

UNIVERSITY OF ILLINOIS
GRADUATE COLLEGE
DIGITAL COMPUTER LABORATORY

GENERALIZED
ALGEBRAIC
TRANSLATOR-
COMPILER
(GAT COMPILER)

File No. 384
July 3, 1961

"Certain basic features of this translator were suggested by the original Internal Translator (IT) written by Prof. A. Perlis et. al. of Carnegie Institute of Technology. In particular the use of subscripted letters as variables and the general form of the permissible statements are the same as the original IT. In addition we are indebted to J. Smith of Tech Ops for some very helpful comments regarding the decomposition of algebraic statements.

"The operating packages were written by W. Hanson, who wrote the input-output routines, and T. O'Brien, P. Anderson and C. Page who adapted the functional subroutines for use with the Translator. The manual was the joint effort of R. Graham, Prof. B. Galler, B. Arden, and Mrs. Sarah Brando who bore the clerical burden of typing and the preparation of flow charts."

From the original GAT manual
University of Michigan

The original GAT manual was extensively revised by J. W. Flenner and C. Wilmot, advised by L. D. Fosdick, S. J. Fenves, J. N. Snyder. The control panels for the 533 and 407 were designed by R. H. Flenner.

"But screw your courage to the sticking-place, and we'll not fail."

Lady Macbeth from Macbeth
by William Shakespeare, 1605

TABLE OF CONTENTS

	<u>Page</u>
PART I. INTRODUCTION TO THE GAT COMPILER	1
PART II. THE ALGEBRAIC CODING LANGUAGE	21
SECTION 1. CONSTANTS AND VARIABLES	23
SECTION 2. SUBSCRIPTS	26
SECTION 3. ARITHMETIC	28
SECTION 4. STATEMENTS	32
SECTION 5. SUBROUTINES	45
PART III. GAT INPUT-OUTPUT FORMATS	47
SECTION 1. PREPARATION OF CARDS	49
SECTION 2. OUTPUT FORMATS	55
APPENDIX A. SUMMARY OF LANGUAGE	59
APPENDIX B. SUBROUTINE LIBRARY	63
APPENDIX C. ERROR INDICATIONS	71
APPENDIX D. EXAMPLES	75

PART I

INTRODUCTION TO THE GAT COMPILER

The purpose of GAT is to simplify the process of preparing problems for the IBM 650. Since GAT is a program which allows use of a familiar and convenient language for presenting problems to the IBM 650, it is not necessary to use the restricted list of machine-language codes. In general, a program which does this is called a compiler; that is, it converts this familiar language (called the source language) to a series of machine language instructions (called the object program) which perform the desired operations. This conversion process is called compilation.

The presentation of problems to the computer via the compiler can best be illustrated by some examples.

Example I. Assume that the following computational problem is to be repeated many times, taking different values for all the variables, including n , on each repetition.

Given values for variables x_1, x_2, \dots, x_n ;
 y_1, y_2, \dots, y_n ; and n ,
compute $d = x_1y_1 + x_2y_2 + \dots + x_ny_n$.

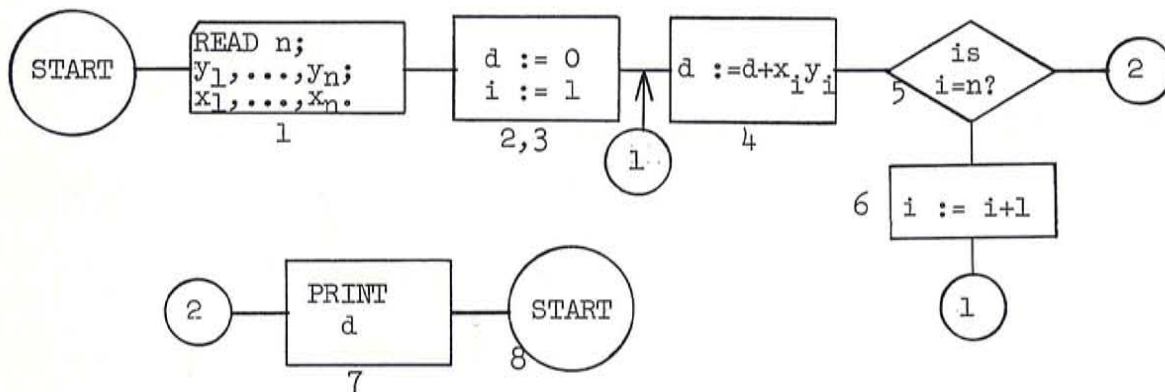
(i.e., Compute the inner product of 2 n -dimensional vectors.)

A prose description of the computation procedure might be as follows:



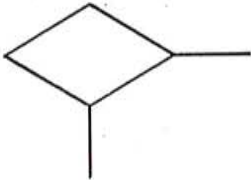
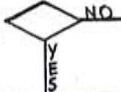
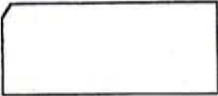

1. Obtain the variable values for the problem to be computed.
2. Set the accumulator to be used to zero. (i.e., Take a blank sheet of paper, clear the dials of the desk calculator, etc.)
3. Take the first numbers of each set.

4. Obtain the product of the selected numbers and add the product to the accumulator.
5. If the product just obtained was the last (i.e., the n^{th}) do step 7 next — if not, do step 6.
6. Take the next numbers of each set and then do step 4.
7. Copy the answer from the accumulator.
8. Advance to the next set of data and then repeat steps 1 through 8 until all of the problems have been handled.

In actual practice the first step in preparing a problem for a computer is not to bother to write a prose description of the procedure but to draw a flow diagram to describe the process. A flow diagram is a pictorial representation of the sequence of operations necessary for the correct solution of the problem.



The number associated with each box indicates the corresponding original prose statement.

FLOW DIAGRAM CONVENTIONS	
	Remote Connectors (i.e., circles containing the same number are assumed to be the same point).
	Processing Boxes.
	Decision Boxes. These contain questions to be answered yes or no. The yes answer takes the path to the right and the no answer down, unless otherwise labeled. e.g. ( , the unconventional case).
	Card Operations (Data input-output).
$\alpha :=$	Set value α to quantity on other side of equal sign; e.g., in above example, " $d := 0$ " means "set d equal to zero".
	Indicates an <u>exception</u> to standard flow pattern. (Direction of standard flow is always to the right down.)

By abstracting from the flow diagram it becomes possible to produce a set of statements describing the process, free from the possible ambiguities of prose:

1. Read $n, x_1, \dots, x_n, y_1, \dots, y_n$
2. $d = 0$
3. $i = 1$
4. $d = d + x_i y_i$
5. Go to 7 if $i = n$
6. $i = i + 1$

- Go to 4
- 7. Print d
- 8. Go to 1.

These statements are now almost in the form on which the compiler can operate. The necessary additions and changes to these statements arise from two facts: 1) the computer can accept only a limited number of characters, and 2) the limited internal capacity of the computer makes it necessary to include some information about the size of the problem.

The 650 can accept only numbers, capital (upper case) alphabetic characters, and the special characters $+ - / * () = , . \$$. Also, the commonly used notation, such as a_i and a^x , cannot be handled directly by the computer, since the 650 is not cognizant of carriage shifts. The GAT compiler requires that the names of all variables be one of the letters C, D, I, J, K, X, Y, Z, followed, without a carriage shift, by a permissible subscript (p. 26). For example, the variable x_3 might be given the name X3, the variable i the name I1, and x_i would be XI1.

With this in mind we will rename the variables of the example as follows:

- n is renamed I1
- i is renamed I2
- d is renamed D1
- x_1, \dots, x_n are renamed X1, ..., XI1
- y_1, \dots, y_n are renamed Y1, ..., YI1 .

In order for the GAT compiler to allocate the available storage it must know:

- a) The highest statement number used (p. 41).
- b) A maximum number of locations used by subroutines (p. 41).

c) The maximum subscript used for each allowable letter (p. 42).

In this example 8 is the highest statement number. Assume that 400 locations are used by subroutines and that the maximum value of n is 100.

The final form of the statements (these are now acceptable to GAT) describing the example problem are:

```
      8 IS HIGHEST STATEMENT NUMBER
      400 USED IN SUBROUTINES
      DIMENSION D(1) I(2) X(100) Y(100)
1     I1, X1, ..., X11, Y1, ..., Y11, READ
2     D1 = 0
3     I2 = 1
4     D1 = D1 + XI2 * YI2
5     GO TO 7 IF I2 U I1
6     I2 = I2 + 1
      GO TO 4
7     TD1
8     GO TO 1
      END
```

Notice that in statement 5 the letter "U" is used to indicate "="; in statement 7 "T" means print.

Although the above source language statements are acceptable to GAT, use of a specialized statement called an iteration statement (p. 34) will allow compilation of a more efficient object program.

```
      8 IS HIGHEST STATEMENT NUMBER
      400 USED IN SUBROUTINES
      DIMENSION D(1) I(2) X(100) Y(100)
```

```
1  I1, X1, ..., XI1, Y1, ..., YI1, READ
2  D1 = 0
3  4, I2, 1, 1, I1,
4  D1 = D1 + XI2 * YI2
7  TD1
8  GO TO 1

END
```

The iteration statement (statement 3) replaces statements 3, 5, and 6 in the previous source language example. It should be read as follows:

Set I2 initially to 1; repeat statement 4 increasing I2 by 1 each time; go to the statement following No. 4 when $I2 > I1$.

EXAMPLE I - PRODUCT OF TWO N-DIMENSIONAL VECTORS

SOURCE LANGUAGE

```

8 IS HIGHEST STATEMENT NUMBER
400 USED IN SUBROUTINES
DIMENSION D(1) I(2) X(100)  N
Y(100)
1  I1,X1,....,XI1,Y1,....,YI1,READ
2  D1=0
3  4,I2,1,1,I1,
4  D1=D1+XI2*YI2
7  TD1
8  GO TO 1
END

```

OBJECT PROGRAM

2	0244	0000011990+	1990	8803970024+
3	0357	8803680023+	0368	2119490361+
3	0405	6002540457+	0457	2100400455+
3	0557	2100400505+	0505	6000400607+
4	0247	0000040411+	0411	6000400707+
4	0555	6000400807+	0807	8080030611+
7	0711	2100370605+	0605	0003560461+
8	0655	6002550957+	0957	2119610705+
100	0755	0007610244+	0256	0000010007+
100	0252	0000000004+	0000	0100000000+
100	0004	7100008010+	0005	7200008010+
100	0009	0000090243+	0010	0001000252+
100	0014	0000000000+	0015	0000000000+
100	0019	0000001480+	0020	0000000000+
100	0024	7965616400+	0025	7577756368+
100	0029	0000000000+	0030	0000000000+
100	0034	0000000000+	0035	0000000000+

DATA

I1	5		
X1	1000000051	ETC	2000000051
Y1	1000000051	ETC	2000000051

RESULTS

D1	5500000052
----	------------

INTRODUCTION

11

0397	2119500355+	0245	0000020355+	0355	6002530357+
0361	6019490407+	0407	2100370405+	0246	0000030405+
0455	0004610411+	0461	6002540507+	0507	1000400557+
0607	1100390657+	0657	6180030511+	0511	4603560411+
0707	8080030561+	0561	6021420757+	0757	2119500555+
0611	6020410857+	0857	3919500661+	0661	3200370711+
0250	0000070356+	0356	6002560907+	0907	2119600655+
0705	8803620025+	0362	2119500755+	0251	0000080755+
0255	0000020001+	0254	0000000001+	0253	0000000000+
0001	6300008010+	0002	6400020036+	0003	6900030038+
0006	8701010041+	0007	8801010142+	0008	8900008010+
0011	0000000000+	0012	0000000000+	0013	0000000000+
0016	0000000000+	0017	0000000000+	0018	0000000000+
0021	0000000000+	0022	0000000000+	0023	6673766183+
0026	0000000000+	0027	0000000000+	0028	0000000000+
0031	0000000000+	0032	0000000000+	0033	0000000000+
0243	0000008000+	0243	0000008000+	0243	0000008000+

ETC 3000000051
ETC 3000000051

ETC 4000000051
ETC 4000000051

ETC 5000000051
ETC * 5000000051

Example II. Some other features of the language can be illustrated by a second example.

In an A.C. circuit consisting of a resistance R, an impedance L and a capacitance C, subjected to a potential of the form

$$v = V \sin 2\pi ft, \quad (1)$$

the instantaneous current is given as

$$i = \frac{Vn}{\sqrt{R^2 + (\omega L - \frac{1}{\omega C})^2}} \sin (\omega t - \phi) , \quad (2)$$

where

$$\omega = 2\pi f$$

and

$$\phi = \tan^{-1} \frac{\omega L - \frac{1}{\omega C}}{R} .$$

For particular values of V, R, L, C and f, we wish to plot the values of v and i during one cycle (i.e., for ωt going from 0 to 2π). Since we cannot generate the plots directly, we will compute the ordinates of v and i at twenty equal time intervals. These ordinates can then be plotted and connected by a smooth curve.

We first change the given quantities to the compiler notation:

<u>Quantity</u>	<u>GAT Notation</u>	<u>Value For Sample Problem</u>
V	C1	110 volts
R	C2	1000 ohms
L	C3	0.5 henry
C	C4	2.0 μ F (micro-farads)
f	C5	60 cycles/sec.

Reading of the data is accomplished by the statement:

1. VOLT, OHM, HENRY, FARAD, CPS, READ (see Note 1, p. 14).

The second step is to compute the quantities which do not depend on the time t , as follows:

<u>Computation</u>	<u>GAT Statements</u>
Angular frequency $\omega = 2\pi f$	2. $D1 = 2.*3.1415927 * C5$
Reactance $x = \omega L - \frac{1}{\omega C}$	3. $D2 = D1*C3 - 1.0/(D1*C4)$
Impedance $Z = \sqrt{R^2 + X^2}$	4. $D3 = \text{SQRT}.(C2*C2 + D2*D2)$
Max. current $I = \frac{V}{Z}$	5. $D4 = C1/D3$
Phase angle $\phi = \tan^{-1} \frac{X}{R}$	6. $D5 = \text{ARTAN}.(D2/C2)$
	7. $\text{TD1} \dots D5$

The statement 7. $\text{TD1} \dots D5$ causes the five quantities $D1$ through $D5$ to be printed.

Finally, we are ready to compute v and i at the desired time intervals. The time t is to be varied from 0 to $\frac{1}{f}$ in twenty increments of $\frac{1}{20f}$. This can be accomplished by an iteration statement similar to the one given in the preceding example.

<u>Computation</u>	<u>GAT Statements</u>
Vary t from 0 to $\frac{1}{f}$ in increments of $\frac{1}{20f}$	8. $\text{11}, X1, 0, 1./(20.*C5), 1./C5,$
$v = V \sin 2\pi ft$	9. $X2 = C1 * \text{SIN}.(D1*X1)$
$i = I \sin (2\pi ft - \phi)$	10. $X3 = D4 * \text{SIN}.(D1*X1 - D5)$
Print t, v, i	11. $\text{TX1} \dots X3$

The statements, ready for compilation, are as follows:

12 IS HIGHEST STATEMENT NUMBER

800 USED IN SUBROUTINES

DIMENSION C(5) D(5) X(3)

```
1. VOLT, OHM, HENRY, FARAD, CPS, READ
2. D1 = 2.*3.1415927*C5
3. D2 = D1*C3 - 1./(D1*C4)
4. D3 = SQRT.(C2*C2 + D2*D2)
5. D4 = C1/D3
6. D5 = ARTAN.(D2/C2)
7. TD1 ... D5
8. 11, X1, 0., 1.1(20.*C5), 1./C5,
9. X2 = C1*SIN.(D1*X1)
10. X3 = D4*SIN.(D1*X1 - D5)
11. TX1 ... X3
12. GO TO 1

      END
```

In the statements above, notice the following:

Note 1) Preceding the READ statement, information may be added about the quantities being read for the program writer's convenience (p. 39). The assignment of the symbols C1, ..., C5 is done by punching on the data card (p. 51).

Note 2) Functions such as square root, sine, arctangent, which are not part of the compiler's language, may be added by subroutines (p. 45). The statement "800 USED IN SUBROUTINES" reserves space in the object program for the inclusion of these subroutines.

Finally it is to be noted that the same problem could be solved without precomputing the quantities D1 through D5, by solving equations (1) and (2) as originally stated for each value of t . Such a program would be much more inefficient, since the square root, arctangent, etc., would have to be evaluated ten times instead of once.

The results produced by the object program for the sample values are shown on p. 18.

The preceding examples serve to illustrate the various types of statements allowable in the source language. A more careful and systematic description of the language follows.

