20-GATE

ALGEBRAIC COMPILER

FOR THE

BENDIX G-20

COMPUTATION CENTER

CARNEGIE INSTITUTE OF TECHNOLOGY

PITTSBURGH, PENNA.

JULY 28, 1961

INTRODUCTION

THIS IS A PRELIMINARY MANUAL FOR 20-GATE. IT GIVES A DESCRIPTION OF THE 20-GATE LANGUAGE AND INFORMATION ON THE PREPARATION OF CARDS FOR PROGRAM AND DATA. IT IS EXPECTED THAT FAIRLY EXTENSIVE CHANGES WILL BE MADE IN 20-GATE DURING THE SUMMER (1967). AS EACH CHANGE IS MADE AND ADDED TO THE SYSTEM, A DESCRIPTION OF THE CHANGE WILL BE MADE AVAILABLE AT THE COMPUTATION CENTER. IN GENERAL, THE CHANGES WILL BE SUCH THAT ALL EXISTING PROGRAMS WILL CONTINUE TO RUN. (SEE, HOWEVER SECTION 1 OF PART II ON PUNCHING STATEMENT CARDS.) IN SEPTEMBER 1967 A NEW MANUAL WILL BE AVAILABLE CONTAINING A DESCRIPTION OF THE REVISED SYSTEM. IT IS THEREFORE REQUESTED THAT USERS WHO NOTE ERRORS OR AMBIGUITIES IN THIS MANUAL BRING THEM TO THE ATTENTION OF THE COMPUTER CENTER STAFF.

THE 20-GATE LANGUAGE IS ESSENTIALLY EQUIVALENT TO THE "GATE" LANGUAGE (CORREGATE) WHICH WAS USED ON THE CARNEGIE TECH 650; INDEED, EXISTING 650-GATE PROGRAMS CAN BE RUN ON THE 6-20 WITH ONLY MINOR CHANGES. (SEE APPENDIX C FOR A DISCUSSION OF DIFFERENCES BETWEEN 20-GATE AND 650-GATE.) HOWEVER, CERTAIN FEATURES WHICH WERE AVAILABLE IN 650-GATE HAVE NOT YET BEEN PROVIDED IN 20-GATE, ALTHOUGH THEY WILL BECOME AVAILABLE SOME TIME DURING THE SUMMER. THE MOST IMPORTANT SUCH FEATURES ARE THOSE WHICH REQUIRED THE DISK UNIT ON THE 650 - DATA RECORDS ("MERAM" AND "DISK", ETC.), SEGMENTATION AND RELOCATABLE SUBROUTINES.

THIS MANUAL CONSISTS OF THE FOLLOWING:

PART I: INTRODUCTION TO 20-GATE PROGRAMMING. (THIS PART HAS NOT YET BEEN PREPARED; IT WILL BE IN THE FINAL MANUAL.)
PART II: DEFINITION OF THE LANGUAGE.
PART III: HOW TO USE THE SYSTEM.

APPENDIX A: ERROR INDICATIONS.
APPENDIX B: SUBROUTINES. THIS PART OF THE MANUAL HAS NOT YET BEEN COMPLETED. THE PRESENT MANUAL CONTAINS ONLY A LIST OF CURRENTLY AVAILABLE SUBROUTINES. MANY MORE SUBROUTINES WILL BE WRITTEN IN THE NEXT FEW WEEKS, AND DESCRIPTIONS WILL BE MADE AVAILABLE AS THE ROUTINES ARE CHECKED OUT.
APPENDIX C: CHANGES IN 20-GATE FROM 650-GATE.
APPENDIX D: SUMMARY OF THE SYNTAX OF THE LANGUAGE.
APPENDIX E: DESCRIPTION OF THE SYNTAX.

THE FORMAL DESCRIPTION OF THE LANGUAGE IN PART II OF THE MANUAL HAS, FOR EACH TOPIC, BEEN ORGANIZED TO CONTAIN THE FOLLOWING, ALWAYS IN THE SAME ORDER:

EXAMPLES OF THE LANGUAGE UNIT BEING DEFINED
DETAILED DESCRIPTION IN ENGLISH
SYNTAX
MORE EXAMPLES, SOME OF THEM INTENTIONALLY ERRONEOUS

1. CONSTANTS AND VARIABLES

1.1 CONSTANTS

1.1.1. NUMERICAL CONSTANTS

EXAMPLES:

