMANELLI

REFERENCES

1. Campbell, L. W. and Beck, G. A. The Forast Programming Language for ORDVAC and BRLESC. BRL Report No. 1172, August 1962.

2. Lanahan, J. BRLESC Output Subroutines for Magnetic Tape Dataplotter. BRL Technical Note No. 1495, April 1963.

APPENDIX A.

SOME ADDITIONS TO THE FORAST LANGUAGE

The following additions to the FORAST language are now available.

1.) GOTO,I (A)B)C).............%

where contents of I must be an integer 1,2,3, ...

The transfer of control is directed to A if I = 1
" " " " " " " B if I = 2
" " " " " " " C if I = 3, etc.

A,B,C, etc., may themselves be indexable names.

2.) GOTO,(I ± inc)(A)B)C) .... %

Same as above with provision for incrementing the existing value of I by a positive or negative increment.

<u>NOTE</u>: These additions reduce the size of the <u>ORDVAC</u> SYN Table from 64 to 55. Compiler still checks for synonym full at 64.

3.) ENTER (PLOT)r)X_0)XM)Y_0)YM)h)ix)iy)

where

X_0 = Address of 1st X

XM = Address of last X

Y_0 = Address of 1st Y

YM = Address of last Y

h = Handler number

ix = distance between X entries)
iy = distance between Y entries) Taken as 1 if not entered

If r > 0, tape is rewound before return to program is made.

This subroutine is designed to be used when a "quick look" at some data is desired. It is not necessary to determine beforehand the scales, maximum and minimum values, etc., since the subroutine scans the data and computes the necessary values.

The subroutine produces (on tape) the information for a 26 by 26 inch * plot of the data and a plot of the quantities XMAX, XMIN, XSCALE, YMAX, YMIN, YSCALE as 4 digit (+ or -) integers with 2 digit (+ or -) 10's exponents.

WARNING: Processing data for plotting uses a lot of tape. If there are many large variations in Y throughout the plot, one entry to this subroutine may easily use one-half of a reel of tape.

4.) ENTER(SET.TI)u)E.T)B)BMAX %

optional after first entrance for this tape unit.

u is tape unit integer (eff. address itself is used.)
$1 \leq u \leq 5$ or $9 \leq u \leq 14$

E.T. is optional; if it is zero (or blank) then the routine goes to N.PROB when the END TAPE sentinel is read.
If specified (not zero), then the routine jumps to that address when the END TAPE sentinel is read.

B is the initial address of a block of core storage that is large enough to hold the largest block on the tape being read.

BMAX is the last address in the storage block for this tape.

SET.TI allows a program to read data on magnetic tape. It sets the computer so that subsequent READ statements (or A.READ or READBL subroutines) will cause data to be read from the tape unit specified. Each 80 characters on tape is considered to be a "card" by this routine. (Easier use of "formated" hi-speed printer tapes might be allowed in the future.) As many as six input tapes may be used in one program by entering this subroutine at different times in the program, the data will continue with the "card" that follows the last "card"

* This may be changed to 26 x 13 by means of a manual switch at the plotter board.

that was read from that tape. Each unit should have its own block if the program ever re-uses that unit because a part of a block may need to stay there while another unit is being used. When entering with a unit that was previously used, it is not necessary to specify the storage block addresses; if specified, they will be ignored. It is not possible to change the storage block once it has been already assigned. The storage block may be longer than any block on the tape but must not be shorter than the longest block that is read from the tape. (A "card" requires 8 words of memory storage. The storage addresses may be larger than 04000 in the large memory.) The tape block length can be variable and if any block is longer than the storage allocated, the rest of the block will be ignored. All tape reading is parity checked and re-read five times before causing the erroneous "card" to be punched and a RUN ERROR card saying "PAR.ERRORu".

The "E.T." (end tape) address should be zero unless it is actually needed. If it is zero, this tape unit is rewound, the computer is set to read cards and control goes to N.PROB. If an address is specified, then these things should be done by the program before going to N.PROB.

It is desirable that a standard end of tape sentinel be used by everyone. It is also nice to have a standard end of reel sentinel. This routine uses "ENDbTAPEbb" (b is blank) as the end of tape sentinel when it appears as the first ten characters at the beginning of a block and the next ten characters do not say "ENDbREELbb". When the next ten characters do say "ENDbREELbb", then it assumes that there is another reel to be read on this same unit, so it rewinds the tape and halts at 081 so that the operator can mount the new reel. (The unit no. is in the B address of the halt order.) Standard BRLESC output will have the END TAPE sentinel if the "rewind tape 8" switch was properly used. When making tapes off-line, an extra block of one card with this sentinel should be added at the end of all the data.