EXAMPLE II - POTENTIAL AND
INSTANTANEOUS CURRENT IN AN
A.C. CIRCUIT

SOURCE LANGUAGE

```

12 IS HIGHEST STATEMENT NUMBER
800 USED IN SUBROUTINES
DIMENSION C(5) D(5) X(3)
1  VOLT,OHM,HENRY,FARAD,CPS,READ
2  D1=2.*3.1415927*C5
3  D2=D1*C3-1./(D1*C4)
4  D3=SQRT.(C2*C2+D2*D2)
5  D4=C1/D3
6  D5=ATAN.(D2/C2)
7  TD1...D5
8  I1,X1,0.,1./(20.*C5),1./C5,
9  X2=C1*SIN.(D1*X1)
10 X3=D4*SIN.(D1*X1-D5)
11 TX1...X3
12 GO TO 1
END

```

OBJECT PROGRAM

2	0053	0000011990+	1990	8802100024+
3	0170	3900670174+	0174	3900410224+
3	0220	3900430274+	0274	2119500268+
4	0318	6000390320+	0320	3900430374+
4	0368	6000440370+	0370	3900440474+
5	0524	3219500574+	0574	2119600468+
6	0518	6000370470+	0470	3400450624+
7	0520	3400380674+	0674	2119600618+
7	0668	6000710570+	0570	2119600718+
8	0670	2119620818+	0818	8802880027+
8	0720	2100490918+	0918	0002370187+
8	0968	6000680820+	0820	3419500774+
8	0870	3400410874+	0874	2119501068+
9	0974	4601690187+	0061	0000090187+
10	0219	8801890028+	0189	3900371074+
10	1020	3900430175+	0175	3300470225+
11	0275	2100510369+	0063	0000110369+
11	0171	2119610469+	0469	6000740221+
100	0569	0001690237+	0064	0000120169+
100	0074	0000060003+	0073	2000000052+
100	0069	0000020005+	0068	1000000051+
100	0000	0100000000+	0001	6300060036+
100	0005	7200008010+	0006	8700040048+
100	0010	0001000065+	0011	0000000000+
100	0015	0000000000+	0016	0000000000+
100	0020	0000000000+	0021	0000000000+
100	0025	8278798300+	0026	6179836175+
100	0030	0000000000+	0031	0000000000+
100	0035	0000000000+	0052	0000008000+

0210	2119500168+	0054	0000020168+	0168	6000660170+
0224	2100430218+	0055	0000030218+	0218	6000400220+
0268	6000680270+	0270	3419500324+	0324	2119500318+
0374	3319500424+	0424	2100440368+	0056	0000040368+
0474	2119500418+	0418	6000380420+	0420	3900380524+
0468	8801880025+	0188	2100450518+	0057	0000050518+
0624	2100460568+	0058	0000060568+	0568	6000440520+
0618	8802380026+	0238	2100470668+	0059	0000070668+
0718	6000700620+	0620	2119610768+	0768	6000690670+
0288	2119500868+	0060	0000080868+	0868	6000720720+
0237	6000410770+	0770	3900730724+	0724	2119500968+
0774	3200490824+	0824	2100491018+	1018	6000680870+
1068	6000490920+	0920	3319500924+	0924	6180030974+
0187	6000490970+	0970	3900431024+	1024	2119600219+
1074	2100500269+	0062	0000100269+	0269	6000491020+
0225	2119600319+	0319	8802390028+	0239	3900460275+
0369	6000761070+	1070	2119600419+	0419	6000750171+
0221	2119620519+	0519	8802890027+	0289	2119500569+
0169	0003380053+	0076	0000020011+	0075	0800060001+
0072	0000000000+	0071	0000020007+	0070	0800020001+
0067	20000000051+	0066	3141592751+	0065	0000000011+
0002	6400060042+	0003	6900008010+	0004	7100008010+
0007	8800008010+	0008	8900008010+	0009	0000130052+
0012	0000000000+	0013	0000000000+	0014	0000000000+
0017	0000000000+	0018	0000000000+	0019	0000001080+
0022	0000000000+	0023	0000000000+	0024	7965616400+
0027	7577756368+	0028	8269750000+	0029	0000000000+
0032	0000000000+	0033	0000000000+	0034	0000000000+
0052	0000008000+	0052	0000008000+	0052	0000008000+

DATA

C1	1100000053	C2	1000000054
----	------------	----	------------

RESULTS

D1	3769911253	D2	1137795654-
X1		X2	
X1	8333333347	X2	3399187352
X1	1666666748	X2	6465637552
X1	2500000048	X2	8899186752
X1	3333333348	X2	1046162253
X1	4166666648	X2	1100000053
X1	4999999948	X2	1046162253
X1	5833333248	X2	8899186752
X1	6666666548	X2	6465637552
X1	7499999848	X2	3399187352
X1	8333333148	X2	
X1	9166666448	X2	3399186352-
X1	9999999748	X2	6465637552-
X1	1083333349	X2	8899186552-
X1	1166666649	X2	1046162153-
X1	1249999949	X2	1100000053-
X1	1333333249	X2	1046162253-
X1	1416666549	X2	8899190152-
X1	1499999849	X2	6465643752-
X1	1583333149	X2	3399194652-
X1	1666666449	X2	9900000046-

C3 5000000050

C4 2000000045

C5 * 6000000052

D3 1514786754

D4 7261748549

D5 8497660050-

X3 5454487749

X3 6668925549

X3 7230561949

X3 7084420549

X3 6244806649

X3 4793908049

X3 2873747749

X3 6732497748

X3 1594985049-

X3 3706126949-

X3 5454487749-

X3 6668925049-

X3 7230561449-

X3 7084420449-

X3 6244807349-

X3 4793909649-

X3 2873750349-

X3 6732534148-

X3 1594979449

X3 3706121949

X3 5454482849

PART II

THE GAT SOURCE LANGUAGE

1. CONSTANTS AND VARIABLES

In Example II (p. 12) the characters 2. and 3.141 are constants, i.e., quantities whose representation is only numeric; the characters D2 and X2 are variables, i.e., their representation is symbolic, and their values may vary during the solution of the problem. There are two classes of constants and variables: fixed point and floating point.

1.1. Fixed Point Constants

Fixed point constants in the source language contain from one to ten digits. The decimal point is always assumed to be immediately to the right of the rightmost digit (e.g., 25 means 25.0) but the point is always omitted. These constants may be positive, in which case the sign may be omitted, or they may be negative, in which case a "-" sign must precede the integer (e.g., 2, +100, -25, -10000, 0* are all fixed point constants). Leading (but not following) zeros may be omitted (e.g., 0005 is the same integer as 5 or 05, however 1 and 100 are not the same).

1.2. Fixed Point Variables

A fixed point variable consists of one of the letters "I", "J", or "K" followed by a subscript α (the form of allowable subscripts is treated on p. 26; for the time being α represents any allowable subscript). For example, I_{α} , J_{α} , K_{α} are fixed point variables. Fixed point variables may take on only the values allowed fixed point constants.

1.3. Floating Point Constants

There are two ways of writing floating point constants in the

* In this write-up 0 is considered a positive integer.

source language. In both forms, if the number is positive no sign need be written, but if it is negative a preceding "-" sign is written.

a) Without exponent. This form contains from one to eight digits and a decimal point, ".", which must be written. The decimal point may appear anywhere among the digits (e.g., 2.4, .42, 0., -.0005, -4. are all floating point constants).

b) With exponent. This form contains from one to eight digits with or without a decimal point (if the decimal point is immediately to the right of the rightmost digit it may be omitted but not otherwise) which form the mantissa, followed by the letter "E", followed by the exponent. The exponent is one or two digits in length preceded by its sign ("+" signs need not be written). The exponent gives the power of ten by which the mantissa is to be multiplied and must be ≥ -50 and ≤ 49 . For example, $25E2 (= 25.0 \times 10^2)$, $2.4E-4 (= 2.4 \times 10^{-4})$, $.4E40 (= 0.4 \times 10^{40})$, $-4E2 (= -4.0 \times 10^2)$, $-.04E-3 (= -.04 \times 10^{-3})$ are floating point constants. All non-zero floating point numbers, N , must be in the range $10^{-50} < |N| < 10^{49}$.

1.4. Floating Point Variables

A floating point variable consists of one of the letters "C", "D", "X", "Y", or "Z" followed by a subscript α . Floating point variables may take on only the values allowed floating point constants.

1.5. Alphabetic Constants

In addition to numeric constants, alphabetic constants may be written. A purpose of these constants is to permit the coder to print, during execution of the object program, pertinent comments which are meaningful to the coder. Any group of five or less characters enclosed by "\$" symbols is an alphabetic constant. Blank spaces are counted as a character for the

purpose of alphabetic constants. In this write-up a blank space will be represented by the symbol "□", even though nothing is punched on a card. Any character acceptable to the computer is allowed with the exception of the character "\$" which is used only to enclose the alphabetic constant. For example, \$ A □ - □ B \$ is an alphabetic constant of five characters. The characters of an alphabetic constant are translated into the corresponding computer two digit internal codes for the characters.

2. SUBSCRIPTS

2.1. Normal Subscription

All variables must be subscripted. All subscripts, with the exception of a specialized matrix form, must be one of the following three:

- a) a non-negative fixed point constant ≤ 999 ;
- b) a non-negative fixed point variable whose value ≤ 1999 ;
- c) an expression, β (p. 28), whose value is a non-negative fixed point number ≤ 1999 .

Therefore 2, 48, K5, KJ4, (J2-5), (K8*J45-I67) are all allowable subscripts, making K2, Y48, ZK5, KKJ4, C(J2-5), and I(K8*J45-I67) all allowable variables. (Again note that a fixed point number is always an integer.)

Variables with the same initial letter and consecutive subscripts are located consecutively in storage. Hence the variables X1, X2, X3, which might represent the components of a 3-dimensional vector, would be stored in this manner. A consecutive block of storage can be designated as a matrix stored by rows. For example, the matrix

$$\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix}$$

may be stored in X1 through X9, where the elements of the first row ($a_{1,1}$, $a_{1,2}$, $a_{1,3}$) are stored in X1, X2, and X3, the elements of the second row ($a_{2,1}$, $a_{2,2}$, $a_{2,3}$) in X4, X5, and X6, and the elements of the third row ($a_{3,1}$, $a_{3,2}$, $a_{3,3}$) in X7, X8, and X9.

2.2. Matrix Subscription

It is possible to refer to the elements of a two-dimensional array using the familiar matrix notation. An element is designated by writing $V(m,n)$ where V is the proper letter (C, D, I, J, K, X, Y, or Z), m is the row number and n is the column number of the element. Here m and n must be one of the following three:

- a) a positive (not zero) fixed point constant ≤ 999 ;
- b) a positive (not zero) fixed point variable whose value ≤ 1999 ;
- c) an expression β (p. 28) whose value is a positive non-zero fixed point number ≤ 1999 .

Thus in the above matrix $a_{1,2} = X(1,2) \equiv X2$ and $a_{3,1} = X(3,1) \equiv X7$.

3. ARITHMETIC

3.1. Arithmetic Operations

The following operations are allowed:

Operation	Source Language Symbols	Examples
a) Addition	+	$X4 + Z8$
b) Subtraction	-	$X4 - Z8$
c) Division	/	$X4 / Z8$
d) Multiplication	*	$X4 * Z8$; multiplication may <u>not</u> be written as juxtaposition.
e) Exponentiation	P	$K8P3 \equiv K8^3$ $X4PZ8 \equiv X4^{Z8}$
f) Absolute value	A	$AI4 \equiv I4 $
g) Negation	-	$-I4 \equiv$ negative of the value of $I4$ $X4P -.5 \equiv X4^{-.5}$

3.2. Arithmetic Expressions

Any properly parenthesized string of variable names, constants, and/or subroutine calls, separated by operation symbols, is an expression.

Parentheses are used in the same manner as in ordinary algebra to specify sequence of computation. Thus $(I4 + I6) - K8$ means $I4$ is added to $I6$, then $K8$ is subtracted from that result; $I8*(K9 - J2)$ means $J2$ is subtracted from $K9$, and that result is then multiplied by $I8$. Parenthesis nesting may not exceed 10.

Where the sequence of computation is specified by the following two rules, parentheses may be omitted. (Redundant parentheses are allowed.)

RULE: Within an expression the sequence of computation, if not otherwise specified by parentheses, is:

- a) subscription,
- b) absolute value and negation,
- c) exponentiation,
- d) multiplication and division,
- e) addition and subtraction.

That is, the arithmetic operations are performed according to their hierarchy, which is defined in the above list. Thus, subscription is said to be the operation of highest hierarchy, while addition and subtraction are the operations of lowest hierarchy and are also of equal hierarchy. (Note that any expression intended as a subscript must be parenthesized; e.g., $K(I^4 + J^2)$ without parentheses becomes $KI^4 + J^2$.)

Examples:

1) $A(I^4 - K^6)$ means K^6 subtracted from I^4 and the absolute value of the result is taken, while $AI^4 - K^6$ means K^6 is subtracted from the absolute value of I^4 .

2) $-I^2 + I^4$ means the negative of I^2 plus I^4 , while $-(I^2 + I^4)$ means the negative of the result of adding I^2 and I^4 .

3) $I^7 P I^2 + K^2$ means $I^7 I^2$ plus K^2 , $I^7 P - I^2 + K^2$ means $I^7^{-I^2}$ plus K^2 since negation appears before exponentiation on the above list.

4) $K^2 / I^2 - I^4$ means K^2 divided by I^2 and then I^4 is subtracted from the result, while $K^2 / (I^2 - I^4)$ means I^4 is subtracted from I^2 and the result is divided into K^2 .

5) $K^2 + I^3 * I^5$ means $K^2 + (I^3 * I^5)$.

6) $K^2 P^2 / J^7$ means $((K^2)^2) / J^7$.

- 7) $K5 * I6PI2$ means $K5 * ((I6)^{I2})$
 8) $AI^4 + J2 * K2$ means $|I^4| + (J2 * K2)$

RULE: Within an expression the sequence of computing when operations are of equal hierarchy is from left to right, if not otherwise specified by parentheses.

Examples:

- 1) $I^4 + I6 - K7 + J2 - I7$ means $((((I^4 + I6) - K7) + J2) - I7)$.
 2) $J2 * I7 / J4 / K5 * K2$ means $((((J2 \wedge I7) / J4) / K5) * K2)$.
 3) $I7 + K2 * J4 / I2 + K7 * I2$ means $(I7 + ((K2 * J4) / I2)) + (K7 * I2)$.
 4) $I6 + K2 * I8PI4 / I3 - K2P - J2 * K4 + I8$ means
 $((I6 + ((K2 * (I8^{I4})) / I3)) - ((K2^{-J2}) * K4)) + I8$

3.3. Mode of Arithmetic

Definitions of the terms "fixed point number" and "floating point number" have previously appeared (pp. 23, 24). The same terms also describe the arithmetic of a given number, i.e., the actual mechanics of calculation involved when that number is operated upon. If the arithmetic of all constants and variables in a given expression is the same, then the result after performing the operations in the expression has that same arithmetic. Thus the result of $I^4 + K7 * JI2$ is fixed point, and the result of $C9 / Z7 * .04E2$ is floating point. As a general rule, mixed arithmetic in an expression always leads to a floating point result. It is useful to know the sequence of conversions, however, to avoid misunderstandings. The procedure followed by the compiler is to maintain the arithmetic of the two arguments involved in a given operation if they have the same arithmetic, but to convert to floating point when an operation is encountered (according to the hierarchy

on p. 29) in which the arguments have differing arithmetics. Thus, in $Y4 = .6 + 1/I3$, since the division is performed first, and since both arguments for the division operation are fixed point numbers, the division is in fixed point. This result is converted to floating point for the addition to a floating point constant .6. If the above example had read $Y4 = .6 + 1./I3$, the value of I3 would have been converted to floating point, since the other argument of the division operation is now a floating point constant.

4. STATEMENTS

A statement to GAT corresponds to a sentence in the English language; that is, it conveys a complete thought to the compiler. The acceptable forms of GAT statements will be thoroughly illustrated below.

4.1. Substitution Statements

The form of a substitution statement is $V\alpha = \beta$ where $V\alpha$ is any variable and β any arithmetic expression. The value of $V\alpha$ is replaced by the value of β . Thus $K7 = J4 * I4$; after execution of this statement the value of $K7$ is equal to the product of $J4$ and $I4$. When a variable takes on a new value as a result of a substitution statement, the arithmetic of the expression on the right of the "=" symbol is converted to that of the variable on the left before substitution, if their arithmetics differ. For example, in the statement $Y4 = K2 + I4$, the quantity $K2 + I4$ is computed in fixed point arithmetic but is converted to floating point, i.e., "floated", before it is substituted into $Y4$. In the statement $I2 = K2 + Z8$, the quantity $K2 + Z8$ is computed in floating point but is converted to fixed point, i.e., "fixed", before being substituted in $I2$. See discussion of FIX subroutine, p. 63.

4.2. Control Statements

The normal sequence of program execution is from one statement to the next statement in the order in which they appear. If a statement is explicitly referred to by another statement it must have a unique non-zero number (≤ 999) which is called the statement number. If no reference is made to a statement, then the statement number may be omitted. Numbers need not be in order, nor do all numbers below a given number have to be used. Statements which transfer control to other statements by number are called control statements.

One way of varying the normal sequence of program execution is by the use of an unconditional GO statement. This statement is of the form

"GO TO μ " where μ may be:

- a) a constant,
- b) a variable,
- c) an expression,

all of whose values are non-zero fixed point numbers ≤ 999 . Thus "GO TO 5" means "go to statement number 5". "GO TO J6" means "go to the statement whose number is equal to the current value of J6". (The value of J6 may be changed during the execution of the program; hence when the statement "GO TO J6" is executed, control will transfer to one of several different places in the program, depending on the current value of J6. This serves as a variable "switch".)

Another method of varying this sequence is the use of a conditional GO statement. This statement is of the form "GO TO μ IF $\beta_1 \mathcal{R} \beta_2$ ", which means that the normal sequence of execution is altered if and only if β_1 stands in relation \mathcal{R} to β_2 . If this relation is not satisfied then computation proceeds to the next statement in the normal sequence. Here β_1 and β_2 may be any expressions, μ has the same form as for the unconditional GO statement discussed above, and $\beta_1 \mathcal{R} \beta_2$ may be:

- 1) $\beta_1 \cup \beta_2$ meaning $\beta_1 = \beta_2$
- 2) $\beta_1 \vee \beta_2$ meaning $\beta_1 > \beta_2$
- 3) $\beta_1 \supset \beta_2$ meaning $\beta_1 \geq \beta_2$
- 4) $\beta_1 \cap \beta_2$ meaning $\beta_1 < \beta_2$
- 5) $\beta_1 \mathcal{R} \beta_2$ meaning $\beta_1 \leq \beta_2$.

Thus "GO TO 2 IF K7 U 3" means "Go to statement number 2 if K7 equals 3, otherwise follow the normal sequence". "GO TO I5 IF Y8 + 5. V Z7P2" means "Go to the statement whose number is equal to the current value of I5 if $Y8 + 5$ is greater than $Z7^2$, otherwise follow the normal sequence".

A third method of varying the normal sequence of execution is by means of an iteration statement. In most programs there are usually groups of statements which are to be repeated many times, incrementing some variable each time. This repetition is called iteration and the statement controlling this is the iteration statement. The block of statements repeated in this manner define the scope of the iteration.

2: k, $V\alpha$, β_1 , β_2 , β_3 ,	iteration statement
4: ----	} scope of iteration

k: ----	

This iterative procedure halts when the variable reaches a certain pre-defined value. The iteration statement m: "k, $V\alpha$, β_1 , β_2 , β_3 , (all five commas must be written, particularly the last one) means "Initially set the value of $V\alpha$ equal to β_1 , then execute the block of statements starting with the statement following this one (if $k \neq m$), up to and including the statement numbered k, then repeat the execution of this block again and again, each time adding β_2 to $V\alpha$ until $V\alpha > \beta_3$. If $k = m$, the statement number of the iteration statement itself, the incrementing and testing contained in the iteration statement itself constitute the scope. The block of statements is executed if $V\alpha = \beta_3$ but not for $V\alpha > \beta_3$, hence if $(\beta_3 - \beta_1)$ is not an integral multiple of β_2 , the last time the block will be executed is for $(\beta_3 - \beta_2) < V\alpha < \beta_3$.

Here k is a statement number, $V\alpha$ any variable, and β_1 , β_2 , and β_3 any expressions. If β_2 is negative, the test for completion of the iteration will be $V\alpha \leq \beta_3$ instead of $V\alpha \geq \beta_3$.

Examples:

1) The program segment,

1: J2 = J9

2: 3, I7, 2, 1, K7,

3: J2 = J2*J9

computes the value of J9 raised to the $K7^{\text{th}}$ power ($K7 \geq 2$).

(Note that I7 is a dummy variable used as a counter.) The purpose of this example is to illustrate a variable upper limit; in actual practice it is more efficient to use the exponentiation operation, i.e., $J2 = J9^{K7}$.

2) The program segment,

Z1 = 0

Z2 = 1

1, K1, 1, 1, K2,

Z1 = Z1 + XK1

1: Z2 = Z2 * YK1

accumulates the sum of the K2 numbers X1, X2, ..., XK2,

and the product of the K2 numbers Y1, Y2, ..., YK2.

3) Given a vector X1, X2, ..., XK1 the following program segment

computes square root of the sum of the last J1 elements.