1	594.
-.2	03.0
$25_{10}2$ ($25 * 10^2 = 2500$)	
$.4_{10}-40$ ($.4 * 10^{-40}$)	

DESCRIPTION:

A CONSTANT IS A REAL NUMBER CONTAINING ONE OR MORE DIGITS AND, POSSIBLY, A DECIMAL POINT, '.', WHICH MAY APPEAR ANYWHERE AMONG THE DIGITS. TO A CONSTANT MAY BE ATTACHED AN EXPONENT SIGNALLED BY THE MARK '$_{10}$' FOLLOWED BY THE NUMERICAL POWER OF 10 WHICH IS THE ACTUAL EXPONENT. THIS EXPONENT IS AN INTEGER WITH, IF DESIRED, A SIGN.

THE VALUE, X, OF A CONSTANT MUST BE ZERO OR SATISFY

$1.2744735289039_{10}-57 < |X| < 3.450873173389_{10}69$

SYNTAX:

<DIGIT> ::= 0|1|2|3|4|5|6|7|8|9

<INTEGER> ::= <DIGIT> | <INTEGER> <DIGIT>

<DECIMAL NUMBER> ::= <INTEGER> | <INTEGER>. | .<INTEGER> | <INTEGER>.<INTEGER>

<EXPONENT PART> ::= $_{10}$<INTEGER> | $_{10}$<ADDING OPERATOR> <INTEGER> | <EMPTY>

<SIGNED> ::= <DECIMAL NUMBER> <EXPONENT PART>

MORE EXAMPLES:

A) CORRECT

00100.00	100
$3_{10}+7$	$4_{10}6$
$-15._{10}-2$	

B) INCORRECT

$1.S_{10}129$	$7_{10}-90$
$3_{10}-13.0$	

1.1.2. ALPHABETIC CONSTANTS

EXAMPLES:

$ABCD$	QPX
AC	T
$)+(/$	$=$
$?!=1$	$ A $

DESCRIPTION:

IN ADDITION TO NUMERIC CONSTANTS, ALPHABETIC CONSTANTS MAY ALSO BE EMPLOYED. ANY GROUP OF AT MOST FOUR CHARACTERS ENCLOSED BY '$' SYMBOLS IS AN ALPHABETIC CONSTANT. THUS, DIG IS AN ALPHABETIC CONSTANT OF THREE CHARACTERS.

IN THE CONTEXT OF ALPHABETIC CONSTANTS A BLANK SPACE IS COUNTED AS A CHARACTER. NOTE THAT ANYWHERE ELSE IN A STATEMENT A BLANK SPACE IS IGNORED.

ALPHABETIC CONSTANTS OF LESS THAN FOUR CHARACTERS WILL BE RIGHT JUSTIFIED.

FOR AN ALPHABETIC CONSTANT ANY CHARACTER ACCEPTABLE TO THE G-20 IS ALLOWED WITH THE EXCEPTION OF THE CHARACTER '$', WHICH IS USED ONLY TO ENCLOSE THE ALPHABETIC CONSTANT. THE CHARACTERS OF AN ALPHABETIC CONSTANT ARE TRANSLATED INTO THE CORRESPONDING G-20 8-BIT INTERNAL CHARACTER CODE. (SEE PART III FOR A TABLE OF ALLOWABLE G-20 CHARACTERS, THEIR CARD PUNCH REPRESENTATION AND THEIR G-20 INTERNAL CHARACTER CODE.) THEN, AFTER TRANSLATION AN ALPHABETIC CONSTANT IS TREATED EXACTLY LIKE ANY OTHER CONSTANT.

ALPHABETIC CONSTANTS SHOULD BE ASSIGNED TO FLOATING POINT VARIABLES SINCE FIXED POINT VARIABLES DO NOT CONTAIN SUFFICIENT INFORMATION TO REPRESENT FOUR CHARACTERS.
THUS, Y1 ← $GATE$ IS CORRECT, BUT J1 ← $GATE$ WILL NOT STORE THE CONSTANT PROPERLY.