When reading tape, the three "header cards" that were produced in front of previous FORAST or FORTRAN output are automatically skipped. (It checks for "bbBRLESCbb" in characters 11-20.) A dictionary will not be automatically skipped, but it is always followed by four blank cards and hence the A.READ routine may be used and your program can check for the blank cards (you need only check the first word) before starting to read actual data.

A word within this routine is named SKP.TL and it may be used to "skip tape lines". If it is set to an integer (not fl.pt.), then the next tape read will skip that many "cards". (If the skip includes the "header cards", then they must be included in the integer that is put into SKP.TL).

The input data may alternate between tape and cards at will. The use of ENTER(SET.CI)% will set the computer for reading cards. (SET.CI is a small subroutine within the SET.TI subroutine. If tape No. 6 is being used instead of card input, then it sets for tape 6 input. If cards or tape 6 is being used at the time SET.CI is entered, it does nothing.)

There are three possible error prints in this subroutine. They are:

SET.TI 6	Tried to use more than six tape units.
SET.TI-BUL	Have negative buffer length. (B > BMAX)(or length > 16,383)
PAR.ERRORu	Parity error on tape unit u. Is preceded by the "card" that contains the error.

It is permissible to set for tape unit u when the same unit u is being used at the time SET.TI is entered.

5.) GOTO(C.PROB)%

C.PROB is the name of a subroutine that allows the compilation and running of several programs without stopping between them. It is similar to N.PROB and does everything that N.PROB normally does except for putting a file mark on output tape 8 and checking for rewinding tape 8. (For purposes of tape 6 input, tape 8 output and for operator control of the computer, several programs combined by using C.PROB will still be considered as one problem.)

Either ENTER or GOTO (or any jump order) may be used to enter C.PROB.

Any new program that is compiled after going to C.PROB must have a PROB card as its first card. It is permissible to allow a READ statement to read this PROB card as the sentinel to indicate that the previous problem is done running if the name C.PROB was used somewhere in the program. (If C.PROB is in the dictionary, then reading a PROB card causes control to go to C.PROB subroutine and the PROB card is also used as the first card of the next program to be compiled.)

In any set of programs, the last one should not use C.PROB. (If this is done and tape 6 input is used, then the next problem will be done as a continuation of your problem.)

The "DATE" will be propagated through all the programs if it precedes the first one.

When using C.PROB, the "card counter" (067) and the input-output options will remain set to what they were at the completion of the previous program. (The permanent constant block (040-07L) is not re-read before the next program is compiled. However ERROR (066) and M.DUMP (058) are reset to N.PROB)

After a RUN ERROR print, control goes to N.PROB (not C.PROB) unless the program has used ERROR as a location.

If tape input has been used (SET.TI subroutine), C.PROB will set for "card input" before compiling the next program.

6.) ENTER(MAX.)A)B)C).......%

MAX is a subroutine that finds the largest floating point number and stores it in the last address specified in the ENTER statement.

7.) ENTER(MIN.)A)B)C)..........%

MIN. is a subroutine that finds the smallest floating point number and stores it in the <u>last</u> address specified.

8.) ENTER(MAX.I)I)J)K).........%

MAX.I is a subroutine that finds the largest integer and stores it in the last address specified.

9.) ENTER(MIN.I)I)J)K).........%

MIN.I is a subroutine that finds the smallest integer and stores it in the last address specified.

In each of the above four subroutines (actually four entrances to one subroutine), the number of arguments is variable and is determined by the number of addresses written in the ENTER statement. (There must be at least two arguments.) The last address is always the store address and is not used as an argument. Each argument is a single number, not a block of numbers.

Care must be exercised in using negative integers as arguments for MAX.I or MIN.I because a "negative integer" in an index register will appear as a large positive integer. (The comparison is made on a full word basis.)

The next order following one of the subroutine entrances must not be a NOP. (It won't be unless you write one as an assembly order.)

10.) ENTER(MOD.)A)B)C %

This subroutine will compute C = A(mod B) where A,B, and C are the respective addresses of floating point numbers as indicated in the ENTER statement above. It is defined as C = A - WHOLE(A/B)*B in terms of a FORAST formula.

11.) Six optional arguments have been added to the MAT MP Subroutine to allow the elements of the rows and columns of the matrices to be stores in equally spaced memory positions.

ENTER(MAT.MP)A_{11})B_{11})C_{11})i)j)k)z)r_a)c_a)r_b)c_b)r_c)c_c)%

where

A_{11},B_{11},C_{11} are the addresses of the first elements of the respective matrices.

i, j, k are the dimensions of the matrices (j is always the common dimension.)

z is a 3 digit sexadecimal number ($0d_1d_2d_3$)

same as above

where d_1 indicates the options applying to matrix A, d_2 to matrix B, and d_3 to matrix C.

d_i = 0 means to use the matrix as stored, not augmented

d_i = 1 means to use the transpose of the matrix stored.

d_i = 2 means the matrix is augmented

d_i = 3 means both 1 and 2 apply

if d_3 = 4, 5, 6, or 7 it means to accumulate in C as well as above options.

r_a is the distance between the first elements of each row of matrix A.