4: X0 = 0

2, I1, K1 - J1 + 1, 1, K1,

2: $X0 = X0 + X11$

$X0 = X0P.5$

- 4) To evaluate the polynomial $C_n X^n + c_{n-1} X^{n-1} + \dots + c_1 X + c_0$ use the formula $(\dots((c_n X + c_{n-1}) X + c_{n-2}) X + \dots + c_1) X + c_0$. Letting $Y1$ = value of the polynomial, $X1 = x$, $K1 = n$, and $CK1 = c_n$, $C(K1 - 1) = c_{n-1}$, ..., $C1 = c_1$, $C0 = c_0$, the following program segment will evaluate this polynomial:

$Y1 = 0.$

2, $K2, K1, -1, 0,$

2: $Y1 = X1 * Y1 + CK2$

- 5) x , the square root of the number y can be obtained by applying Newton's method to the equation $f(x) = x^2 - y = 0$. The general iterative procedure called Newton's method proceeds by calculating successive approximations to the root x of $f(x) = 0$ as follows:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}.$$

Applied to the equation $f(x) = x^2 - y$ this becomes

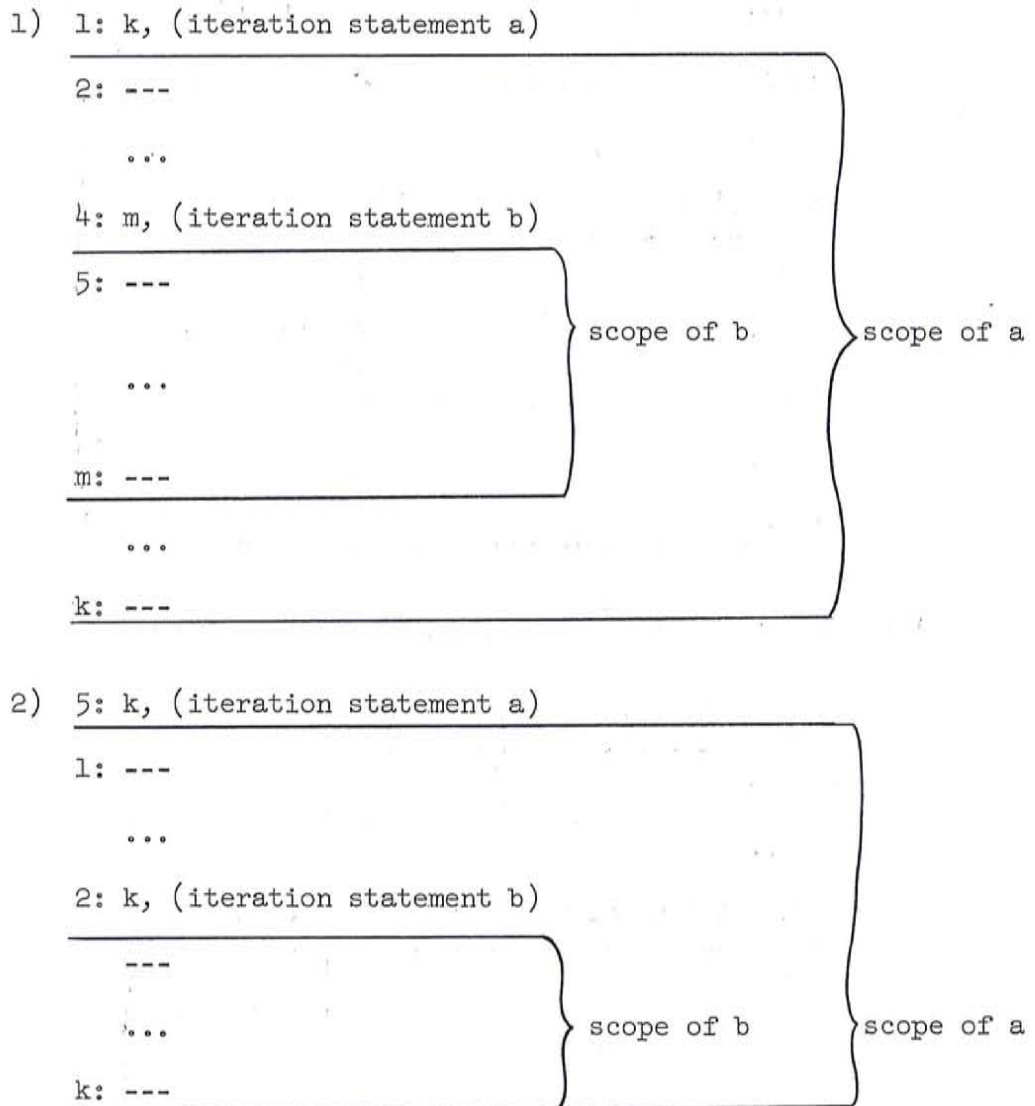
$$x_{i+1} = x_i - \frac{x_i^2 - y}{2x_i} = \frac{1}{2} \left(x_i + \frac{y}{x_i} \right).$$

The solution can be written as a single statement if the criterion for stopping the iteration is $|f(x)| < \epsilon$:

1: $1, X1, X0, - (X1P2 - Y1)/(2. * X1), X1 - A(X1P2 - Y1) + Y2.$

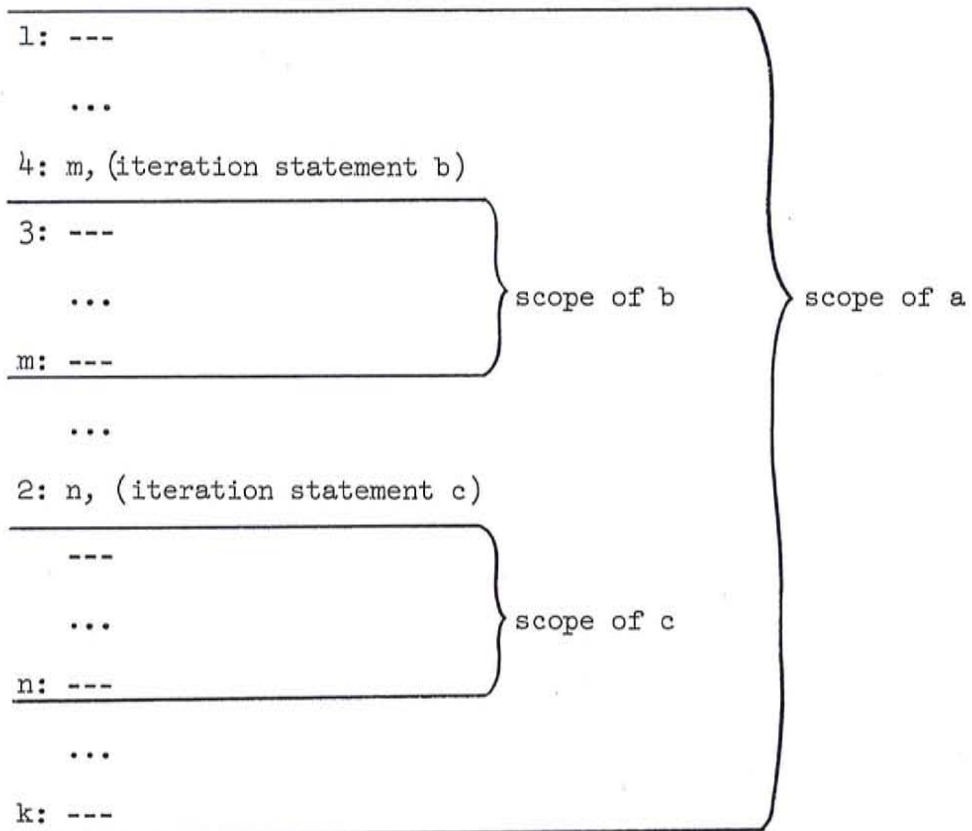
In the above statement, $Y1 = y$, $Y2 = \epsilon$, $X0$ is the initial estimate of x , and $X1$ becomes the desired answer at the termination of the iteration. (Note again that this is an illustrative problem; in actual practice the square root can be obtained simply as $X1 = Y1P.5$).

Iteration statements may contain within their scope other iteration statements. This is called nesting. If iteration statement b is in the scope of another iteration statement a, then the scope of b must be entirely within the scope of a. The following diagrams represent some valid configurations:

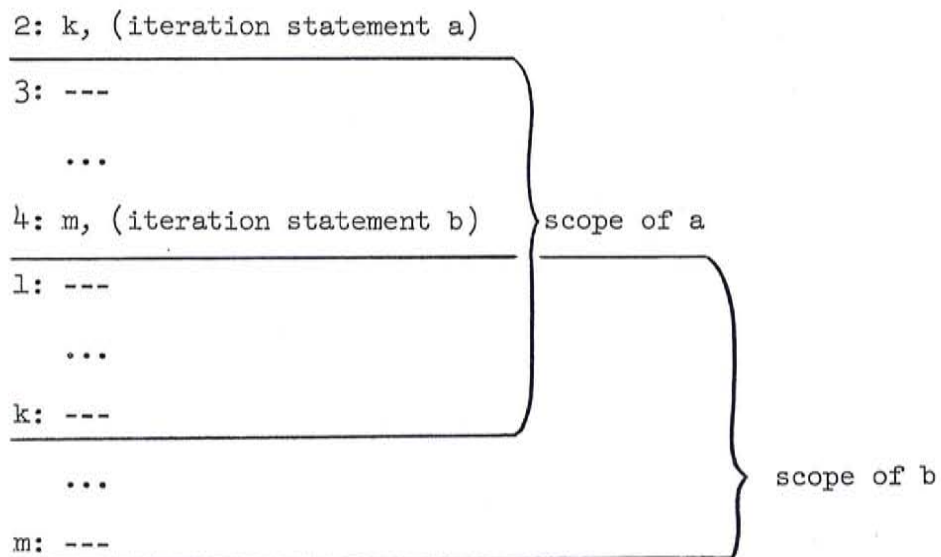


Note: Even though the scope of a and b both end on statement k, the iteration b is incremented and tested first, hence iteration b is completed before iteration a is incremented.

3) k, (iteration statement a)

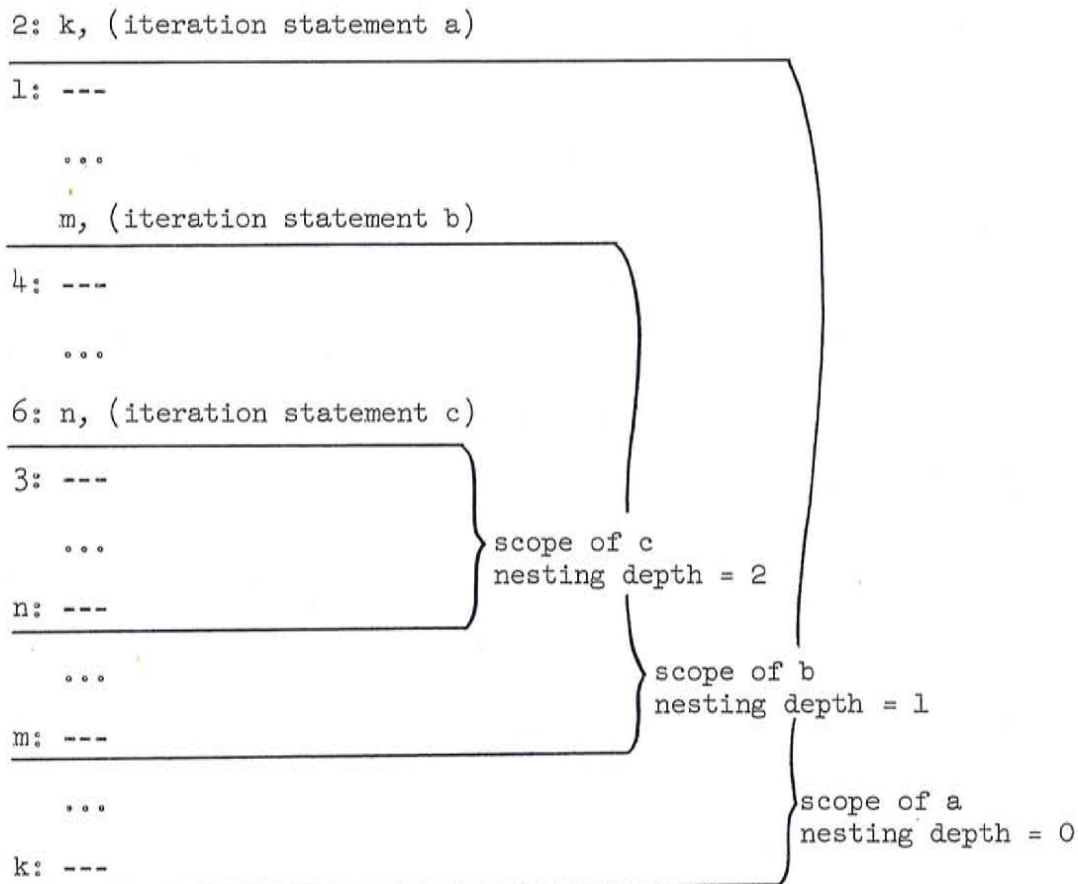


The following diagram represents an invalid configuration:



The nesting depth of an iteration statement is the number of iteration statements in whose scope it appears. The nesting depth of an iteration statement may not exceed eight.

For example,



4.3. Input-Output Statements

a) Input Statement

The object program usually needs additional data in order to perform its calculations. The execution of the input statement "READ" causes the next card(s) appearing in the problem deck to be read by the I.B.M. 533 card reader (p. 51 for data card formats). No other information is necessary to the READ statements since all data is fully identified on the card. Comments identify-

ing any particular READ statement may precede the word "READ" in that statement, but they have no actual effect on the program.

b) Output Statements

The form of the output statement used for printing numeric data on the I.B.M. 407 printer is " $T\lambda_1 T\lambda_2 \dots T\lambda_n$ " where $\lambda_1 \lambda_2 \dots \lambda_n$ may be any numeric constants or any variables (not expressions) and n , the number of constants and/or variables specified in the print statement, must be less than or equal to 16 ($n \leq 16$). For example, $TY4TK(2,8)T4$ will print the names $Y4$ and $K(2,8)$ [the name " $K(2,8)$ " is not printed but appears as " $K\alpha$ " where α is its numeric position in the K vector storage, as on p. 26] along with their values, and the fixed point constant 4 (a constant is usually printed merely as identification); $TZK8$, when $K8 = 4$, will print the name $Z4$ and its value.

To print alphabetic data the statement " $AT\lambda_1 T\lambda_2 \dots T\lambda_n$ " is used, where $\lambda_1 \lambda_2 \dots \lambda_n$ may be either fixed point variables (I, J or K) or alphabetic constants (p. 24) and $n \leq 16$. For example, $ATT4TJ15$ will print numeric equivalents for the names $I4$ and $J15$ and will print their values as if they were five-character alphabetic constants (see I.B.M. 650 manual G24-5000-0, p. 21); $AT\$NO\$SOT\$LUTIO\$T\$N\$\$\$$ will print "NO SOLUTION". (This illustrates the use of the "\$" notation as a means of printing a useful message to the coder during execution of the object program.) See p. 55 for specific output formats.

Either of the two previous output statements may specify regions to be printed rather than single variables. Writing $T\lambda_i \dots \lambda_j$ (the three dots must appear) will cause printing of all the names $\lambda_i, \lambda_{i+1}, \lambda_{i+2}$ through λ_{j-1}, λ_j inclusive with their associated values. λ_i and λ_j may be any numeric constants or any variables (not expressions), and i must be less than j . Each

region designated is regarded by the compiler as two single variables, therefore only eight (or less) regions may be specified in one T statement. Regions, single variables and constants may be mixed, but $(n + 2m)$ must be ≤ 16 , where n = number of single variables and/or constants and m = number of regions.

4.4. Executive Statements

The function of the following statements is to provide information to the compiler as to how to allocate storage for the object program, its variables, and its constants.

a) Statement Number Allocation

The statement

n IS HIGHEST STATEMENT NUMBER

causes reservation of n words (necessary to GAT during compilation) which merely occupy storage space in the computer during execution of the object program. Hence it is to the coder's advantage to make n no greater than the highest statement number actually used. However, every number $\leq n$ need not be used. n must be ≤ 999 .

b) Subroutine Allocation (p. 45)

Subroutines to be included when the object program is executed also occupy space in computer storage. GAT must therefore reserve a sufficient number of locations for these subroutines. The statement

m USED IN SUBROUTINES

provides a number, m , which must be ≤ 999 and must be \geq the total number of locations to be saved for subroutines. Appendix B gives the exact number of locations required by each subroutine available to GAT.

NOTE: The above two statements must appear if statement numbers and/or subroutines are used.

c) Allocation of Constants

Normally storage space is provided for 100 constants created by the source language (as opposed to constants read by the object program as data). Here-

after these constants will be referred to as absolute constants, or abcons.
 (The numbers 3.14, 4. are abcons, but 2. and -2. occupy only one location.)
If the coder knows that less than 100 abcons will be generated by the compiler,
 and if he knows that extra storage space is needed by his program, the state-
 ment

ℓ ABCONS

will cause reservation of ℓ locations for abcons. Here, the relationships
 $100 > \ell \geq$ the actual number of abcons generated must be satisfied.

d) Allocation of Variables

Appearing immediately after the above three statements must be the statement

DIMENSION $V_1(s_1, c_1, b_1) V_2(s_2, c_2, b_2) \dots V_i(s_i, c_i, b_i) \dots V_m(s_m, c_m, b_m)$

(the commas must appear, but not the dots). Each V_i is one of the variable
 letters: C, D, I, J, K, X, Y, or Z. Each variable letter used in the program
must be mentioned in the DIMENSION statement. If V_i is not to be used as
 a matrix (not necessarily square), c_i and b_i do not appear. Thus,

$V_i(s_i)$ is an array;

s_i is the highest subscript number in the V_i array;

$V_i(s_i, c_i, b_i)$ is a matrix included in the V_i array;

c_i is the number of columns in the V_i matrix;

b_i is the subscript number of $V_{(1,1)}$; $V_{(1,1)}$ is the base element
 of the V_i matrix, and $s_i > b_i$ (the base element may be any
 element within the V_i array);

also, s_i is a non-negative fixed point constant, and $s_i \leq 1999$;

c_i and b_i are non-negative, and may be either fixed point constants
 or fixed point variables but with a constant subscript (e.g., K2
 but not JK2).

(GAT assumes the number of rows in the V_i matrix to be equal to the largest integer less than $(s_i - b_i + 1)/c_i$.)

To illustrate:

```
DIMENSION J(25) K(40, 5, 15)
```

J is an array of 26 elements; J0 is its first element, J13 is its 14th element, and J25 is its 26th and last element.

K is a 41 element array ranging from K0 through K40. K15 is the base element of a 5-column matrix included in the K array. The K matrix is assumed to range from K15 through K39, so that K40 is outside the matrix. K15 may be referred to as K(1,1), K20 as K(2,1), K27 as K(3,3), and K39 as K(5,5).

```
DIMENSION X(100, K6, J2)
```

X is a 101-element array (X0 through X100), containing a matrix of K6 columns whose base element is XJ2. Both K6 and J2 may vary during execution of the object program; e.g., if initially K6 = 7 and J2 = 0, then X contains a 98 element matrix, $X(1,1) \equiv X0$, $X(7,5) \equiv X46$, and $X(14,7)$, the last element in the X matrix, $\equiv X97$. (X98, X99, and X100 are not within the matrix.) If K6 and J2 are later computed to be 3 and 98 respectively, then there are 3 elements in the X matrix, $X(1,1) \equiv X98$, $X(1,2) \equiv X99$, and $X(1,3) \equiv X100$. (Now X0 through X97 are not within the matrix.)