SYNTAX:

<LETTER> ::= A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z

<BASIC CHARACTER> ::= <LETTER>|<DIGIT>|←|¬|,|.|.|+|-|*|/|=|≠|<|$|>|;|(|)|↓|↑|→|^|∨|[|]|:|'

<DIGIT> ::= 0|1|2|3|4|5|6|7|8|9

<ALPHABETIC CONSTANT> ::= <ANY BASIC CHARACTER OTHER THAN $> | <ALPHABETIC CONSTANT> <ALPHABETIC CONSTANT>

<CONSTANT> ::= <NUMBER> | $<ALPHABETIC CONSTANT>$

MORE EXAMPLES:

A) CORRECT

$ A $	V
C5←$ASHU$	

B) INCORRECT

$SHFGK$	+F $
(ABCD)	J3←$ABCD$

1.2 VARIABLES

EXAMPLES:

X1	X29
Y0	J1
I2	C15

DESCRIPTION:

THERE ARE TWO CLASSES OF VARIABLES, INTEGER AND FLOATING POINT.

FLOATING POINT: A NOTATION IN WHICH ANY REAL NUMBER CAN BE REPRESENTED AS A PRODUCT OF A NUMBER AND AN EXPONENT TO SOME BASE. (IN THIS MANUAL THE NUMBER WILL ALWAYS BE DECIMAL AND THE EXPONENT WILL ALWAYS BE TO THE BASE 10 UNLESS STATED OTHERWISE.)
EXAMPLES: $4000_{,}$, -30, AND $.000002$ CAN BE EQUIVALENTLY WRITTEN AS $4*10^{3}$, $-3*10^{1}$, AND $2*10^{-6}$ RESPECTIVELY.

1.2.1. INTEGER VARIABLES

AN INTEGER VARIABLE IS DENOTED BY ONE OF THE LETTERS
'I', 'J', OR 'K'
FOLLOWED BY A SUBSCRIPT γ. (THE FORM OF ALLOWABLE SUBSCRIPTS IS GIVEN IN SECTION 2; HERE, γ REPRESENTS ANY ALLOWABLE SUBSCRIPT.) THUS, $I\gamma$, $J\gamma$, $K\gamma$ ARE INTEGER VARIABLES.

INTEGER VARIABLES MAY TAKE ON INTEGER VALUES X, WHERE $-2{,}097{,}151 < X < 2{,}097{,}151$.

IF AN INTEGER VARIABLE HAS A NUMERIC VALUE μ, THEN ITS VALUE, X, IN THE COMPUTER SATISFIES $|X| \equiv |\mu| \pmod{2^{21}}$.

1.2.2. FLOATING POINT VARIABLES

A FLOATING POINT VARIABLE IS DENOTED BY ONE OF THE LETTERS
'C', 'D', 'G', 'H', 'X', 'Y', 'Z', OR 'B'
FOLLOWED BY AN ALLOWABLE SUBSCRIPT γ.

FLOATING POINT VARIABLES MAY TAKE ON ONLY THE RANGE OF VALUES ALLOWED NUMERIC CONSTANTS. (SEE SECTION 1.1.1.)

THE CLASS '$B\gamma$' DENOTES A SPECIAL VARIABLE CLASS USED PRINCIPALLY AS SUBROUTINE PARAMETERS. THE '$B\gamma$' CLASS CAN NOT BE MENTIONED IN THE DIMENSION STATEMENT AND DOES NOT POSSESS A MATRIX REPRESENTATION. (SEE SECTION 2)

SYNTAX:

```
<INTEGER NAME TYPE> ::= I|J|K
<NAME TYPE> ::= C|D|G|H|X|Y|Z|<INTEGER NAME TYPE>
<SIMPLE SUBSCRIPT> ::= <VARIABLE> | <INTEGER> | (<EXPRESSION>)
<SUBSCRIPT> ::= <SIMPLE SUBSCRIPT>|(<EXPRESSION>,<EXPRESSION>)
<VARIABLE> ::= <NAME TYPE> <SUBSCRIPT> | B<SIMPLE SUBSCRIPT>
```

MORE EXAMPLES:

A) CORRECT

C5	Z8
G50	I10

B) INCORRECT

L2	P15
3A	

2. SUBSCRIPTS

EXAMPLES:

2, 48, I1, (J2-5) ARE ALL ADMISSIBLE SUBSCRIPTS; HENCE, Y2, Z48, I11, C(J2-5) ARE ALL ADMISSIBLE VARIABLES.

DESCRIPTION:

A SUBSCRIPT MAY BE ANY OF:

1) A NON-NEGATIVE INTEGER
2) AN INTEGER VARIABLE
3) AN EXPRESSION OF THE FORM '(β)', WHERE β IS AN EXPRESSION. (SEE SECTION 3.2)
4) A UNARY OPERATOR FOLLOWED BY A PRIMARY (SEE SECTION 3).