(ie) address A_{21} - address A_{11}

c_a is the distance between the first elements of each column of matrix A.

(ie) address A_{12} - address A_{11}

r_b, c_b ; r_c, c_c have the same meaning for matrices B and C.

Pairs of arguments may be omitted from the right. Zero arguments for r and c are not valid. When r and c is specified, the augment portion of z for that matrix is ignored.

EXAMPLE:

Given: a_{11} b_{11} c_{11} a_{12} b_{12} c_{12} a_{13} b_{13} c_{13} x x x x

a_{21} b_{21} c_{21} a_{22} b_{22} c_{22} a_{23} b_{23} c_{23} x x x x

a_{31} b_{31} c_{31} a_{32} b_{32} c_{32} a_{33} b_{33} c_{33} x x x x

ENTER(MAT.MP)a_{11})b_{11})c_{11})3)3)3)0104)13)3)13)3)13)3)%

would produce

$$A^T B + C \text{ in } C$$

i.e., $c_{11} = c_{11} + a_{11}b_{11} + a_{21}\ b_{21} + a_{31}b_{31}$, etc.

AD________________ Accession No.________________ UNCLASSIFIED

Ballistic Research Laboratories, APG
INTRODUCTORY PROGRAMMING FOR ORDVAC AND BRLESC
FORAST (Formula and Assembly Translator)
Michael J. Romanelli

Computers - Programming
Programming language
ORDVAC - Programming
BRLESC - Programming

BRL Report No. 1209 July 1963

RDT & E Project No. 1M010501A003
UNCLASSIFIED Report

FORAST is a programming language designed for use on ORDVAC and BRLESC, the high-speed digital computers of the Ballistic Research Laboratories. Programs written in this language, with minor limitations, may be executed on either computer. BRL Report No. 1172, [1], describes FORAST in its generality and was written primarily for professional programmers. This report is intended for the novice. Fundamental concepts and details of the language are illustrated in many examples so that the novice is taught how to program and obtain practical solutions for a variety of mathematical problems. Intended as a supplement to [1], this report does not illustrate the full generality of the language. Some of the material is repetitious but amplified and several references are made to [1].

AD________________ Accession No.________________ UNCLASSIFIED

Ballistic Research Laboratories, APG
INTRODUCTORY PROGRAMMING FOR ORDVAC AND BRLESC
FORAST (Formula and Assembly Translator)
Michael J. Romanelli

Computers - Programming
Programming language
ORDVAC - Programming
BRLESC - Programming

BRL Report No. 1209 July 1963

RDT & E Project No. 1M010501A003
UNCLASSIFIED Report

FORAST is a programming language designed for use on ORDVAC and BRLESC, the high-speed digital computers of the Ballistic Research Laboratories. Programs written in this language, with minor limitations, may be executed on either computer. BRL Report No. 1172, [1], describes FORAST in its generality and was written primarily for professional programmers. This report is intended for the novice. Fundamental concepts and details of the language are illustrated in many examples so that the novice is taught how to program and obtain practical solutions for a variety of mathematical problems. Intended as a supplement to [1], this report does not illustrate the full generality of the language. Some of the material is repetitious but amplified and several references are made to [1].

AD________________ Accession No.________________ UNCLASSIFIED

Ballistic Research Laboratories, APG
INTRODUCTORY PROGRAMMING FOR ORDVAC AND BRLESC
FORAST (Formula and Assembly Translator)
Michael J. Romanelli

Computers - Programming
Programming language
ORDVAC - Programming
BRLESC - Programming

BRL Report No. 1209 July 1963

RDT & E Project No. 1M010501A003
UNCLASSIFIED Report

FORAST is a programming language designed for use on ORDVAC and BRLESC, the high-speed digital computers of the Ballistic Research Laboratories. Programs written in this language, with minor limitations, may be executed on either computer. BRL Report No. 1172, [1], describes FORAST in its generality and was written primarily for professional programmers. This report is intended for the novice. Fundamental concepts and details of the language are illustrated in many examples so that the novice is taught how to program and obtain practical solutions for a variety of mathematical problems. Intended as a supplement to [1], this report does not illustrate the full generality of the language. Some of the material is repetitious but amplified and several references are made to [1].