Thus the coder is able to vary the size of matrices during computation.

e) HALT

The statement

HALT

when encountered during execution, causes execution of the object program

to cease. The last instruction executed must be either HALT or READ.

f) END

A card containing "END" must be the last card of each source language deck. It indicates to the compiler that the object program is now completely compiled.

5. SUBROUTINES

Subroutines are subprograms, precoded in machine language, which perform frequently used functions. The set of these routines available to the GAT user is called the GAT Subroutine Library. A subroutine is incorporated in the object program by writing

$$\text{NNNNN. } (\beta_1, \beta_2, \beta_3, \dots, \beta_k)$$

(the period after NNNNN and the commas must be written), where: NNNNN is the 5(or less)-character name of the subroutine; $\beta_1, \beta_2, \beta_3, \dots, \beta_k$ are the arguments needed by the subroutine (k , the number of arguments, may vary from subroutine to subroutine) and may be any constants, any variables, or any expressions, subject to the limitations of the particular subroutine (see Appendix B for descriptions of library subroutines).

Subroutine calls appearing in conditional GO statements must be parenthesized:

$$\text{GO TO } \mu.\text{IF}(\text{NNNNN. } (\beta_1, \dots, \beta_k)) \text{ } \mathcal{R}(\text{NNNNN. } (\beta_1, \dots, \beta_k))$$

The term "NNNNN. ($\beta_1, \beta_2, \beta_3, \dots, \beta_k$)" may be thought of as meaning "the result of NNNNN" and the result is treated during computation as a variable; i.e., $Y1 = C3 + 1.3/\text{SIN } X1$.

Some subroutines may require as an argument a symbol for the location of a variable rather than the variable name. The letter L preceding the variable name is that symbol. (One of β_i being "LJ2" means "the location of J2".)

A subroutine may in itself be a complete statement, if one of the required arguments specifies where the result of the computation is to be stored. This is analogous to a substitution statement. (p.104, statement no. 2, Runge-Kutta subroutine.)

Object programs generally use some subroutines not explicitly called for in the source language. These may be:

- a) FIX
 - b) FLOAT
 - c) READ
 - d) T, AT (PRINT)
 - e) EXPONENTIATION
 - f) MATRIX SUBSCRIPTION
- needed for changes of
arithmetic in a statement

The coder should be careful to reserve enough room for these subroutines if they are needed (see m USED IN SUBROUTINES, p. 41).

It is imperative that the coder know enough about the operation of GAT to know which subroutines will be required to carry out various operations, e.g.,

Y1 = J1, (requires FLOAT subroutine);
K5 = J1P3, (requires exponentiation subroutine);
TJ7TY (1,5) (requires print subroutine and matrix subscription
subroutine).

An object program may not call for more than 16 distinct subroutines. (When a subroutine has two (or more) entries and both (or more) are used, each distinct entry used is considered a subroutine call.)

NOTE: The first statement of a source language program may not contain a call to a subroutine which needs an "L" argument; the first statement may not consist entirely of a subroutine call (other than READ, T or AT).

PART III
GAT INPUT-OUTPUT FORMATS

1. PREPARATION OF CARDS

1.1. Statement Cards

Statement cards have the following format:

Cols. 1-10 Blank

Cols. 11-15 Identification (or Blank)

Cols. 16-40 Blank

Cols. 41-43 Statement Number

Cols. 44-50 Blank

Cols. 51-80 The Statement

Statement numbers (right-justified in Cols. 41-43) may be punched as one-, two-, or three-digit integers, since leading zeros need not be punched.

If a statement cannot fit entirely on a single card, it may be continued on up to three succeeding cards of the same format: a total of 4 cards per statement is the allowable limit. The signal that a given card is not the last of a statement's cards is the punching of the letter "N" (Next) as the last character on that card.

The statement number need not be punched on the second, third, or fourth cards of a statement. If a number is punched on the first card, it may be omitted on the others, but if the statement number is punched on later cards it must be the same number as on the first card of the statement.

Except when appearing in an alphabetic constant, blanks in statements are always ignored by the compiler. Thus they may be used as desired for easier reading of the statements.

[illegible]

1.2. Data Cards

A set of data cards which is intended to be read by the object program as the result of a single "READ" statement is called a read group. The program will read cards and store the information as indicated on each card until an end-of-group character (* or L, see p. 51, 53) is encountered in column 75. After the contents of the card containing this character are stored, the program passes on to the next statement.

One of the two types of data cards is the numeric data card. The format of this data card is as follows:

Cols. 1-10	Value of 1st variable
Cols. 11-20	Value of 2nd variable
Cols. 21-30	Value of 3rd variable
Cols. 31-40	Value of 4th variable
Cols. 41-50	Value of 5th variable
Cols. 51-55	Alphanumeric name of 1st variable
Cols. 56-60	Alphanumeric name of 2nd variable
Cols. 61-65	Alphanumeric name of 3rd variable
Cols. 66-70	Alphanumeric name of 4th variable
Cols. 71-75	Alphanumeric name of 5th variable
Cols. 76-80	Not used

If the numeric data card is the last of a read group, punch an "*" in column 75. On this card values of up to 5 variables must be punched, right-justified, in the designated ten-column field. Leading zeros must be punched. Values of C, D, X, Y, and Z variables must be punched as ten-digit numbers, the first eight numbers being the mantissa and the last two the modified exponent; i.e.,

50 + exponent. On all numeric data cards the sign is punched in the tenth column of the designated field. Plus signs may be omitted; minus signs must be punched. The alphanumeric name of a variable must be punched left-justified in its designated field as a letter (C, D, I, J, K, X, Y, Z) followed by a single numeric subscript, i.e., C1, D0, but not CI2, Y(1,4). Whenever a name field is entirely blank, the corresponding value is completely ignored, and may also be blank. If the letters ETC are punched in a name field the name is assumed to be that of the last variable encountered, with the subscript increased by 1. For example, if the name fields in columns 51-75 of a numeric data card contained the names

YO, ETC, ETC, J17, ETC,

the variable names will be interpreted as

YO, Y1, Y2, J17, J18.

An ETC may also refer to the previous card within a read group. Thus, if the card following that above contained

ETC, ETC, ETC, ETC, ETC,

the names of that card would be interpreted as

J19, J20, J21, J22, J23.

Each name (including ETC) must be left-justified in its five-column name field.

The other type of data card is the alphabetic data card. No signs are punched on alphabetic data cards. The format of these cards is as follows:

Cols. 1-4	Blank or zero
Cols. 5-10	Numeric name of 1st variable
Cols. 11-14	Blank or zero
Cols. 15-20	Numeric name of 2nd variable
Cols. 21-24	Blank or zero
Cols. 25-30	Numeric name of 3rd variable
Cols. 31-34	Blank or zero
Cols. 35-40	Numeric name of 4th variable
Cols. 41-44	Blank or zero
Cols. 45-50	Numeric name of 5th variable
Cols. 51-55	Value of 1st alphanumeric variable
Cols. 56-60	Value of 2nd alphanumeric variable
Cols. 61-65	Value of 3rd alphanumeric variable
Cols. 66-70	Value of 4th alphanumeric variable
Cols. 71-74	Value of 5th alphanumeric variable
Col. 75	$\left\{ \begin{array}{l} \text{A if } \underline{\text{not}} \text{ the last card of a read group} \\ \text{L if the last card of a read group} \end{array} \right.$
Cols. 76-80	Not used

Note that the value of the 5th variable may contain only four alphanumeric characters. When this value is stored in the computer the two-digit code for "blank" (00) replaces "A" or "L" to make a full ten-digit word. The numeric names are of the form $\square\square\square\square$ tt nnnn, where \square represents a blank column, nnnn is the subscript of the variable and must be punched as a four-digit number including leading zeros, and tt represents variable name according to the following code (the zero must be punched):

PREPARATION OF CARDS

<u>tt</u>	<u>Variable</u>
01	C
02	D
03	I
04	J
05	K
06	X
07	Y
08	Z

Whenever a numeric name field is entirely blank, the corresponding alphanumeric value is completely ignored, and may also be blank. An ETC notation completely analogous in its effect to that described on p. 52 is available for the alphabetic card. This is accomplished by setting $tt = 10$ and $nnnn = 0000$ in the numeric name described above.

2. OUTPUT FORMATS

2.1. Listing

During compilation, the input statement cards are listed and if an error is detected in a particular statement an error indication appears on the following line. See Appendix C for list of errors detected during compilation. Only one error is detected per statement. An example of a listing containing an error indication is:

```
400 USED IN SUBROUTINES
8 IS HIGHEST STATEMENT NUMBER
DIMENSION I(1) J(1) N
X(100, K1, 1) Y(100, J1, 1) N
Z(100, J1, 1)
1  I1, J1, K1, X, Y, READ
2  6, IO, 1, 1, I1,
3  6, JO, 1, 1, J1,
4  Z(IO, JO) = 0
5  6, KO, 1, 1, K1,
5  STATEMENT - ERROR NO. 02
6  Z(IO, JO) = X(IO, KO) * Y(KO, JO) N
    Z(IO, JO)
6  STATEMENT - ERROR NO. 02
    HALT
    END
```

Error 02 occurred because K was not defined in the DIMENSION statement.

Listed data (with the exception of messages using the \$ notation, which are listed in the above format) appears as follows:

C1	4000000051	X1	1000000051				
X1	1000000052	X2	2000000052			I1	3
X3	3000000052	X4	4000000052			J1	5
X5	5000000052	X6	6000000052			K1	2
Y1	1000000051	ETC	2000000051	ETC	3000000051	ETC	4000000051
						ETC	5000000051

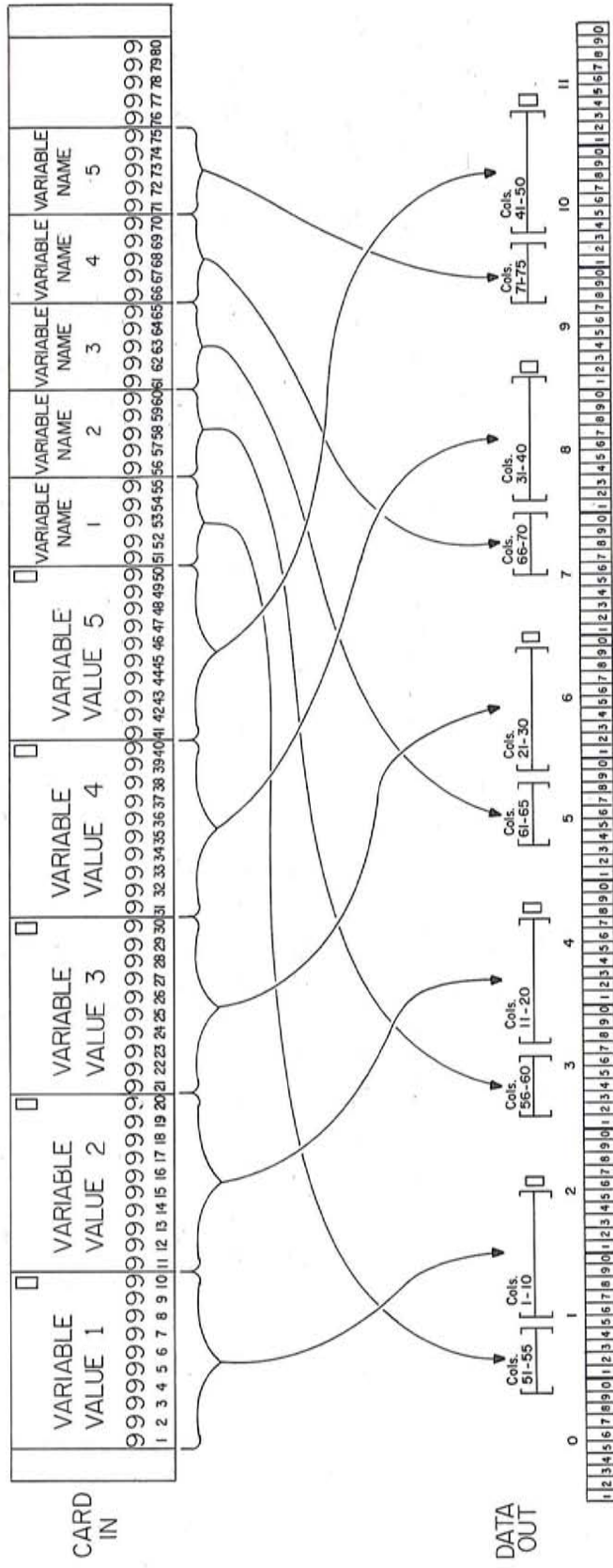
The first name, C1, corresponds to the name in columns 51-55 on input data; the value of this variable corresponds to columns 1-10, etc.

Listing of an object program or a dump appears in the form shown on p. 57.

2.2. Card Output

After a successful compilation the object program may be punched in the 5-instructions-per-card format described on p. 50. Following punching of the object program, tables necessary for its execution are punched in the same format.

Statement Number	Memory Loc. A	Contents of Loc. A		Memory Loc. B	Contents of Loc. B		Memory Loc. C	Contents of Loc. C		Memory Loc. D	Contents of Loc. D		Memory Loc. E	Contents of Loc. E	
2	0043	0000011990+		1990	8802080024+		0208	2119500166+		0044	0000020166+		0166	6000390168+	
2	0168	2100390216+		0216	0002330183+		0233	6100410218+		0218	3200390172+		0172	2100390266+	
2	0266	6700410268+		0268	2019500316+		0316	6000390318+		0318	3319500222+		0222	3200640272+	
3	0272	2119500366+		0366	6000390368+		0368	3319500322+		0322	4601670183+		0045	0000030183+	
3	0183	6000370418+		0418	6900390416+		0416	8801880020+		0188	2119500466+		0466	6000370468+	
3	0468	2119600516+		0516	8801840025+		0184	3900660372+		0372	3919500422+		0422	3200650472+	
3	0472	2119500566+		0566	6000370518+		0518	6900390616+		0616	8802380020+		0238	3200390522+	
4	0522	3419500572+		0572	2100410666+		0666	0001670233+		0046	0000040167+		0167	6000680568+	
4	0568	2119600716+		0716	6000670618+		0618	2119610766+		0766	8802340026+		0234	2119500816+	
100	0047	0000050816+		0816	0119490217+		0068	0000010004+		0067	0000060001+		0066	2302585151+	
100	0065	1000000051+		0064	1000000046+		0063	0000000005+		0000	0100000000+		0001	6300020036+	
100	0002	6400008010+		0003	6900008010+		0004	7100008010+		0005	7200008010+		0006	8700020038+	
100	0007	8800008010+		0008	8900020040+		0009	0000210042+		0010	0001000063+		0011	0000000000+	
100	0012	0000000000+		0013	0000000000+		0014	0000000000+		0015	0000000000+		0016	0000000000+	
100	0017	0000000000+		0018	0000000000+		0019	0000001130+		0020	6665877700+		0021	8765877700+	
100	0022	0000000000+		0023	0000000000+		0024	7965616400+		0025	7376679190+		0026	7577756368+	
100	0027	0000000000+		0028	0000000000+		0029	0000000000+		0030	0000000000+		0031	0000000000+	
100	0032	0000000000+		0033	0000000000+		0034	0000000000+		0035	0000000000+		0042	0000008000+	



This illustrates the difference between the appearances of input data (on cards) and printed data. Printing allows placing the value of the variable to the right of its name and makes for easier reading. (On alphabetic data, the numeric name of the variable appears to the right of its alphabetic value.)

APPENDIX A - SUMMARY OF THE LANGUAGE

1. Types of Constants

- a. fixed point - 1 to 10 digits.(p. 23)
- b. floating point - 1 to 8 digits with decimal point; 1 to 8 digit mantissa with or without a decimal point, followed by E, followed by a 1 or 2-digit exponent. (p. 23)
- c. alphabetic - 1 to 5 characters including blanks (\$ is not a valid character in an alphabetic constant since it is used as a delimiter). (p. 24)

2. Types of Variables

- a. fixed point - I, J or K, followed by a subscript. (p. 23)
- b. floating point - C, D, X, Y or Z followed by a subscript. (p. 24)

3. Types of Subscripts

- a. constant - non-negative, fixed point, ≤ 999 . (p. 26)
- b. variable - non-negative, fixed point, ≤ 1999 . (p. 26)
- c. expression - parenthesized; value is non-negative, fixed point, ≤ 1999 . (p. 26)
- d. matrix - $V(m,n)$; m and n may be any of 3.a, 3.b, or 3.c. (p. 27)

4. Arithmetic Operations

+, -, *, /, P, A and negation. (p. 28)

5. Expressions (p. 28)

An expression is any combination of constants, variables, subroutines, arithmetic operations, and parentheses.

6. Hierarchy of Operations (p. 29)

- a. subscription
- b. absolute value and negation
- c. exponentiation
- d. multiplication and division
- e. addition and subtraction

7. Mode of Arithmetic (p. 30)

- a. fixed point if all operands are fixed point.
- b. floating point if any operand is floating point.

8. Substitution Statement (p. 32)

$V_{\alpha} = \beta$ V_{α} is any variable; β is any expression; β converted to arithmetic of V_{α} before substitution.

9. Unconditional GO Statement (p. 33)

GO TO μ : μ is non-zero, fixed point, and ≤ 999 .

10. Conditional GO Statement (p. 33)

GO TO μ IF $\beta_1 \mathcal{R} \beta_2$: μ is non-zero, fixed point, and ≤ 999 ; β_1 and β_2 are any expressions; \mathcal{R} is

- u for =,
- V for >,
- W for \geq ,
- Q for <,
- R for \leq .

11. Iteration Statement (p. 34)

$k, V_{\alpha}, \beta_1, \beta_2, \beta_3$, - do through statement number k , starting with $V_{\alpha} = \beta_1$, in steps of β_2 , through just less than or equal to β_3 ; k is fixed point constant ≤ 999 ; V_{α} is any variable; $\beta_1, \beta_2, \beta_3$ are any expression; if β_2 is negative, do through V_{α} just greater than or equal to β_3 ; nesting depth ≤ 8 .

12. Input Statement (p. 39)

READ - preceded by suitable comments if desired; data cards read until * or L in col. 75.

13. Output Statements

- a. numeric - $T\lambda_1 T\lambda_2 \dots T\lambda_n$ (single variable) or $T\lambda_j \dots T\lambda_k$ (regional); λ_i is any constant or variable; $n + 2m \leq 16$ where m is number of regions. (p. 40)
- b. alphabetic - $AT\lambda_1 \dots T\lambda_n$ or $AT\lambda_j \dots \lambda_k$; λ_i any fixed point constant or variable (value appears as alphabetic characters when printed); $n + 2m \leq 16$, m = number of regions.