IN ALL CASES THE VALUE OF THE SUBSCRIPT MUST BE A NON-NEGATIVE INTEGER LESS THAN 16,384.

SINCE VARIABLES WITH THE SAME INITIAL LETTER AND CONSECUTIVE SUBSCRIPTS ARE LOCATED CONSECUTIVELY IN STORAGE, THE VARIABLES X1, X2, X3 MIGHT REPRESENT THE COMPONENTS OF A VECTOR OF LENGTH THREE.

A CONSECUTIVE BLOCK OF STORAGE CAN BE DESIGNATED AS A RECTANGULAR MATRIX STORED BY ROWS.

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 Y1 THROUGH Y9, WHERE THE FIRST ROW ($A_{1,1}$ $A_{1,2}$ $A_{1,3}$) IS STORED IN Y1, Y2, AND Y3; THE SECOND ROW ($A_{2,1}$ $A_{2,2}$ $A_{2,3}$) IS STORED IN Y4, Y5, AND Y6; AND THE THIRD ROW ($A_{3,1}$ $A_{3,2}$ $A_{3,3}$) IS STORED IN Y7, Y8, AND Y9.

IF INFORMATION AS TO THE LOCATION AND SIZE OF A MATRIX IS GIVEN THE TRANSLATOR IN THE 'DIMENSION STATEMENT' (SEE SECTION 11.1), THEN THE FOLLOWING MATRIX NOTATION MAY BE USED TO REFER TO THE ELEMENTS OF THE MATRIX.

AN ELEMENT IS REFERRED TO BY WRITING

'V(α, β)'

WHERE V IS THE PROPER LETTER (C,D,G,H,I,J,K,X,Y, OR Z) IDENTIFYING THE ARRAY, α IS THE ROW NUMBER OF THE ELEMENT, AND β IS THE COLUMN NUMBER OF THE ELEMENT. α AND β MAY BE ANY OF THE SUBSCRIPT TYPES DEFINED ABOVE, ALTHOUGH AN EXPRESSION NEED NOT BE ENCLOSED BY PARENTHESES. THUS, Y(I2+J3, J4*K2) IS CORRECT.

NOTE THAT THE LETTER 'B' CAN NOT BE USED IN DENOTING A MATRIX.

SYNTAX:

<VARIABLE> ::= <NAME TYPE> <SUBSCRIPT> | B<SIMPLE SUBSCRIPT>
<SIMPLE SUBSCRIPT> ::= <VARIABLE> | <INTEGER> | (<EXPRESSION>) |
<UNARY OPERATOR> <PRIMARY>
<SUBSCRIPT> ::= <SIMPLE SUBSCRIPT>|(<EXPRESSION>,<EXPRESSION>)

MORE EXAMPLES:

A) CORRECT

0, ↓CK5, (K8*J45-167), I(K8,J1*5), AND JI(K8,J1*5) ARE ALL ADMISSIBLE SUBSCRIPTS. HENCE THE VARIABLES YO, Y↓CK5, G(K8*J45-167), JI(K8,J1*5), AND ZJI(K8,J1*5), ARE ALL ADMISSIBLE.

IN THE MATRIX EXAMPLE ABOVE Y(1,2) OR Y2 REFERS TO $A_{1,2}$ AND Y(3,1) OR Y7 REFERS TO $A_{3,1}$.

B) INCORRECT

C5 AND *4 ARE INCORRECT SUBSCRIPTS; HENCE THE VARIABLES ZC5 AND J*4 ARE INCORRECT.

NOTE THAT DIVISION ALWAYS YIELDS A FLOATING POINT RESULT; THEREFORE, X((I1-1)/2) IS INCORRECT. HOWEVER, THE UNARY OPERATION '↓' MAY BE USED TO TRUNCATE AN ARITHMETIC EXPRESSION, AS EXPLAINED IN SECTION 3.1.2. THUS, C↓Z5 IS WELL DEFINED WHILE CZ5 IS NOT.