AD________________ Accession No.________________ UNCLASSIFIED

Ballistic Research Laboratories, APG
INTRODUCTORY PROGRAMMING FOR ORDVAC AND BRLESC
FORAST (Formula and Assembly Translator)
Michael J. Romanelli

Computers - Programming
Programming language
ORDVAC - Programming
BRLESC - Programming

BRL Report No. 1209 July 1963

RDT & E Project No. 1M010501A003
UNCLASSIFIED Report

FORAST is a programming language designed for use on ORDVAC and BRLESC, the high-speed digital computers of the Ballistic Research Laboratories. Programs written in this language, with minor limitations, may be executed on either computer. BRL Report No. 1172, [1], described FORAST in its generality and was written primarily for professional programmers. This report is intended for the novice. Fundamental concepts and details of the language are illustrated in many examples so that the novice is taught how to program and obtain practical solutions for a variety of mathematical problems. Intended as a supplement to [1], this report does not illustrate the full generality of the language. Some of the material is repetitious but amplified and several references are made to [1].

AD_______________ Accession No._______________ UNCLASSIFIED

Ballistic Research Laboratories, APG
INTRODUCTORY PROGRAMMING FOR ORDVAC AND BRLESC
FORAST (Formula and Assembly Translator)
Michael J. Romanelli

Computers - Programming
Programming language
ORDVAC - Programming
BRLESC - Programming

BRL Report No. 1209 July 1963

RDT & E Project No. 1M010501A003
UNCLASSIFIED Report

FORAST is a programming language designed for use on ORDVAC and BRLESC, the high-speed digital computers of the Ballistic Research Laboratories. Programs written in this language, with minor limitations, may be executed on either computer. BRL Report No. 1172, [1], describes FORAST in its generality and was written primarily for professional programmers. This report is intended for the novice. Fundamental concepts and details of the language are illustrated in many examples so that the novice is taught how to program and obtain practical solutions for a variety of mathematical problems. Intended as a supplement to [1], this report does not illustrate the full generality of the language. Some of the material is repetitious but amplified and several references are made to [1].

AD_______________ Accession No._______________ UNCLASSIFIED

Ballistic Research Laboratories, APG
INTRODUCTORY PROGRAMMING FOR ORDVAC AND BRLESC
FORAST (Formula and Assembly Translator)
Michael J. Romanelli

Computers - Programming
Programming language
ORDVAC - Programming
BRLESC - Programming

BRL Report No. 1209 July 1963

RDT & E Project No. 1M010501A003
UNCLASSIFIED Report

FORAST is a programming language designed for use on ORDVAC and BRLESC, the high-speed digital computers of the Ballistic Research Laboratories. Programs written in this language, with minor limitations, may be executed on either computer. BRL Report No. 1172, [1], describes FORAST in its generality and was written primarily for professional programmers. This report is intended for the novice. Fundamental concepts and details of the language are illustrated in many examples so that the novice is taught how to program and obtain practical solutions for a variety of mathematical problems. Intended as a supplement to [1], this report does not illustrate the full generality of the language. Some of the material is repetitious but amplified and several references are made to [1].

AD_______________ Accession No._______________ UNCLASSIFIED

Ballistic Research Laboratories, APG
INTRODUCTORY PROGRAMMING FOR ORDVAC AND BRLESC
FORAST (Formula and Assembly Translator)
Michael J. Romanelli

Computers - Programming
Programming language
ORDVAC - Programming
BRLESC - Programming

BRL Report No. 1209 July 1963

RDT & E Project No. 1M010501A003
UNCLASSIFIED Report

FORAST is a programming language designed for use on ORDVAC and BRLESC, the high-speed digital computers of the Ballistic Research Laboratories. Programs written in this language, with minor limitations, may be executed on either computer. BRL Report No. 1172, [1], describes FORAST in its generality and was written primarily for professional programmers. This report is intended for the novice. Fundamental concepts and details of the language are illustrated in many examples so that the novice is taught how to program and obtain practical solutions for a variety of mathematical problems. Intended as a supplement to [1], this report does not illustrate the full generality of the language. Some of the material is repetitious but amplified and several references are made to [1].

AD_______________ Accession No._______________ UNCLASSIFIED

Ballistic Research Laboratories, APG
INTRODUCTORY PROGRAMMING FOR ORDVAC AND BRLESC
FORAST (Formula and Assembly Translator)
Michael J. Romanelli

Computers - Programming
Programming language
ORDVAC - Programming
BRLESC - Programming

BRL Report No. 1209 July 1963

RDT & E Project No. 1M010501A003
UNCLASSIFIED Report

FORAST is a programming language designed for use on ORDVAC and BRLESC, the high-speed digital computers of the Ballistic Research Laboratories. Programs written in this language, with minor limitations, may be executed on either computer. BRL Report No. 1172, [1], described FORAST in its generality and was written primarily for professional programmers. This report is intended for the novice. Fundamental concepts and details of the language are illustrated in many examples so that the novice is taught how to program and obtain practical solutions for a variety of mathematical problems. Intended as a supplement to [1], this report does not illustrate the full generality of the language. Some of the material is repetitious but amplified and several references are made to [1].