14. Executive Statements

- a. m USED IN SUBROUTINES (p. 41)

m is fixed point constant ≤ 999 ; $m \geq$ total number of locations needed by subroutines; appears previous to DIMENSION statement.

b. n IS HIGHEST STATEMENT NUMBER (p. 41)

n is fixed point constant ≤ 999 ; $n \geq$ highest statement number used; appears previous to DIMENSION statement.

c. ℓ ABCONS (p. 42) - ℓ is fixed point constant ≤ 99 ; $\ell \geq$ number of generated abcons; 100 locations reserved if this statement omitted; appears previous to DIMENSION statement.d. DIMENSION $V_1 (s_1, c_1, b_1) \dots V_k (s_k, c_k, b_k)$ (p. 42)

V is C, D, I, J, K, X, Y and/or Z; s_i is fixed point constant ≤ 999 and is maximum subscript; c_i is fixed point constant or variable (constant subscript) and is number of columns in V matrix; b_i is fixed point constant or variable (constant subscript) and is subscript of base element of V matrix.

e. HALT (p. 43) - terminates execution.

f. END (p. 44) - last card of source language deck

15. Subroutines (p. 45)

NNNN. $(\beta_1, \beta_2, \dots, \beta_k)$ - fixed point result if leftmost N is "X";

LV_α means "loc. of V_α ".

APPENDIX B - SUBROUTINE LIBRARY

1. FIX SUBROUTINE: 9 locations, 1 calling name.

Calling name: XFIX. (V)

This subroutine converts the floating point variable V to a fixed point number ≤ 8 significant digits. If V contains a fractional part it will be truncated without rounding. (E.g., 1576000052 becomes 0000000015.)

Error No.: 1 if $V \geq 10^9$

Note: This subroutine is called for whenever a statement calls for the substitution of a floating point value into a fixed point variable.

2. FLOAT SUBROUTINE: 17 locations, 1 calling name.

Calling name: FLOAT. (V)

This subroutine converts the fixed point number V to floating point.

Error No.: 2 if $|V| \geq 10^9$

Note: This subroutine is called for whenever mixed arithmetic is encountered (p. 30) or a fixed point value is to be substituted into a floating point variable.

3. READ SUBROUTINE: 89 locations, 1 calling name.

Calling name: READ. (no arguments required)

This subroutine reads one group of alphabetic or numeric data cards (p. 51).

Error No.: 3 if one of the variables on a card read in was not defined in the DIMENSION statement.

Note: This subroutine is called for whenever a READ statement is written in the source language.

4. PRINT SUBROUTINE: 161 locations, 2 calling names.

Calling name 1: NPRNT. ($\lambda_1, \lambda_2, \dots, \lambda_k$)

This subroutine prints numeric data of the format previously described. (p. 56)

Calling name 2: APRNT. ($\lambda_1, \lambda_2, \dots, \lambda_k$)

This subroutine prints alphabetic data of the format previously described. (p. 56)

Error No.: None

Note: This subroutine is called for whenever an output statement (T or AT) is written in the source language. The λ 's are the numeric codes for the desired variable names. (p. 40)

5. MATRIX SUBSCRIPTION SUBROUTINE: 20 locations, 1 calling name.

Calling name: XMATX. (V_1 , V_2 , V_3)

This subroutine obtains the desired element of a matrix where:
 V_1 is the row index (m), V_2 is the column index (n), and V_3 is the code number for the variable letter indicated (p. 54). All V's are non-negative fixed point constants.

Error No.: None

Note: This subroutine is called for whenever matrix subscription notation (p. 27) is used in the source language.

6. GENERAL EXPONENTIATION SUBROUTINE: 168 locations, 3 calling names.

Calling name 1: FEXP. (V_1 , V_2)

This subroutine computes $V_1^{V_2}$ (V_1 to the V_2 power) when V_1 and V_2 are both floating point variables. The result is obtained by finding $\log_{10} V_1$ (see subroutine 10, below), then computing $10^{V_2 \log_{10} V_1}$ (see subroutine 9, below). The special case where $V_2 = .5$ is computed directly by Newton's method (see subroutine 7, below).

Calling name 2: XEXP. (V_1 , V_2)

This subroutine computes $V_1^{V_2}$ when V_2 is a fixed point variable. The method used is binary product expansion. (E.g., $V_1^{13} = V_1 \cdot V_1^4 \cdot V_1^8$)

Calling name 3: GLOG. (V)

This subroutine computes $\log_{10} V$ as in subroutine 10, below.

Error No.: 6 for 0^0 ; 7 for $V < 0$ in \sqrt{V} ; 9 for $-51 \geq V \leq 50$ in 10^V ;
10 for $V \leq 0$ in $\log_{10} V$.

Note: This subroutine is called for whenever P is used in the source language (p. 28). The arguments, V_1 and V_2 , are expected by the subroutine to be in the upper accumulator and the distributor. If P is used in the source language, \log_{10} and square root can be obtained with no additional storage by using GLOG. and $V_1P.5$ instead of LOG 10. and SQRT.

NOTE THAT NONE OF THE ABOVE SUBROUTINES NEED BE EXPLICITLY
CALLED IN THE SOURCE LANGUAGE IN ORDER TO GENERATE A SUB-
ROUTINE CALL. GREAT CARE MUST BE TAKEN BY THE CODER,
THEREFORE, IN CALCULATING m OF m USED IN SUBROUTINES.

7. SQUARE ROOT SUBROUTINE: 36 locations, 1 calling name.

Calling name: SQRT. (V)

This subroutine computes by Newton's method the square root of the floating point variable V , where $0 \leq \sqrt{V} < 10^{25}$.

Range of Argument: $0 \leq V \leq 10^{50}$.

Error No.: 7 if $V < 0$.

8. ARCTANGENT SUBROUTINE: 65 locations, 1 calling name.

Calling name: ARTAN. (V)

This subroutine computes the arctangent of the floating point variable V by evaluating Hastings' 15th degree polynomial. If $|V| \geq 1$, $\tan^{-1} V$ is obtained by computing $\tan^{-1} 1/V \pm \pi/2$. $-\pi/2 < \tan^{-1} V < \pi/2$.

Range of Argument: $10^{-51} < V < 10^{50}$.

Error No.: None

9. 10^X SUBROUTINE: 35 locations, 1 calling name.

Calling name: 10EXP. (V)

This subroutine raises 10 to the floating point power V by evaluating Hastings' 7th degree polynomial. $10^{-5} < 10^V < 10^{50}$.

Range of Argument: $-51 < V < 50$.

Error No.: 9 if $-51 \geq V \geq 50$.

10. LOG (BASE 10) SUBROUTINE: 44 locations, 1 calling name.

Calling name: LOG10. (V)

This subroutine computes \log_{10} of floating point variable V by evaluating Hastings' 9th degree polynomial; $-51 < \log_{10} V < 50$.

Range of Argument: $0 < V < 10^{50}$.

Error No.: 10 if $V \leq 0$.

11. SINE-COSINE SUBROUTINE: 56 locations, 2 calling names.

Calling name 1: SIN. (V)

This subroutine computes the sine of the floating point variable V (in radians) by evaluating Hastings' 9th degree polynomial.

Calling name 2: COS. (V)

This subroutine computes $\cos V = \sin(V + \pi/2)$ as in SIN. above.

Range of Argument: $|V| < 10^9$. The argument V^1 used in the polynomial evaluation is reduced to the range $-\pi/2 \leq V^1 \leq \pi/2$ by subtracting multiples of 2π from the original argument V. Therefore large values of V will result in arguments V^1 with fewer significant figures. Note that if the argument is very large, a large amount of computing time will be used in the process. The coder is advised to formulate his problem so that these arguments stay with reasonable bounds.

Error No.: 11 if $|V| > 10^9$.

12. RUNGE-KUTTA-GILL DIFFERENTIAL EQUATION SUBROUTINE: 78 locations,
1 calling name.

Calling name: RKSUB. (n, LV, r)

Given the set of n simultaneous, first-order differential equations

$$\begin{aligned} y_1' &= f_1 (y_1, \dots, y_n) \\ y_2' &= f_2 (y_1, \dots, y_n) \\ &\cdot \\ &\cdot \\ &\cdot \\ y_n' &= f_n (y_1, \dots, y_n) \end{aligned}$$

with initial conditions at $t = t_0$

$$y_1 = y_1 (t_0), y_2 = y_2 (t_0), \dots, y_n = y_n (t_0)$$

this routine computes

$$y_1 = y_1 (t_0 + k), y_2 = y_2 (t_0 + k), \dots, y_n = y_n (t_0 + k)$$

by the Runge-Kutta-Gill method. This subroutine enters a program, called the auxiliary, which must be written by the user and which must compute the numbers

$$\begin{aligned} k_1 &= h f_1 (y_1, \dots, y_n) , \\ k_2 &= h f_2 (y_1, \dots, y_n) , \\ &\cdot \\ &\cdot \\ &\cdot \\ k_n &= h f_n (y_1, \dots, y_n) . \end{aligned}$$

The subroutine parameter n specifies the number of equations; LV specifies the location of the first word of the data storage block which contains $3n$ words, the first n words of data storage containing y_1, y_2, \dots, y_n , the second n words containing k_1, k_2, \dots, k_n , and the last n words being used as temporary storage; r is the statement number of the first statement of the auxiliary.

Before entry to this subroutine the user must put the initial conditions into the first n words of the data storage. The auxiliary must use the variables in the first n words of the data storage to compute k_1, k_2, \dots, k_n and it places these results into the second n words of the data storage. The subroutine places the result of the integration, namely,

$$y_1 = y_1(t_0 + h), y_2 = y_2(t_0 + h), \dots, y_n = y_n(t_0 + h)$$

in the first n words of the data storage, overwriting the initial conditions placed there by the user; hence, these results can serve as initial conditions for a second integration step, etc.

If the differential equations depend explicitly on t then the user should include as one of the n equations for k the equation

$$k_i = h,$$

with the initial condition

$$y_i(t_0) = t_0,$$

and the auxiliary should use y_i as the value of t (i.e. $y_i = t$) in computing any of the h 's which depend explicitly on t .

Error No.: None

ERROR NUMBERS DURING COMPILATION

- 01 A floating point subscript has been used.
- 02 A variable letter has been used that did not appear in the DIMENSION statement or a numeric subscript larger than the number of variables reserved in the DIMENSION statement.
- 03 The actual (or implied) parentheses nesting exceeded 10.
- 04 The subroutine entry table capacity of 16 has been exceeded.
- 05 The compiled program has exceeded the available storage.
- 06 Floating point subscript, statement number (in GO TO), or exponent (in floating point constant).
- 07 An abcon with more than 10 digits or alphabetic constant with more than 5 characters (or missing \$).
- 08 A fixed subscript, or statement number with more than 3 digits, or exponent with more than 2 digits (in floating point constant).
- 09 Statement longer than 120 characters (i.e. 4 statement cards).
- 10 There are statement numbers larger than the one designated as largest.
- 13 More left parentheses than right.
- 14 More than 100 abcons generated.
- 16 Constant in an executive statement exceeds allowable limit.
- 17 Improper formation of executive statement or improper sequence.
- 21 Exponent of a floating point constant out of range, i.e., < -50 or ≥ 50 .

- 25 Improper statement formation; missing operation, parenthesis, etc.
- 27 Improper combination of operation symbols.
- 29 Improper formation of matrix subscript, output statement, or use of "L".
- 30 One of first 4 commas in iteration statement missing.
- 31 Iteration statement nesting depth > 8, improper nesting, or end of program without terminating all iterations.
- 32 More than 16 arguments in an output statement or more than 17 arguments in a subroutine call.
- 33 More right parentheses than left.
- 99 Illegal character or combination of characters in a statement.

SUBROUTINE ERROR NUMBERS

- 01 XFIX. (V), if $V \geq 10^9$.
- 02 FLOAT. (V), if $|V| \geq 10^9$.
- 03 READ., if variable on card not defined in DIMENSION statement.
- 06 FEXP. and XEXP., if attempting 0^0 .
- 07 FEXP. and SQRT. if $V < 0$ in \sqrt{V} .
- 09 FEXP., XEXP., and LOEXP., if $-51 \geq V \geq 50$ in 10^V .
- 10 FEXP., GLOG., and LOG10., if $V \leq 0$ in $\log_{10} V$.
- 11 SIN. (V) and COS. (V), if $|V| > 10^9$.

EXAMPLE 1

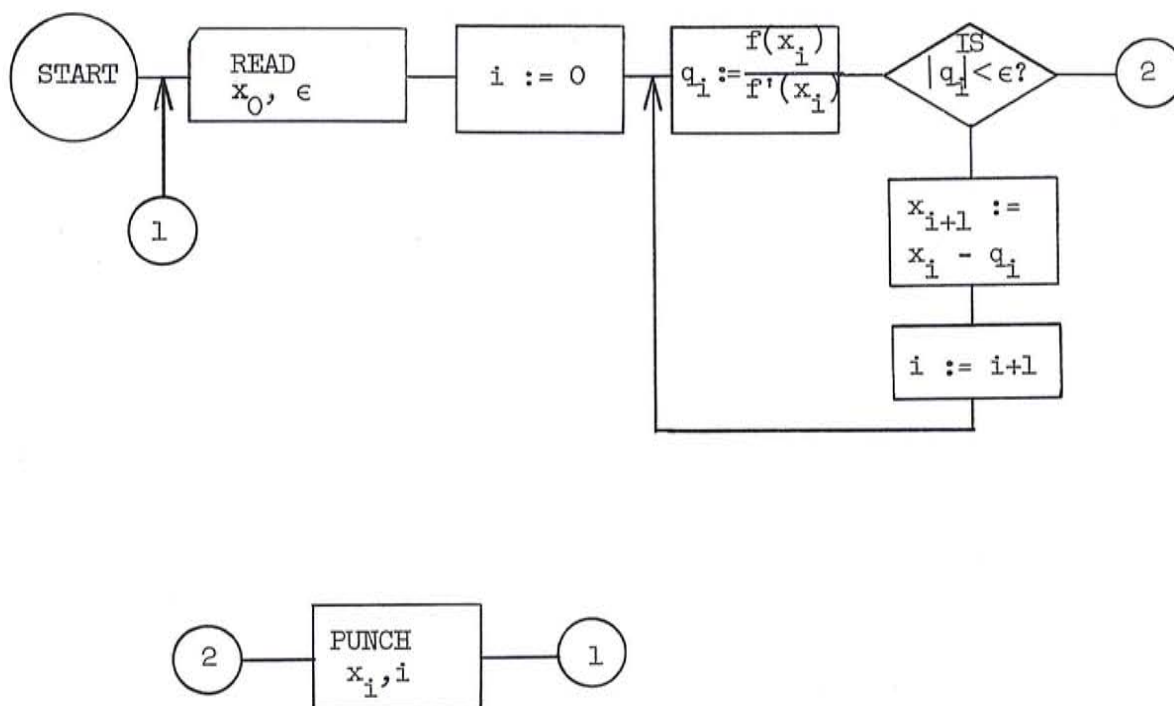
Problem: Solve the equation $f(x) = 0$ (where f is a differentiable function) by Newton's Method, allowing an arbitrary initial value x_0 and an arbitrary tolerance ϵ .

Analysis: Newton's Method consists of choosing an initial guess $x = x_0$ and successively improving it by means of the iterative formula:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} .$$

If we let $q_i = \frac{f(x_i)}{f'(x_i)}$, we may use as a criterion for stopping the iteration the condition

$$|q_i| = |x_i - x_{i+1}| < \epsilon .$$

Flow Diagram of Solution Procedure:

Rough Program: $[f(x) = 5x^3 + 4x^2 + 6x - 7 = 0]$

```

1: Read  $x_0, \epsilon$ 
2:  $i = 0$ 
3:  $q = (5x^3 + 4x^2 + 6x - 7) / (15x^2 + 8x + 6)$ 
4: Go to 8 if  $|q| < \epsilon$ 
5:  $x = x - q$ 
6:  $i = i + 1$ 
7: Go to 3
8: Punch  $x, i$ 
9: Go to 1

```

We make the following storage assignments:

```

i: I1
x: X0
q: D1
 $\epsilon$ : D2

```

Source Language Program:

```

250 USED IN SUBROUTINES
9 IS HIGHEST STATEMENT NUMBER
DIMENSION I(1) X(0) D(2)
1: X0, EPSILON READ
2: I1 = 0
3: D1 = (5.*X0*X0*X0 + 4.*X0*X0 + 6.*X0 - 7.) /      N
          (15.*X0*X0 + 8.*X0 + 6.)
4: Go to 8 IF AD1 Q D2
5: X0 = X0 - D1
6: I1 = I1 + 1
7: GO TO 3

```

```

8: TXO TIL
9: GO TO 1
END

```

NOTE: To produce a more efficient and accurate program, statement 3 could be replaced by:

```

3: D1 = (((5.*XO + 4.) *XO + 6.) *XO - 7.)/      N
      ((15.*XO + 8.) *XO + 6)

```

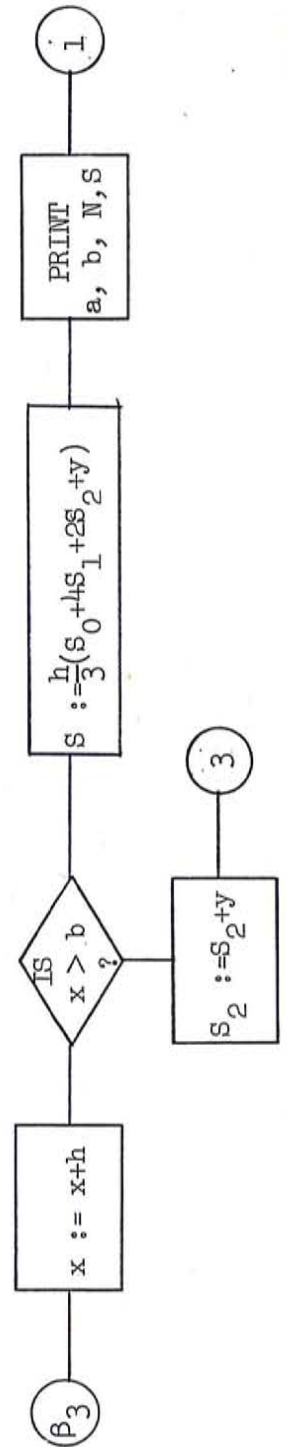
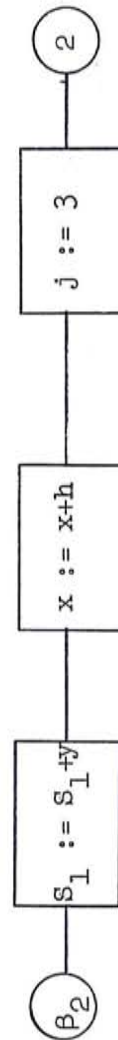
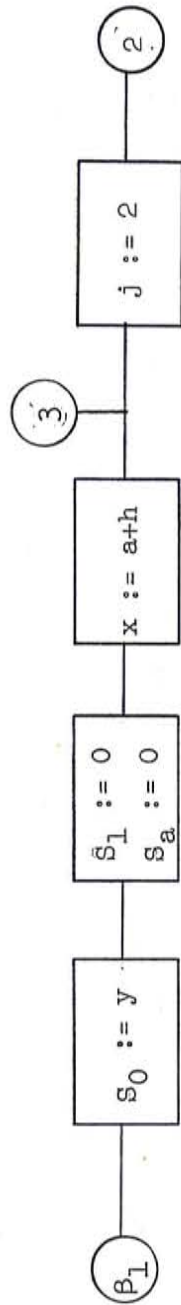
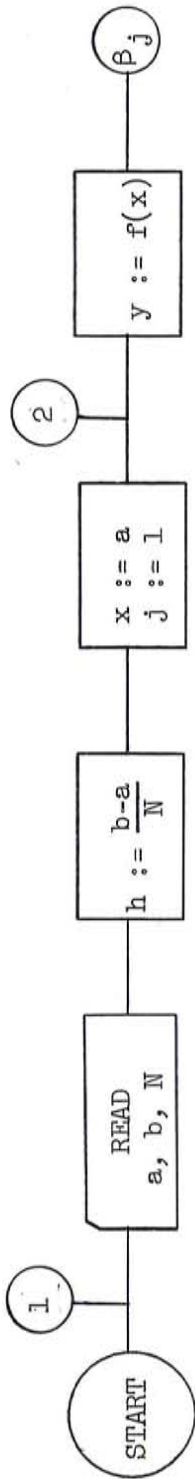
Example 2

Problem: To approximate $\int_a^b f(x)dx$ by Simpson's Rule for an arbitrary interval $[a,b]$ using N equal subintervals (where N is an arbitrary even integer).