3. ARITHMETIC STATEMENTS

3.1 ARITHMETIC OPERATIONS

EXAMPLES:

```
Z5+X8                    I1-J7
X4/Y17                   B8*C9
K8↑3  (K8 CUBED)
A|4  (ABSOLUTE VALUE OF I4)
-I2  (NEGATION OF I2)
↓Z3  (TRUNCATION OF Z3)
```

DESCRIPTION:

3.1.1. BINARY OPERATIONS

THE USUAL BINARY OPERATION OF ADDITION (WRITTEN AS +), SUBTRACTION (-), MULTIPLICATION (*), AND DIVISION (/) ARE PROVIDED, AND WORK AS MIGHT BE EXPECTED. MULTIPLICATION MAY NOT BE DENOTED BY JUXTAPOSITION. THUS, 4X5, Z4C3, AND (X1 + 1)(X2 + 2) ARE NOT LEGAL, BUT MUST BE WRITTEN 4*X5, Z4*C3, AND (X1 + 1)*(X2 + 2), RESPECTIVELY.

THE RESULT OF DIVISION IS ALWAYS A FLOATING POINT NUMBER. THUS AN EXPRESSION CONTAINING DIVISION WILL IN GENERAL NOT PRODUCE CORRECT RESULTS IF IT IS USED AS A SUBSCRIPT, UNLESS THE VALUE OF THE EXPRESSION IS TRUNCATED BY THE '↓' OPERATOR.

THE OPERATION EXPONENTIATION (WRITTEN AS ↑) IS ALSO PROVIDED. THUS, X↑A MEANS X RAISED TO THE POWER A. IF A IS A NON-ZERO INTEGER AND |A|<11, THE OPERATION IS DONE BY |A| MULTIPLICATIONS. FOR THIS CASE THE ROUTINE IS VERY FAST. IF A<0, THE RECIPROCAL OF THE RESULT IS TAKEN. IF A=0, THE RESULT IS ONE, REGARDLESS OF X. THUS, 0↑0 = 1. IF X=0 AND A>0 THE RESULT IS 0. FOR ALL OTHER LEGAL CASES X↑A IS TAKEN AS EXP.(A * LOG.(X)). THE CASE 0↑A WHERE A<0 IS AN ERROR AND STOPS THE PROGRAM.

3.1.2. UNARY OPERATIONS

ALSO AVAILABLE ARE FIVE UNARY OPERATIONS:

+	PLUS (UNNECESSARY)
-	NEGATION
A	ABSOLUTE VALUE
↓	TRUNCATION
L	LOCATION.

EACH OF THESE OPERATIONS OPERATES ON THE PRIMARY IMMEDIATELY TO ITS RIGHT. THE FIRST THREE OPERATIONS OPERATE AS EXPECTED. ↓ CONVERTS FLOATING POINT VALUES TO INTEGER FORM. THE OPERATION IS ONE OF TRUNCATION AND REDUCTION MODULO 2↑32. (2↑32 = 4294967296)

THE OPERATOR 'L', WHICH MAY ONLY BE APPLIED TO A VARIABLE OR A CONSTANT, REFERS TO THE LOCATION OF THE VARIABLE OR CONSTANT. THUS LX3 IS THE LOCATION (I.E., MACHINE ADDRESS) WHERE X3 IS STORED. L3, LZ11, OR LC(I1 + 1, 3) ARE PERMITTED. 'L' IS USUALLY USED ONLY IN FUNCTION PARAMETERS. (SEE SECTION 4)

MORE EXAMPLES:

A) CORRECT

```
H16+Z2                    I10*K0
X5/-X3
Y4↑Z8   (Y4 RAISED TO THE POWER Z8)
Z1↑-.5  (THE RECIPROCAL OF THE SQUARE ROOT OF Z1)
```

B) INCORRECT

```
I1Z5                      Y3A
D2*/C9
```

3.2 ARITHMETIC EXPRESSIONS

EXAMPLES:

```
C(J2+I1)                  A(I4-20)+G3
D2/(C1-X4)
I4+I6-J2+I7+K9      MEANS  (((I4+I6)-J2)+I7)+K9
```

DESCRIPTION:

AN ARITHMETIC EXPRESSION IS A RULE FOR COMPUTING A NUMERICAL VALUE. THIS VALUE IS OBTAINED BY EXECUTING THE INDICATED ARITHMETIC OPERATIONS ON THE ACTUAL NUMERICAL VALUES OF THE VARIABLES AND CONSTANTS OF THE EXPRESSION. (FOR VARIABLES THE ACTUAL NUMERICAL VALUE IS THE CURRENT VALUE.)