Analysis: By Simpson's Rule,

$$\int_a^b f(x)dx \approx \frac{b-a}{3N} (y_0 + 4y_1 + 2y_2 + 4y_3 + \dots + 4y_{N-1} + y_N),$$

where $y_i = f(x_i)$, and $a = x_0, x_1, \dots, x_N = b$ are the partition points of the interval $[a,b]$. By changing only the statement(s) needed to compute $f(x)$ (of "a" in the flow diagram below), this program may be used to approximate the integral for any reasonable function $f(x)$. The program below is written for $f(x) = x^3$.



We make the storage assignments:

a:	C1	x:	X1	S ₂ :	Z2
b:	C2	y:	Y1	S:	Z3
N:	J0	S ₀ :	Z0	h:	Z4
j:	J1	S ₁ :	Z1		

Rough Program: $[f(x)] = x^3$

Source Language Program:

250 USED IN SUBROUTINES

23 IS HIGHEST STATEMENT NUMBER

DIMENSION C(2) J(1) X(1) N

Y(1) Z(4)

1: Read a, b, N

1: A, B, N READ

2: $h = (b-a)/N$

2: $Z4 = (C2-C1)/J0$

3: $x = a$

3: $X1 = C1$

4: $j = 7$

4: $J1 = 7$

5: $y = x^3$

5: $Y1 = X1*X1*X1$

6: Go to j

6: GO TO J1

7: $S_0 = y$

7: $Z0 = Y1$

8: $S_1 = 0$

8: $Z1 = 0.$

9: $S_2 = 0$

9: $Z2 = 0.$

10: $x = a+h$

10: $X1 = C1+Z4$

11: $j = 13$

11: $J1 = 13$

12: Go to 5

12: GO TO 5

13: $S_1 = S_1+y$

13: $Z1 = Z1+Y1$

14: $x = x+h$

14: $X1 = X1+Z4$

15: $j = 17$

15: $J1 = 17$

16: Go to 5

16: GO TO 5

17: x = x+h	17: X1 = X1+Z4
18: Go to 21 if x > b	18: GO TO 21 IF X1VC2
19: $S_2 = S_2 + y$	19: Z2 = Z2+Y1
20: Go to 11	20: GO TO 11
21: $S = h(S_0 + 4S_1 + 2S_2 + y)/3$	21: Z3 = Z4*(Z0+4.*Z1+2.*Z2+Y1)/3.
22: Punch a, b, N, S	22: TC1 TC2 TJO TZ3
23: Go to 1	23: GO TO 1
	END

NOTE: We could approximate the integral $\int_a^b \sin(3x^2)dx$ by simply replacing statement 5 by:

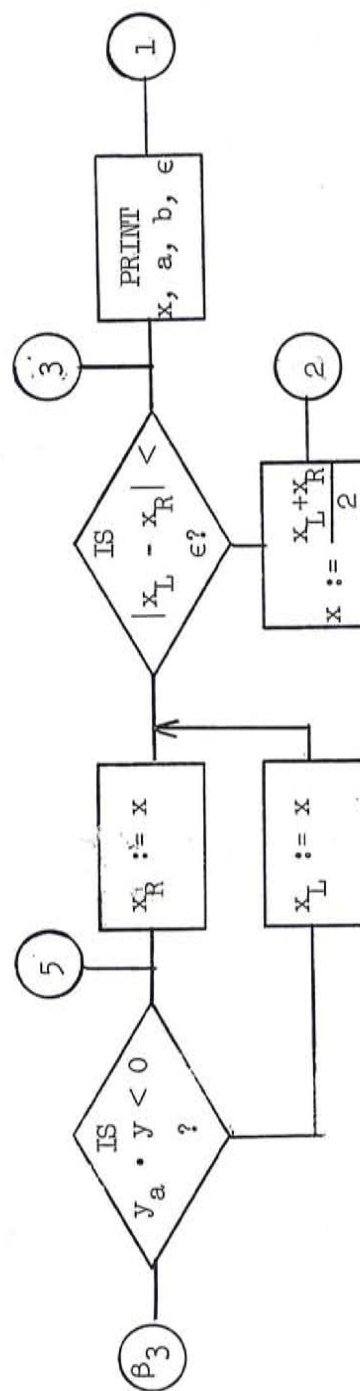
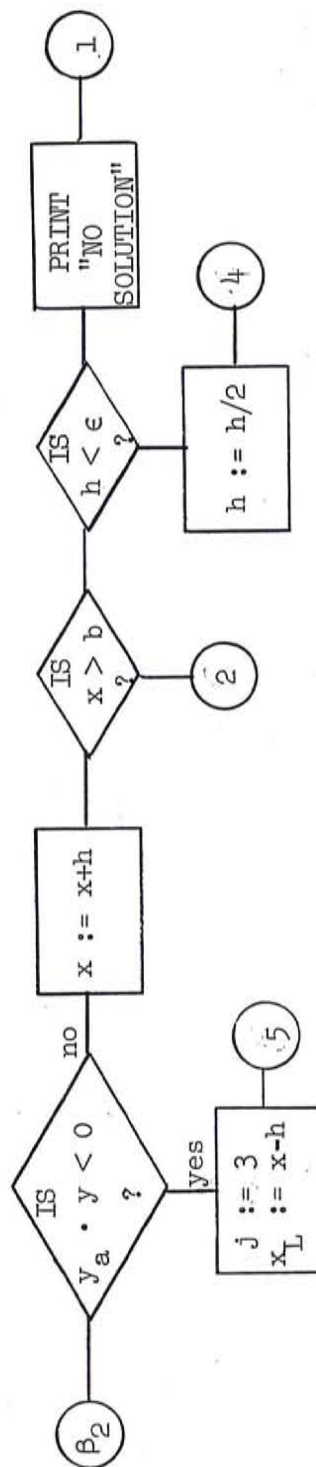
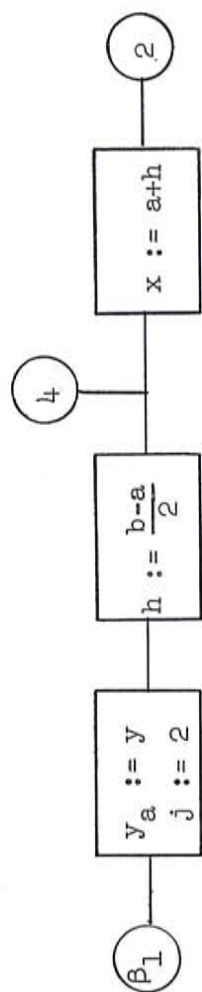
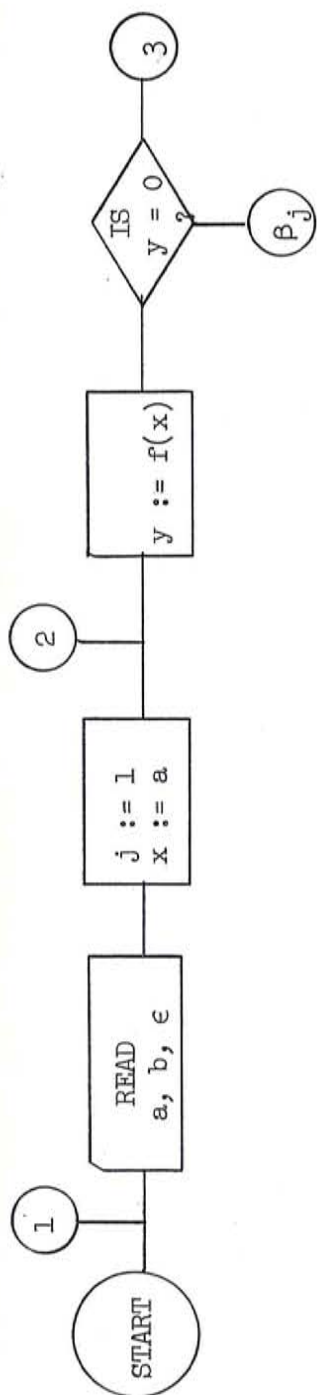
5: Y1 = SIN.(3.*X1*X1)

Example 3

Problem: To find a real solution (if it exists) of the equation $f(x) = 0$ (where f is a continuous function) on an arbitrary interval $[a,b]$, provided the roots (if there are more than one) are at least ϵ apart.

Analysis: We specify a , b , and ϵ as parameters, and we shall write the program in such a way that the evaluation of $f(x)$ need be written only once. This will allow an easy change to another equation.

The method used will be "half-interval convergence", in which the function is evaluated at $x = a$, and then the interval is scanned for a change of sign in the value of $f(x)$. If no change of sign is found, the scanning is repeated with a step size for searching equal to one-half the previous step size. If the step size becomes smaller than ϵ , and no change of sign is found, the process is terminated, and a comment is printed saying "NO SOLUTION".



If a change of sign is found between $x = x_1$ and $x = x_2$, the value of f is computed at $x_M = \frac{x_1 + x_2}{2}$, i.e., the midpoint of the interval $[x_1, x_2]$. We then determine which of the intervals $[x_1, x_M]$, $[x_M, x_2]$ contains the change of sign. We then compute the value of f at the midpoint of that smaller interval, etc., until the interval being considered finally has length less than ϵ , at which time either end may be taken as the solution with an error less than ϵ .

We make the storage assignments:

j:	J1	x:	X1	x_L :	Z1
a:	C1	y_a :	Y0	x_R :	Z2
b:	C2	y:	Y1		
ϵ :	C3	h:	Y2		

Rough Program: $[f(x) = \sin x + .3 \cos x = 0]$

Source Language Program:

```

300 USED IN SUBROUTINES

28 IS HIGHEST STATEMENT NUMBER

DIMENSION J(1) C(3) X(1)  N
          Y(2) Z(2)

```

```

1: Read a, b,  $\epsilon$ 
2: j = 7
3: x = a
4: y = sin x + .3 cos x
5: Go to 27 if y = 0
6: Go to j
7:  $y_a = y$ 
8: j = 12
9: h = (b-a)/2

```

```

1: A,B,EPSILON READ
2: J1 = 7
3: X1 = C1
4: Y1 SIN.(X1)+.3*COS.(X1)
5: GO TO 27 IF Y1 U 0.
6: GO TO J1
7: Y0 = Y1
8: J1 = 12
9: Y2 = (C2-C1)/2.

```


Rough Program

```

10: 12, x, a+h, h, b+h/2,
11: Go to 4
12: Go to 18 if  $y_a \cdot y < 0$ 
13: Go to 16 if  $h < \epsilon$ 
14:  $h = h/2$ 
15: Go to 10
16: Punch comment
17: Go to 1
18:  $j = 2^4$ 
19:  $x_L = x-h$ 
20:  $x_R = x$ 
21: Go to 27 if  $|x_L - x_R| < \epsilon$ 
22:  $x = (x_L + x_R)/2$ 
23: Go to 4
24: Go to 20 if  $y_a \cdot y < 0$ 
25:  $x_L = x$ 
26: Go to 21
27: Punch x, a, b,  $\epsilon$ 
28: Go to 1

```

Source Language Program

```

10: 12, X1, C1+Y2, Y2, C2+Y2/2.,
11: GO TO 4
12: GO TO 18 IF Y0*Y1Q0.
13: GO TO 16 IF Y2 Q C3
14: Y2 = Y2/2.
15: GO TO 10
16: AT$NO SO$ T$LUTIO$ T$N $
17: GO TO 1
18: J1 = 24
19: Z1 = X1-Y2
20: Z2 = X1
21: GO TO 27 IF A(Z1-Z2)QC3
22: X1 = (Z1+Z2)/2.
23: GO TO 4
24: GO TO 20 IF Y0*Y1 Q 0.
25: Z1 = X1
26: GO TO 21
27: TX1 TC1 TC2 TC3
28: GO TO 1

```

END

NOTE: In statement 10, the upper limit is $b+h/2$ rather than b because of the possibility that because of round-off error the iteration might end before the computation for $x = b$ is performed. In other words, if $x = a+mh = b$ exactly, it might happen that due to round-off errors, $a + mh$ is slightly larger than b , and the iteration would cease before this last computation with $x = a+mh$.

Example 4

Problem: Multiply the matrix $A = (a_{ij})$ by the matrix $B = (b_{ij})$ to produce the matrix $C = (c_{ij})$; i.e., $C = A \cdot B$. Assume that A and B (and therefore C) are square matrices of order $n \leq 20$.

Analysis: An element c_{ij} of the matrix C is computed by the formula

$$c_{ij} = \sum_{k=1}^n a_{ik} \cdot b_{kj} \quad .$$

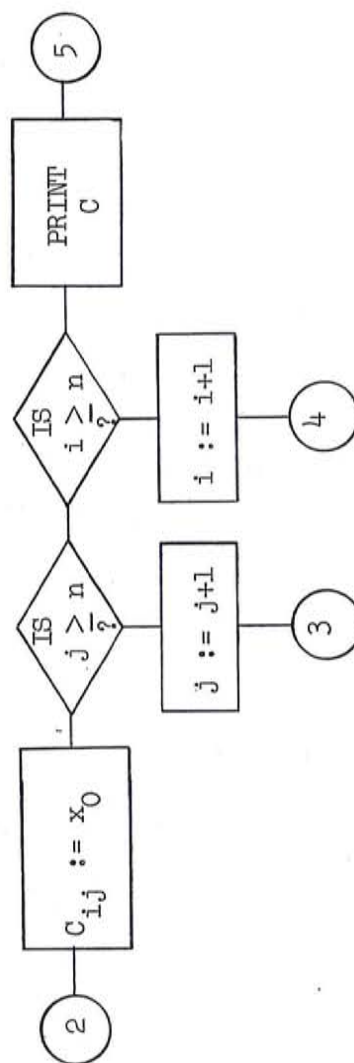
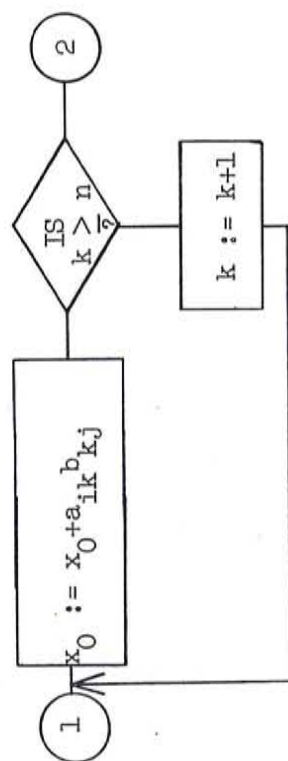
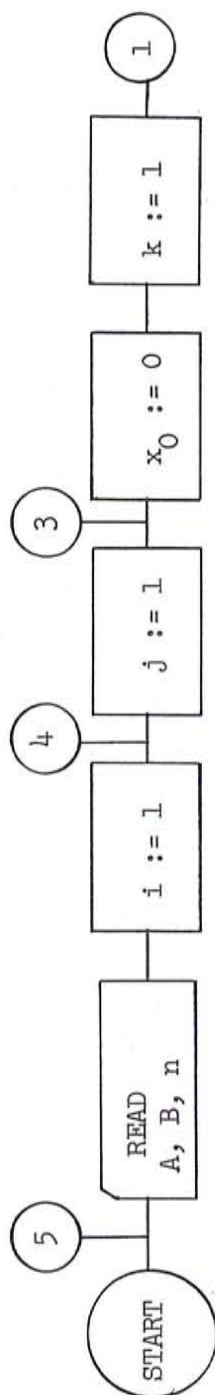
We shall write the program twice, the first time without using the iteration statement, to illustrate the usefulness of this type of statement in such problems.

We make the storage assignment:

i:	IO	x ₀ :	X0	a _{ij} :	X(IO,J0)
j:	J0	n:	11	b _{ij} :	Y(IO,J0)
k:	K0			c _{ij} :	Z(IO,J0)

The matrices A, B, and C will be in storage starting with X1, Y1, and Z1, respectively.

NOTE: The DIMENSION statement below reflects the fact that each of the matrices in this problem may have as many as $20 \cdot 20 = 400$ elements.



Rough Program:

```

1: Read, A, B, n
2: i = 1
3: j = 1
4:  $x_0 = 0$ 
5: k = 1
6:  $x_0 = x_0 + a_{ik} \cdot b_{kj}$ 
7: Go to 10 if  $k \geq n$ 
8: k = k+1
9: Go to 6
10:  $c_{ij} = x_0$ 
11: Go to 14 if  $j \geq n$ 
12: j = j+1
13: Go to 4
14: Go to 17 if  $i \geq n$ 
15: i = i+1
16: Go to 3
17: Punch matrix C
18: Go to 1

```

Source Language Program:

```

250 USED IN SUBROUTINES

18 IS HIGHEST STATEMENT NUMBER

DIMENSION I(1) J(0) K(0)      N
      X(399,11,1) Y(399,11,1)  N
      Z(399,11,1)

1: A,B,N READ
2: IO = 1
3: JO = 1
4: XO = 0.
5: KO = 1
6: XO = XO+X(IO,KO)*Y(KO,JO)
7: GO TO 10 IF KO W 11
8: KO = KO+1
9: GO TO 6
10: Z(IO,JO) = XO
11: GO TO 14 IF JO W 11
12: JO = JO+1
13: GO TO 4
14: GO TO 17 IF IO W 11
15: IO = IO+1
16: GO TO 3
17: TZ(1,1)...Z(11,11)
18: GO TO 1

END

```

Shorter Rough Program:

```

1:  Read A, B, n
2:  10, i, 1, 1, n,
3:  10, j, 1, 1, n,
4:   $x_0 = 0$ 
5:  6, k, 1, 1, n,
6:   $x_0 = x_0 + a_{ik} \cdot b_{kj}$ 
10:  $c_{ij} = x_0$ 
17: TZ(1,1)...Z(11,11)
18: Go to 1

```

Shorter Source Language Program:

```

250 USED IN SUBROUTINES

18 IS HIGHEST STATEMENT NUMBER

DIMENSION I(1) J(0) K(0)      N
      X(399,11,1) Y(399,11,1)  N
      Z(399,11,1)

1:  A, B, N READ
2:  10, IO, 1, 1, 11,
3:  10, JO, 1, 1, 11,
4:  XO = 0.
5:  6, KO, 1, 1, 11,
6:  XO = XO+X(IO,KO)*Y(KO,JO)
10: Z(IO,JO) = XO
17: T Z(1,1)...Z(11,11)
18: GO TO 1

END

```

Example 5

Problem: Solve a system of $n \leq 20$ simultaneous linear equations in n unknowns, assuming that one does not encounter a zero on the main diagonal of the coefficient matrix during the solution process.

Analysis: We shall use a Jordan Elimination Method, in which each diagonal coefficient is used to "clear" all other coefficients in its column to zero by appropriate multiplications and subtractions. Since we shall divide the "clearing row" by the diagonal element in that row before clearing the

column, we shall finish the process with only a diagonal of ones and the solution to the problem as the resulting right hand side of the equations.

We denote the system of equations to be solved by:

$$\begin{aligned}
 & a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = a_{1,n+1} \\
 & a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = a_{2,n+1} \\
 (1) \quad & \begin{array}{cccc} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{array} \\
 & a_{n1}x_1 + a_{n2}x_2 + \dots + a_{nn}x_n = a_{n,n+1}
 \end{aligned}$$

We divide the first row by its diagonal element a_{11} . Then to clear a_{21} to zero we subtract a_{21} times the first row from the second row, and so on. In general, to clear a_{ik} to zero (after row k has been divided by a_{kk}), we subtract a_{ik} times row k from row i ($i \neq k$). A typical element a_{ij} is thus transformed each time by the formulas:

$$(2) \quad a_{kj} = a_{kj} / a_{kk}$$

$$(3) \quad a_{ij} = a_{ij} - a_{ik}a_{kj} \quad (i \neq k)$$

where the value of a_{kj} in (3) is the result of (2). These transformations are performed for $k = 1, 2, \dots, n$. For each (fixed) k , we will let $i = 1, 2, \dots, k-1, k+1, \dots, n$, so as to operate on all rows except $i=k$. While transforming each row we will cycle on j from right to left; i.e., $j = n+1, n, n-1, \dots, k$, and we stop at $j=k$ since for $j < k$ there is no change in the matrix.

The array

$$A = (a_{ij}) = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1,n+1} \\ \vdots & & & \vdots \\ a_{n1} & a_{n2} & \dots & a_{n,n+1} \end{bmatrix}$$

is called the "matrix of coefficients" of the system (1).

Storage Assignments:

n:	I0	j:	J1
i:	I1	k:	K1
n+1:	I2	a_{ij} :	C(I1,J1)

The matrix (a_{ij}) will be stored starting with C1. The value of n+1 will be stored in I2 for use in the DIMENSION statement.

Rough Program:

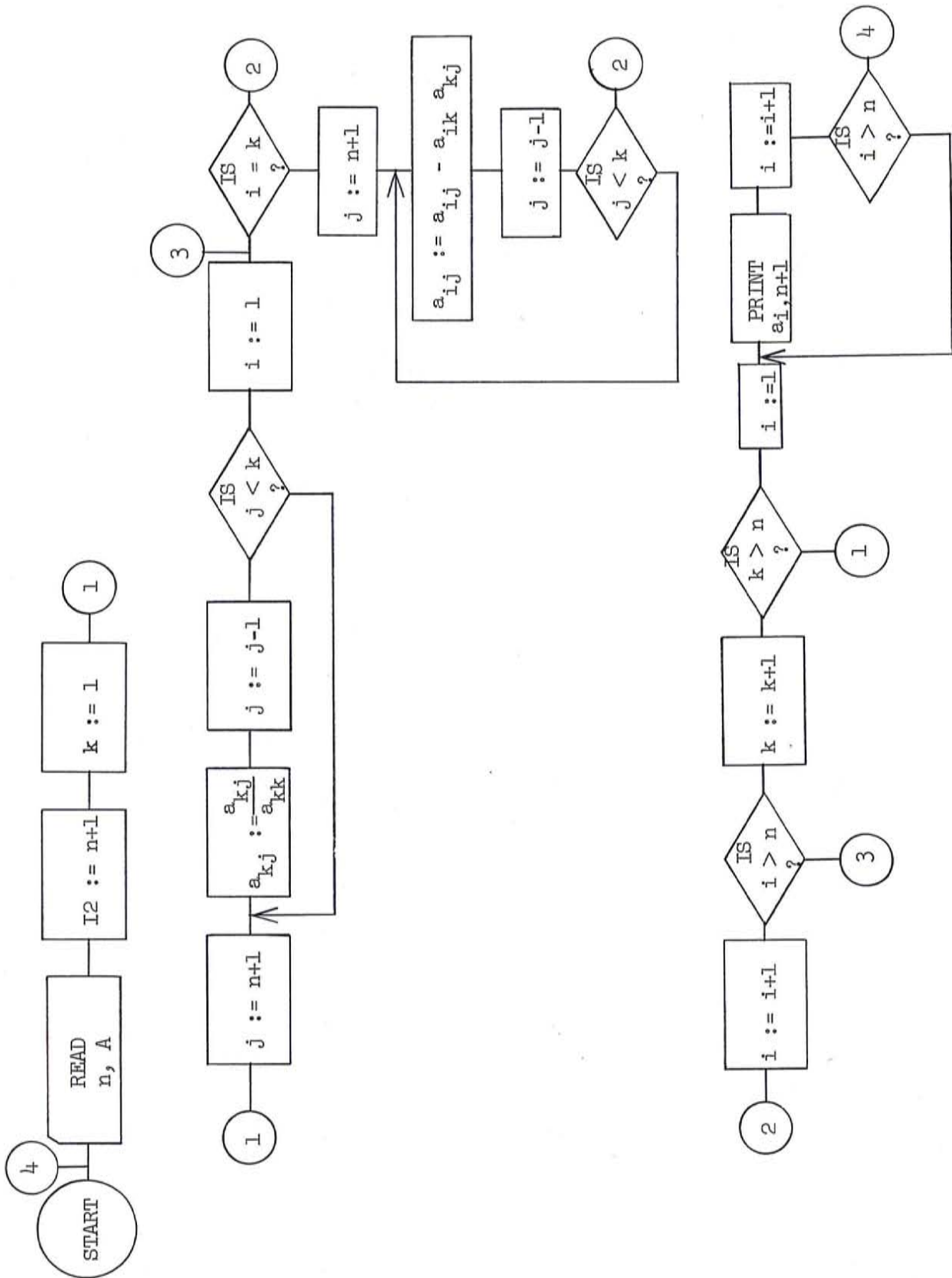
```

1: Read n,A
2: I2 = n+1
3: 10, k, 1, 1, n,
4: 5, j, n+1, -1, k,
5:  $a_{kj} = a_{kj}/a_{kk}$ 
6: 10, i, 1, 1, n,
7: Go to 10 if i=k
```

Source Language Program:

```

13 IS HIGHEST STATEMENT NUMBER
250 USED IN SUBROUTINES
DIMENSION I(2) J(1) K(1)      N
      C(420, I2, 1)
1: N, A READ
2: I2 = I0 + 1
3: 10, K1, 1, 1, I0,
4: 5, J1, I2, -1, K1,
5: C(K1,J1) = C(K1,J1)/C(K1,K1)
6: 10, I1, 1, 1, I0,
7: GO TO 10 IF I1 U K1
```



Rough Program:

```

8:  9, j, n+1, -1, k,
9:   $a_{ij} = a_{ij} - a_{ik}a_{kj}$ 
10: Dummy Statement
11: 12, i, 1, 1, n,
12:  $Ta_{i,n+1} Ti$ 
13: Go to 1

```

Source Language Program:

```

8:  9, J1, I2, -1, K1,
9:  C(I1,J1) = C(I1,J1)-C(I1,K1)*  N
      C(K1,J1)
10: I0 = I0
11: 12, I1, 1, 1, I0,
12: TC(I1,I2) TI1
13: GO TO 1

      END

```

Example 6

Problem: Solve a system of $n \leq 20$ simultaneous linear equations in n unknowns, assuming that a unique solution exists.

Analysis: We use the method of example 5, but we shall no longer assume that a_{kk} is never zero during the computation. If $a_{kk}=0$ for some k , we shall search the elements in column k below the diagonal; i.e., a_{ik} , $i > k$, for a non-zero element, and if there is one, we shall interchange the row containing this element with the k -th row, thus obtaining the desired non-zero diagonal element. If no such element is found, a comment will be punched saying "NO UNIQUE SOLUTION".

Since interchanging rows of a matrix is time-consuming, we shall not physically interchange them. We shall instead maintain a list of "names" of the rows and interchange the "names". Thus, if we have four rows in the matrix labeled "1", "2", "3", and "4", instead of interchanging, for example, the last two rows, we would change our list from 1,2,3,4 to 1,2,4,3 .

At the end of the process, we might have a list such as 4,2,1,3 . This would mean that the originally last row should now be first, etc. This will enable us to associate correctly the final values of the right side of the matrix with the variables involved.

For example, if the final list were 4,2,1,3 , the final matrix would be (for some values of a, b, c, and d):

$$\begin{bmatrix} 0 & 0 & 1 & 0 & a \\ 0 & 1 & 0 & 0 & b \\ 0 & 0 & 0 & 1 & c \\ 1 & 0 & 0 & 0 & d \end{bmatrix} .$$

If the variables in this example were x, y, z, and w (in that order), we would need to be able to tell that

$$z = a$$

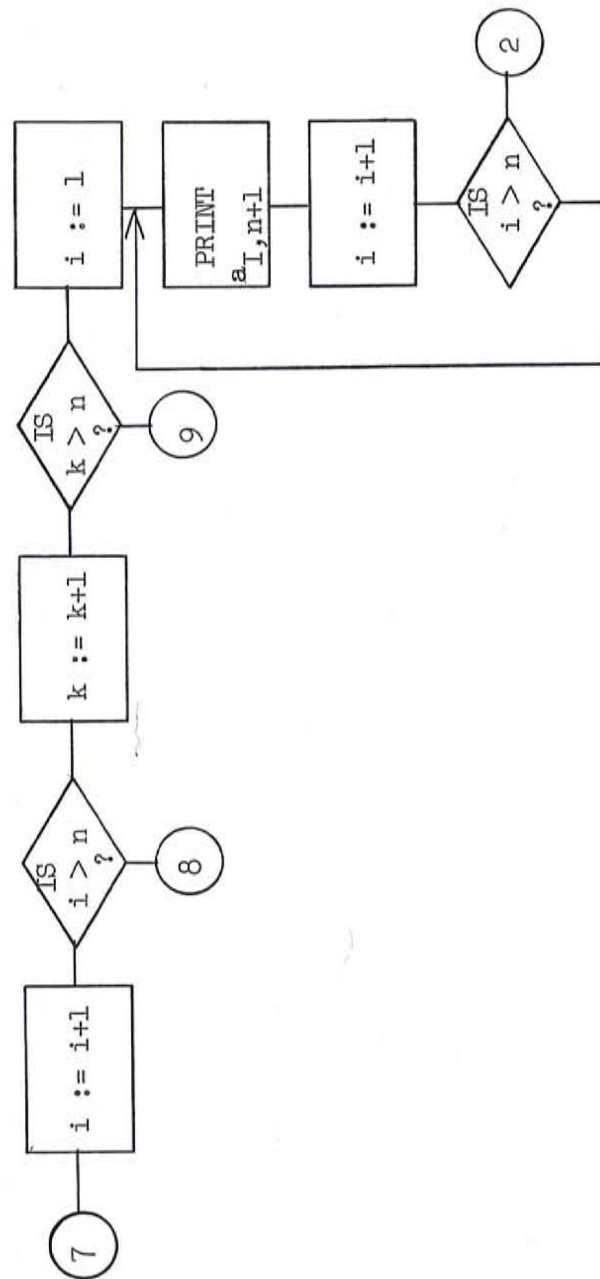
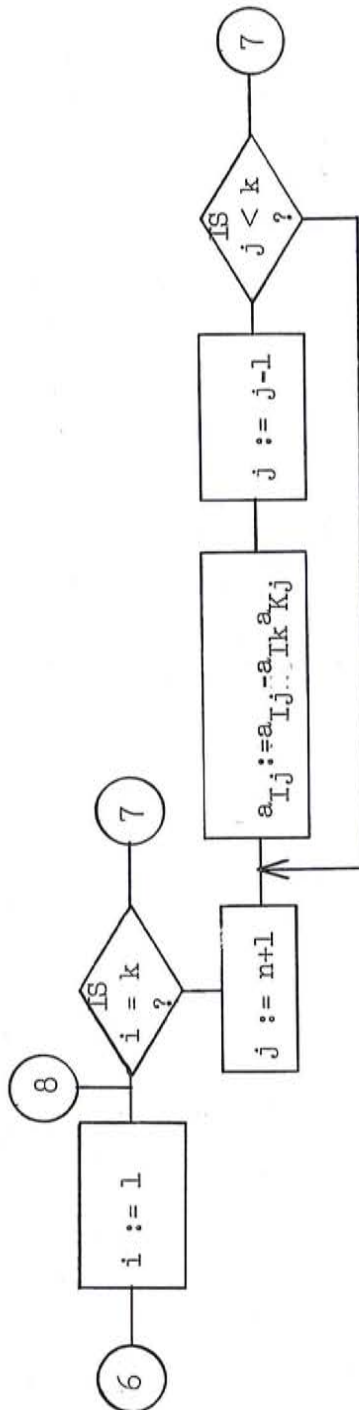
$$y = b$$

$$w = c$$

$$x = d$$

This assignment of the right side to variables comes immediately from the list 4,2,1,3 , since we now associate the number in row "4" (i.e., d) to x, the number in row "2" (i.e., b) to y, and so on.

The program below illustrates the use of a subscripted subscript to manipulate the "name" of a variable. In the flow diagram which follows the "names" are stored in m_1, m_2, \dots, m_n , so the "name" to be used as a subscript in each case is the number in m_1 . We shall indicate this by using capital letters. Thus K will be the number in m_K . The variable Z will be used as temporary storage while interchanging "names". It should be noted that we need "names" (and capital letters, etc.) only for row subscripts. Column subscripts are not affected in any way.



Storage Assignments:

Z:	J1	j:	J0
i:	I0	k:	K0
n:	I1	m_i :	KIO
n+1:	I2	a_{ij} :	C(I0,J0)

The list m_1, m_2, \dots, m_n will start in K1. The matrix $A = (a_{ij})$ will start in C1.

Rough Program:

```

1: Read n,A
2: I2 = n+1
3: 4, i, 1, 1, n,
4:  $m_i = i$ 
5: 13, k, 1, 1, n,
6: Go to 17 if  $a_{Kk} = 0$ 
7: 8, j, n+1, -1, k,
8:  $a_{Kj} = a_{Kj}/a_{Kk}$ 
9: 13, i, 1, 1, n,
10: Go to 13 if i=k
11: 12, j, n+1, -1, k,
12:  $a_{Ij} = a_{Ij} - a_{Ik} \cdot a_{Kj}$ 

```

Source Language Program:

```

250 USED IN SUBROUTINES
24 IS HIGHEST STATEMENT NUMBER
DIMENSION I(2) J(1) K(20)    N
      C(420, I2, 1)
1: N, A READ
2: I2 = I1+1
3: 4, IO, 1, 1, I1,
4: KIO = IO
5: 13, KO, 1, 1, I1,
6: GO TO 17 IF C(KKO,KO) U 0.
7: 8, JO, I2, -1, KO,
8: C(KKO,JO) = C(KKO,JO)/C(KKO,KO)
9: 13, IO, 1, 1, I1,
10: GO TO 13 IF IO U KO
11: 12, JO, I2, -1, KO,
12: C(KIO,JO) = C(KIO,JO) -C(KIO,KO)N
      *C(KKO,JO)

```

Rough Program:

13: Dummy Statement
 14: 15, i, 1, 1, n,
 15: $Ta_{I,n+1} \quad Ti$
 16: Go to 1
 17: 18, i, k+1, 1, n,
 18: Go to 21 if $|a_{Ik}| > 0$
 19: Punch comment

 20: Go to 1
 21: $Z = m_i$
 22: $m_i = m_k$
 23: $m_k = Z$
 24: Go to 7

Source Language Program:

13: IO = IO
 14: 15, IO, 1, 1, I1,
 15: TC(KIO,I2) T IO
 16: GO TO 1
 17: 18, IO, KO+1, 1, I1,
 18: GO TO 21 IF AC(KIO,KO) V O.
 19: AT\$NO UN\$T\$IQUE \$ T \$SOLUT\$ N
 T\$ION \$

 20: GO TO 1
 21: J1 = KIO
 22: KIO = KKO
 23: KKO = J1
 24: GO TO 7

 END

EXAMPLE 7 - SOLUTION OF
 $CPX+X=0$ BY NEWTONS METHOD
 USING AN ITERATION STATEMENT

SOURCE LANGUAGE

```

750 USED IN SUBROUTINES
20 IS HIGHEST STATEMENT NUMBER
DIMENSION C(1) X(1) Z(1)
1  X1,C1,READ
2  3,X1,X1,-Z1,X1-AZ1+.00001,
3  Z1=(C1PX1+X1)/(2.3025851* N
    LOG10.(C1)*C1PX1+1.0)
4  TX1
5  HALT
    END
  
```

OBJECT PROGRAM

2	0043	0000011990+	1990	8802080024+
2	0168	2100390216+	0216	0002330183+
2	0266	6700410268+	0268	2019500316+
3	0272	2119500366+	0366	6000390368+
3	0183	6000370418+	0418	6900390416+
3	0468	2119600516+	0516	8801840025+
3	0472	2119500566+	0566	6000370518+
4	0522	3419500572+	0572	2100410666+
4	0568	2119600716+	0716	6000670618+
100	0047	0000050816+	0816	0019498000+
100	0065	1000000051+	0064	1000000046+
100	0002	6400008010+	0003	6900008010+
100	0007	8800008010+	0008	8900020040+
100	0012	0000000000+	0013	0000000000+
100	0017	0000000000+	0018	0000000000+
100	0022	0000000000+	0023	0000000000+
100	0027	0000000000+	0028	0000000000+
100	0032	0000000000+	0033	0000000000+