WITHIN AN ARITHMETIC EXPRESSION THE RESULTS OF ALL OPERATIONS (EXCEPT ↓) ARE OBTAINED IN FLOATING POINT; INTEGER ARITHMETIC IS ONLY USED FOR STORAGE OF THE RESULT OF A CALCULATION INTO AN INTEGER VARIABLE. THUS, THE EXPRESSION 'I1 ← Z5*2.+J1' IS CALCULATED IN FLOATING POINT ARITHMETIC; THEN THE RESULT IS TRUNCATED TO AN INTEGER AND IS STORED IN I1.

THE SEQUENCE OF OPERATIONS WITHIN ONE EXPRESSION IS GENERALLY FROM LEFT TO RIGHT, CONTROLLED BY THE FOLLOWING GENERAL RULES OF PRECEDENCE:

1) SUBSCRIPTION AND FUNCTION EVALUATION; BEFORE
2) ABSOLUTE VALUE, NEGATION, AND TRUNCATION; BEFORE
3) EXPONENTIATION; BEFORE
4) MULTIPLICATION AND DIVISION; BEFORE
5) ADDITION AND SUBTRACTION; BEFORE
6) STORING.

THE DESIRED ORDER OF EXECUTION OF OPERATIONS WITHIN AN EXPRESSION CAN ALWAYS BE ARRANGED BY APPROPRIATE POSITIONING OF PARENTHESES, AS IN USUAL ALGEBRAIC NOTATION. WHEN AN ARITHMETIC EXPRESSION IS ENCLOSED IN PARENTHESES ITS VALUE IS OBTAINED THROUGH A RECURSIVE ANALYSIS; THUS, THE EXPRESSION BETWEEN A LEFT PARENTHESIS AND THE MATCHING RIGHT PARENTHESIS IS EVALUATED BY ITSELF AND THIS VALUE IS THEN USED IN SUBSEQUENT CALCULATIONS.

A SUBSTITUTION STATEMENT IN PARENTHESES MAY BE USED AS AN ELEMENT OF AN EXPRESSION. IN THIS CASE, THE SUBSTITUTION WILL BE PERFORMED AND THE VALUE OF THE PARENTHESIZED SUBSTITUTION STATEMENT WILL BE THE SAME AS THE EXPRESSION TO THE RIGHT OF THE '←'.

THUS, THE STATEMENT

```
Z1 ← Z1 * Z1 * (Z1 ← SIN.(X1))
```

WILL STORE IN Z1 THE CUBE OF SIN.(X1). HOWEVER, IT WILL RUN IN LESS TIME THAN THE STATEMENT

```
Z1 ← SIN.(X1)↑3.
```

IN A STATEMENT SUCH AS

```
Z1 ← (I1 ← 3.5)
```

Z1 WILL BE SET TO 3.5, NOT TO 3.

THE STATEMENT

```
X1 ← X2 ← X3 ← 0
```

IS ILLEGAL, AND WILL NOT WORK SINCE THE SUBSTITUTION STATEMENTS ARE NOT IN PARENTHESES. THE DESIRED EFFECT MAY BE ACHIEVED BY

```
X1 ← (X2 ← (X3 ← 0))
```

SYNTAX:

<PRIMARY> ::= <NUMBER> | <VARIABLE> | <FUNCTION DESIGNATOR> | L <VARIABLE>|L <CONSTANT>|(<SUBSTITUTION STATEMENT>)| (<EXPRESSION>)|<UNARY OPERATOR> <PRIMARY>

<FACTOR> ::= <PRIMARY> | <FACTOR>↑<PRIMARY>

<TERM> ::= <FACTOR> | <TERM> <MULTIPLYING OPERATOR> <FACTOR>

<EXPRESSION> ::= <TERM> | <EXPRESSION> <ADDING OPERATOR> <TERM>

<UNARY OPERATOR> ::= + | - | A | ↓

<ADDING OPERATOR> ::= + | -

<MULTIPLYING OPERATOR> ::= * | /

MORE EXAMPLES:

A) CORRECT

1) I7↑I2*K2+J1 MEANS ((I7↑I2)*K2)+J1
2) H2/C2-Z4 MEANS (H2/C2)-Z4
3) AI4+J2*K1 MEANS |I4|+(J2*K1)
4) I7+H2*Z4/C2+K9*I0 MEANS (I7+((H2*Z4)/C2))+(K9*I0)
5) D2*D7/H4/Z5*Y2 MEANS (((D2*D7)/H4)/Z5)*Y2
6) I6+K2*Z8↑I4/C3-K2↑-J2*K4+I8 MEANS ((I6+((K2*(Z8↑I4))/C3))-((K2↑-J2)*K4))+I8
7) ((I1+Z5))+D1