DATA

C1 4000000051 X1 1000000051

RESULTS

X1 4999998950-

0208	2119500166+	0044	0000020166+	0166	6000390168+
0233	6100410218+	0218	3200390172+	0172	2100390266+
0316	6000390318+	0318	3319500222+	0222	3200640272+
0368	3319500322+	0322	4601670183+	0045	0000030183+
0416	8801880020+	0188	2119500466+	0466	6000370468+
0184	3900660372+	0372	3919500422+	0422	3200650472+
0518	6900390616+	0616	8802380020+	0238	3200390522+
0666	0001670233+	0046	0000040167+	0167	6000680568+
0618	2119610766+	0766	8802340026+	0234	2119500816+
0068	0000010004+	0067	0000060001+	0066	2302585151+
0063	0000000005+	0000	0100000000+	0001	6300020036+
0004	7100008010+	0005	7200008010+	0006	8700020038+
0009	0000210042+	0010	0001000063+	0011	0000000000+
0014	0000000000+	0015	0000000000+	0016	0000000000+
0019	0000001130+	0020	6665877700+	0021	8765877700+
0024	7965616400+	0025	7376679190+	0026	7577756368+
0029	0000000000+	0030	0000000000+	0031	0000000000+
0034	0000000000+	0035	0000000000+	0042	0000008000+

*

EXAMPLE 8 - SOLUTION OF
 $CPX+X=0$ BY THE HALF-INTERVAL
 METHOD USING ITERATION
 STATEMENTS

SOURCE LANGUAGE

```

750 USED IN SUBROUTINES
20 IS HIGHEST STATEMENT NUMBER
DIMENSION C(3) D(1) X(1) Y(1)
1  C1,C2,C3,D1,READ
2  Y0=C1PC2+C2
3  6,D0,C3=C2,-D0/2.0,D1,
4  6,X1,C2,D0,C3,
6  GO TO 9 IF Y0*(C1PX1+X1)Q0.
7  AT$NO SO$T$LUTIO$T$N      $
8  HALT
9  12,D0,D0,-D0/2.0,D1/2.0,
10 GO TO 12 IF Y0*(C1PX1+X1)Q0.
11 X1=X1+D0
12 X1=X1-D0/2.
13 TX1
14 HALT
END

```

OBJECT PROGRAM

2	0047	0000011990+	1990	8802120024+
3	0172	6900380220+	0220	8801960020+
3	0270	6000390222+	0222	3300380226+
3	0272	3400680276+	0276	6180030326+
4	0322	3300410426+	0426	4601710191+
4	0420	0003410291+	0341	6000400422+
6	0472	3300390526+	0526	6180030576+
6	0522	6900430520+	0520	8802460020+
7	0726	4600550271+	0271	0002210341+
7	0572	2119600570+	0570	6000750622+
8	0670	6000710722+	0722	2119630720+
9	0770	0019498000+	0055	0000090321+
9	0441	6000400822+	0822	3400680776+
9	0870	6000410872+	0872	3400680926+
10	0976	4603710391+	0056	0000100391+
11	0296	3200431026+	1026	3900441076+
12	0421	6000401022+	1022	3200430177+
12	1072	3400680227+	0227	2119501070+
13	1120	0003710441+	0059	0000130371+
14	0223	2119610521+	0521	8801930026+

0212	2119500170+	0048	0000020170+	0170	6000370172+
0196	3200380176+	0176	2100440270+	0049	0000030270+
0226	2100400320+	0320	0002410191+	0241	6000400272+
0326	3200400376+	0376	2100400370+	0370	6000400322+
0050	0000040191+	0191	6000380372+	0372	2100430420+
0422	3200430476+	0476	2100430470+	0470	6000430472+
0576	4602210291+	0052	0000060291+	0291	6000370522+
0246	3200430626+	0626	3900440676+	0676	3300690726+
0221	0001710241+	0053	0000070171+	0171	6000760572+
0622	2119610620+	0620	6000730672+	0672	2119620670+
0720	8801920025+	0192	2119500770+	0054	0000080770+
0321	6000400772+	0772	2100400820+	0820	0004410391+
0776	6180030826+	0826	3200400876+	0876	2100400870+
0926	2119500920+	0920	6000400922+	0922	3319500976+
0391	6000370972+	0972	6900430970+	0970	8802960020+
1076	3300691126+	1126	4600580421+	0057	0000110421+
0177	2100431020+	0058	0000121020+	1020	6000401072+
1070	6000431122+	1122	3319500277+	0277	2100431120+
0371	6000780173+	0173	2119600471+	0471	6000770223+
0193	2119500571+	0060	0000140571+	0571	0019498000+

100	0078	0000010013+	0077	0000060001+
100	0073	0000000305+	0072	7384836976+
100	0068	2000000051+	0067	0000000011+
100	0003	6900008010+	0004	7100008010+
100	0008	8900008010+	0009	0000210046+
100	0013	0000000000+	0014	0000000000+
100	0018	0000000000+	0019	0000001130+
100	0023	0000000000+	0024	7965616400+
100	0028	0000000000+	0029	0000000000+
100	0033	0000000000+	0034	0000000000+

DATA

C1	4000000051	ETC	5000000051-
----	------------	-----	-------------

RESULTS

X1	4999923750-
----	-------------

0076	0000030007+	0075	0000000307+	0074	7576008276+
0071	0000000303+	0070	7500000000+	0069	0000000000+
0000	0100000000+	0001	6300040036+	0002	6400020040+
0005	7200008010+	0006	8700020042+	0007	8800020044+
0010	0001000067+	0011	0000000000+	0012	0000000000+
0015	0000000000+	0016	0000000000+	0017	0000000000+
0020	6665877700+	0021	8765877700+	0022	0000000000+
0025	6177756368+	0026	7577756368+	0027	0000000000+
0030	0000000000+	0031	0000000000+	0032	0000000000+
0035	0000000000+	0046	0000008000+	0046	0000008000+
ETC	5000000051	D1	1000000047	*	

EXAMPLE 9 - THE SOLUTION OF
AN ORDINARY DIFFERENTIAL
EQUATION USING THE RUNGE-KUTTA
SUBROUTINE

SOURCE LANGUAGE

```

750 USED IN SUBROUTINES
11 IS HIGHEST STATEMENT NUMBER
DIMENSION Y(11) D(0)
1  Y0,....,Y3,DELTA Y0 READ
2  RKSUB.(4,LY0,6)
3  TY0 TY1 TY2 TY3
4  GO TO 2 IF Y0 Q 2
5  GO TO 1
6  Y4=D0
7  Y5=D0*(Y2+Y0)
8  Y6=D0*Y3
9  Y7=D0*(-Y0*Y2*Y3*-Y1*Y0*Y0)
10 GO TO 2
11 END

```

OBJECT PROGRAM

2	0050	0000011990+	1990	8802060024+
2	0166	2119600214+	0214	6000630216+
3	0314	8801800025+	0180	2119500364+
3	0414	6000680366+	0366	2119610464+
3	0466	2119630564+	0564	6000650516+
4	0053	0000040664+	0664	6000700566+
6	0616	3319490220+	0220	4600510165+
7	0280	6000360666+	0666	2100410714+
8	0270	3900360320+	0320	2100420764+
9	0370	2100430814+	0058	0000090814+
9	0470	6180030520+	0520	2119500864+
10	0620	3919500670+	0670	6180030720+
100	0914	0001790051+	0070	0000000002+
100	0066	0000070002+	0065	0000070003+
100	0061	0000000009+	0000	0100000000+
100	0004	7100008010+	0005	7200008010+
100	0009	0000120049+	0010	0001000061+
100	0014	0000000000+	0015	0000000000+
100	0019	0000001130+	0020	0000000000+
100	0024	7965616400+	0025	7972828462+
100	0029	0000000000+	0030	0000000000+
100	0034	0000000000+	0035	0000000000+

0206	2119500164+	0051	0000020164+	0164	6000640166+
0216	2119610264+	0264	6000620266+	0266	2119620314+
0052	0000030364+	0364	6000690316+	0316	2119600414+
0464	6000670416+	0416	2119620514+	0514	6000660466+
0516	2119640614+	0614	8802300026+	0230	2119500664+
0566	8801860023+	0186	2119490170+	0170	6000370616+
0054	0000050165+	0165	0002800050+	0055	0000060280+
0056	0000070714+	0714	6000370716+	0716	3200390270+
0057	0000080764+	0764	6000400766+	0766	3900360370+
0814	6000370816+	0816	3900380420+	0420	3900370470+
0864	6000390866+	0866	3900370570+	0570	3900400620+
0720	3900360770+	0770	2100440914+	0059	0000100914+
0069	0000040003+	0068	0000070000+	0067	0000070001+
0064	0000000004+	0063	0000000037+	0062	0000000006+
0001	6300008010+	0002	6400010036+	0003	6900008010+
0006	8700008010+	0007	8800120037+	0008	8900008010+
0011	0000000000+	0012	0000000000+	0013	0000000000+
0016	0000000000+	0017	0000000000+	0018	0000000000+
0021	0000000000+	0022	0000000000+	0023	6673766183+
0026	7577756368+	0027	0000000000+	0028	0000000000+
0031	0000000000+	0032	0000000000+	0033	0000000000+
0049	0000008000+	0049	0000008000+	0049	0000008000+

DATA

D0	2000000050	Y2	1000000051
----	------------	----	------------

RESULTS

Y0	2000000050	Y1	2399260050
Y0	4000000050	Y1	5587418050
Y0	6000000050	Y1	9531262450
Y0	8000000050	Y1	1416620351
Y0	1000000051	Y1	1938914851
Y0	1200000051	Y1	2505277051
Y0	1400000051	Y1	3095865251
Y0	1600000051	Y1	3684212751
Y0	1800000051	Y1	4233317451
Y0	2000000051	Y1	4684769051

Y3	1000000051	Y0	50	Y1 *	50
Y2	1198526851	Y3	9771385450		
Y2	1386967751	Y3	8958392150		
Y2	1551617851	Y3	7358924450		
Y2	1674879451	Y3	4792589550		
Y2	1735988351	Y3	1120974750		
Y2	1711422351	Y3	3812561150-		
Y2	1573002851	Y3	1035873351-		
Y2	1280886051	Y3	1942439951-		
Y2	7646219150	Y3	3345598851-		
Y2	1375931650-	Y3	6060309951-		

EXAMPLE 10 MATRIX MULTIPLICATION USING MATRIX NOTATION

SOURCE LANGUAGE

```

400 USED IN SUBROUTINES
8 IS HIGHEST STATEMENT NUMBER
DIMENSION I(1) J(1) K(1) N
X(100,K1,1) Y(100,J1,1) N
Z(100,J1,1)
1  I1,J1,K1,X,Y, READ
2  6,I0,1,1,I1,
3  6,J0,1,1,J1,
4  Z(I0,J0)=0
5  6,K0,1,1,K1,
6  Z(I0,J0)=X(I0,K0)*Y(K0,J0)N
   Z(I0,J0)
7  TZ(1,1)...Z(I1,J1)
8  GO TO 1
END

```

OBJECT PROGRAM

2	0346	0000011990+	1990	8804990024+
2	0459	2100360507+	0507	0005150465+
3	0557	6000360609+	0609	1100370659+
3	0465	6003550709+	0709	2100380607+
3	0809	2100380657+	0657	6000380859+
4	0349	0000040565+	0565	6000360959+
4	0757	6003571059+	1059	2119620807+
4	0563	2119500857+	0857	6003561159+
5	1209	8080030663+	0663	6919490907+
5	1259	2100400957+	0957	0007150665+
6	1007	6000401409+	1409	1100411459+
6	0665	6000360460+	0460	2119601057+
6	0560	2119621157+	1157	8805160025+
6	1207	6000400660+	0660	2119601257+
6	0760	2119621357+	1357	8805660025+
6	1407	6000360860+	0860	2119601457+
6	0960	2119620658+	0658	8804670025+
6	0963	3219501013+	1013	2119500708+
6	1110	2119610808+	0808	6003571160+
6	1210	1080051063+	1063	8080031113+
7	0558	0005080615+	0508	0004580515+
7	0958	6000391310+	1310	2119611008+
7	0567	1003601410+	1410	2119501108+
7	0461	2119611208+	1208	6003570511+
7	0561	2119511308+	1308	6003620611+
8	1408	6019500711+	0711	2119621458+

0499	2119500457+	0347	0000020457+	0457	6003550459+
0515	6003550509+	0509	1000360559+	0559	2100360557+
0659	6180030463+	0463	4604580465+	0348	0000030465+
0607	0006150565+	0615	6003550759+	0759	1000380809+
0859	1100390909+	0909	6180030513+	0513	4605080565+
0959	2119600707+	0707	6000381009+	1009	2119610757+
0807	8804660025+	0466	8002441109+	1109	1080050563+
1159	8804720023+	0472	2119490613+	0613	6019501209+
0907	2420000713+	0350	0000050713+	0713	6003551259+
0715	6003551309+	1309	1000401359+	1359	2100401007+
1459	6180030763+	0763	4605580665+	0351	0000060665+
1057	6000380510+	0510	2119611107+	1107	6003570560+
0516	8080030813+	0813	6022440610+	0610	2119501207+
1257	6000380710+	0710	2119611307+	1307	6003580760+
0566	8080030863+	0863	6021430810+	0810	2119511407+
1457	6000400910+	0910	2119610608+	0608	6003590960+
0467	8080030913+	0913	6020421010+	1010	3919510963+
0708	6000361060+	1060	2119600758+	0758	6000381110+
1160	2119620858+	0858	8805170025+	0517	8002441210+
1113	6919500908+	0908	2420001163+	1163	0005580715+
0352	0000070458+	0458	6000371260+	1260	2119600958+
1008	6003571360+	1360	2119621058+	1058	8805670025+
1108	6003551460+	1460	2119601158+	1158	6003550461+
0511	2119621258+	1258	8806170025+	0617	1003610561+
0611	2119601358+	1358	6019510661+	0661	2119611408+
1458	8806670026+	0667	2119500761+	0353	0000080761+

100	0761	0004690346+	0362	0000020007+
100	0358	0000000007+	0357	0000000008+
100	0000	0100000000+	0001	6300008010+
100	0005	7200020040+	0006	8701010042+
100	0010	0001000354+	0011	0000000000+
100	0015	0000000000+	0016	0300010041+
100	0020	0000000000+	0021	0000000000+
100	0025	8774618387+	0026	7577756368+
100	0030	0000000000+	0031	0000000000+
100	0035	0000000000+	0345	0000008000+

DATA

X1	1000000052	X2	2000000052
X3	3000000052	X4	4000000052
X5	5000000052	X6	6000000052
Y1	1000000051	ETC	2000000051
ETC	6000000051	ETC	7000000051

RESULTS

Z1	1300000053	Z2	1600000053
Z6	2700000053	Z7	3400000053
Z11	4100000053	Z12	5200000053

0361	0800080000+	0360	0000080000+	0359	0000000006+
0356	0000000000+	0355	0000000001+	0354	0000000008+
0002	6400008010+	0003	6900020036+	0004	7100020038+
0007	8801010143+	0008	8901010244+	0009	0000090345+
0012	0000000000+	0013	0000000000+	0014	0000000000+
0017	0300010039+	0018	0300010039+	0019	0000001480+
0022	0000000000+	0023	6673766183+	0024	7965616400+
0027	0000000000+	0028	0000000000+	0029	0000000000+
0032	0000000000+	0033	0000000000+	0034	0000000000+
0345	0000008000+	0345	0000008000+	0345	0000008000+

ETC 3000000051
ETC 8000000051

ETC 4000000051
ETC 9000000051

I1 3
J1 5
K1 2
ETC 5000000051
ETC * 1000000052

Z3 1900000053
Z8 4100000053
Z13 6300000053

Z4 2200000053
Z9 4800000053
Z14 7400000053

Z5 2500000053
Z10 5500000053
Z15 8500000053

GAT MANUAL ERRATA AND ADDENDA

- P. 12 line 3 . . . an inductance L . . .
- P. 13 line 14 . . . accomplished by an iteration statement . . .
- P. 14 line 8 . . . $1./(20.*C5)$. . .
- P. 14 bottom line - change to "be evaluated twenty times instead of once."
- P. 24 line 26 insert after "an alphabetic constant" (if less than 5 characters, blanks are inserted on the left to make 5 characters).
- P. 30 line 8 $J2*I7/J4/K5*K2$ means $((J2*I7)/J4)/K5)*K2$
- P. 49 line 16 replace previous correction with: If a continuation symbol is used inside the $\$$'s of an alphabetic constant, remember that blanks are valid characters in an alphabetic constant. Hence the N, in this case, must fall in column 80 to be treated as a continuation symbol.
- P. 50 card format 3(5 instr./card) should have 1994 in cols. 7-10.
- P. 56 line 1 (with the exception of alphabetic data and messages using the notation, which are both listed in the above format)
- P. 60 line 18 U for =,
- P. 61 line 13 $T\lambda_1 T\lambda_2 \dots T\lambda_n$ (single variable) or $T\lambda_j \dots \lambda_k$
- P. 63 line 2 FIX SUBROUTINE: 18 locations
- P. 63 line 7 Error No.: 1 if $V \geq 10^9$ or if $V < 10^{-1}$
- P. 63 line 10 FLOAT SUBROUTINE: 10 locations
- P. 63 line 14 Error No.: 2 if $1 V 1 \geq 10^8$
- P. 64 line 1 change to "3. READ SUBROUTINE: 121 locations, 1 calling name.
- P. 64 line 2 Calling name: READ. (0) The (0) must appear for correct compilation.
- P. 64 line 9 PRINT SUBROUTINE: 159 locations
- P. 65 line 1 MATRIX SUBSCRIPTION SUBROUTINE: 21 locations
- P. 65 line 7 Error No.: 4 if attempting matrix subscription of a variable which has not been defined as a matrix in the DIMENSION statement.
- P. 65 line 10 GENERAL EXPONENTIATION SUBROUTINE: 169 locations
- P. 66 line 3 9 for $-51 \geq V \geq 50$ in 10^V ;
- P. 66 line 14 SQUARE ROOT SUBROUTINE: 37 locations
- P. 67 line 1 ARCTANGENT SUBROUTINE: 51 locations
- P. 67 line 8 10^X SUBROUTINE: 37 locations
- P. 67 line 11 $10^{-51} < 10^V < 10^{50}$
- P. 67 line 14 LOG (BASE 10) SUBROUTINE: 41 locations
- P. 68 line 3 SINE-COSINE SUBROUTINE: 57 locations
- P. 70 add:

13. HEAD SUBROUTINE: 12 locations, 1 calling name.

Calling name: HEAD. (\$XXXXX\$, \$YYYYY\$, \$ZZZZZ\$, \$WWWWW\$, \$VVVVV\$)

This subroutine prints the alphabetic characters XXXXX over the first variable names field of the data print format; YYYYY over the second; ZZZZZ over the third; etc. To get a blank heading on a column it is sufficient to write \$\$. All five headings (blank or not) must be specified.

Error No. None

P. 71 line 16 . . . or too extensive use of matrix subscription in this statement.
 P. 71 line 20 . . . or more than number designated in ℓ ABCONS.
 P. 72 insert - 98 Transfer to location 0000 during execution of GAT.
 P. 83 line 21 Y1=SIN.(X1)+.3*COS.(X1)
 P. 87 line 5 X(400, 11, 1) Y(400, 11, 1) Z(400, 11, 1)
 P. 88 line 5 X(400, 11, 1) Y(400, 11, 1) Z(400, 11, 1)
 P. 94 line 2 remote connector ③ should be ⑤
 P. 108 line 15 +Z(10, J0)

ie

01/29/61