B) INCORRECT

```
J(I5+I6
(I2*C0)+J7-I1))
D1+D2*D3   MEANING  (D1+D2)*D3
-I2+I4     MEANING  -(I2+I4)
AX1*Y1     MEANING  |X1*Y1|
```

3.3 SUBSTITUTION STATEMENTS

EXAMPLES:

```
K7 ← J4*I5          Y4 ← I2+Z4*5.96
```

DESCRIPTION:

THE FORM OF A SUBSTITUTION STATEMENT IS

'$v \gamma \leftarrow \beta$'

WHERE $v\gamma$ IS ANY VARIABLE AND β ANY ARITHMETIC EXPRESSION. AFTER THE EXECUTION OF THIS STATEMENT THE VALUE OF $v\gamma$ BECOMES THE VALUE OF β.

3.3.1. ARITHMETIC OF SUBSTITUTION STATEMENTS

A VARIABLE NEVER CHANGES ARITHMETIC. THEREFORE, WHEN A VARIABLE IS TO TAKE ON A NEW VALUE AS A RESULT OF A SUBSTITUTION STATEMENT, THE VALUE OF THE EXPRESSION TO THE RIGHT OF THE '←' SYMBOL IS COMPUTED IN FLOATING POINT ARITHMETIC AND IS CONVERTED, IF NECESSARY, TO THE ARITHMETIC OF THE VARIABLE TO THE LEFT OF THE '←' BEFORE SUBSTITUTION.

IN A STATEMENT SUCH AS I2 ← K2+Z8, THE QUANTITY K2+Z8 IS COMPUTED IN FLOATING POINT ARITHMETIC BUT IS CONVERTED TO AN INTEGER BEFORE ITS VALUE IS ASSIGNED TO I2.

THIS CONVERSION TO AN INTEGER IS ONE OF TRUNCATION, AND NOT OF ROUNDING. THUS, THE CONVERTED VALUE OF THE EXPRESSION, β, IS

$$SGN(\beta) * [|\beta|](MOD\ 2\uparrow 32) .$$

THIS IS BEST ILLUSTRATED BY THE FOLLOWING TABLE OF VALUES OF I1 STORED BY THE STATEMENT I1←Y1, WHERE Y1 IS AS INDICATED.

Y1	I1
1.9	1
-11.8	-11
13.1	13
4294967297	1

SYNTAX:

<SUBSTITUTION STATEMENT> ::= <VARIABLE>←<EXPRESSION>

MORE EXAMPLES:

A) CORRECT

```
C9 ← (Y4+I1)*Y1↑-.4
```

B) INCORRECT

```
I1 ← J2*Z5 ← Y0+J2
C1 ← C2 ← C3 ← 0.
```

4. SUBROUTINES

EXAMPLES:

COS.(Y4) REFERS TO THE VALUE OF THE COSINE OF Y4.
SQRT.(X1) REFERS TO THE VALUE OF THE SQUARE ROOT OF X1.

IF XGCD IS THE NAME OF A GREATEST COMMON DIVISOR SUBROUTINE, THEN XGCD.(I1,J1) REFERS TO THE VALUE OF THE GREATEST COMMON DIVISOR OF I1 AND J1.

DESCRIPTION:

SUBROUTINES ARE PRECODED SUBPROGRAMS WHICH COMPUTE FREQUENTLY USED FUNCTIONS (E.G., SQUARE ROOT, SINE, COSINE, ARCTANGENT) OR ACCOMPLISH OTHER FREQUENTLY NEEDED ROUTINES.

THE CODER MAY CALL FOR ANY OF THE SUBROUTINES FOUND IN THE LIBRARY (SEE APPENDIX B) BY EMPLOYING THE FORM

'NAME.(β_1, β_2, ..., β_K)'

(THE COMMAS AND THE PERIOD FOLLOWING 'NAME' MUST BE WRITTEN) WHERE NAME IS THE ENTRY TO THE SUBROUTINE AND CONTAINS UP TO FIVE ALPHANUMERIC CHARACTERS, K IS THE NUMBER OF ARGUMENTS REQUIRED BY THE SUBROUTINE, AND (THE SEQUENCE) β_1, β_2, ..., β_K ARE THE ARGUMENTS.

NOTE FROM THE SYNTAX THAT A FUNCTION NAME MUST BE BOUNDED ON THE LEFT BY SOME CHARACTER WHICH IS NOT A LETTER OR A DIGIT.

THE EXPRESSION 'NAME.(β_1, β_2, ..., β_K)' IS A SHORTHAND NOTATION FOR 'THE RESULT OF SUBROUTINE NAME' AND IS TREATED IN THE SAME WAY AS A VARIABLE. THEREFORE, THE TERM 'ARITHMETIC EXPRESSION' INCLUDES SUBROUTINES AS WELL AS CONSTANTS AND VARIABLES.

SOME SUBROUTINE EXPRESSIONS, SUCH AS THOSE THAT PRODUCE MORE THAN ONE RESULT, (E.G., MATRIX INVERSION, INTEGRATION BY SIMPSON'S RULE) MAY COMPRISE A COMPLETE STATEMENT. THIS TYPE OF SUBROUTINE SOMETIMES HAS FOR ONE OF ITS ARGUMENTS THE LOCATION OF SOME VARIABLE (INSTEAD OF THE VALUE OF THE VARIABLE). TO REFER TO THE LOCATION OF A VARIABLE, PREFIX THE VARIABLE NAME BY THE LETTER 'L'.

SYNTAX:

<FUNCTION DESIGNATOR> ::= <FUNCTION NAME>.(<PARAMETER PART>)
<PARAMETER PART> ::= <EXPRESSION>|<PARAMETER PART>,<EXPRESSION>
<FUNCTION NAME> ::= <LETTER>|<DIGIT>|<FUNCTION NAME> <LETTER>|
<FUNCTION NAME> <DIGIT>
<SUBROUTINE STATEMENT> ::= <FUNCTION DESIGNATOR>

MORE EXAMPLES:

A) CORRECT

```
X3 ← 1.3; X4 ← 9  IF (SIN.(Y1)) < .5
```

THE STATEMENT VADD.(3,LY2,LX5,LZ7) MIGHT BE A VECTOR ADDITION ROUTINE WITH THE FOLLOWING ARGUMENTS:

3 = THE NUMBER OF ELEMENTS IN THE VECTORS

LY2 = Y2 IS THE FIRST ELEMENT OF VECTOR A=(Y2,Y3,Y4)

LX5 = X5 IS THE FIRST ELEMENT OF VECTOR B=(X5,X6,X7)

LZ7 = Z7 IS THE LOCATION IN WHICH THE FIRST ELEMENT OF THE SUM IS TO BE STORED, I.E., THE VECTOR A+B = (Y2+X5, Y3+X6, Y4+X7) IS STORED IN (Z7, Z8, Z9).

B) INCORRECT

```
SIN(X3)    (NO PERIOD FOLLOWS 'SIN')
VADD.(3,LX1,LY1)    (WHERE VADD IS AS ABOVE; THIS
```

EXAMPLE CONTAINS THE WRONG NUMBER OF ARGUMENTS.)

```
GO TO 5  IF  SQRT.(Y1) > 7        (THIS WILL RESULT
```

IN A CALL FOR A SUBROUTINE NAMED GO TO 5.)

5. CONTROL STATEMENTS

STATEMENTS WHICH REFER TO OTHER STATEMENTS (BY THEIR STATEMENT NUMBER) ARE CALLED CONTROL STATEMENTS. EVERY STATEMENT THAT IS REFERRED TO MUST HAVE A UNIQUE POSITIVE INTEGER (<1000) WHICH IDENTIFIES IT. THESE NUMBERS NEED NOT BE IN ORDER, NOR DO ALL INTEGERS BELOW A GIVEN NUMBER HAVE TO BE USED. STATEMENTS WHICH ARE NOT REFERRED TO DIRECTLY NEED NOT BE NUMBERED.

5.1 TRANSFER STATEMENTS

EXAMPLES:

```
GO TO 4
GO TO J3
GO TO  16+3*K13
```

DESCRIPTION:

THE NORMAL SEQUENCE OF COMPUTATION IS FROM ONE STATEMENT TO THE NEXT STATEMENT IN THE ORDER IN WHICH THEY ARE WRITTEN. TO VARY THIS SEQUENTIAL EXECUTION OF STATEMENTS THE FOLLOWING STATEMENT MAY BE WRITTEN:

'GO TO φ '

WHERE φ IS AN ARITHMETIC EXPRESSION WHOSE VALUE IS POSITIVE AND LESS THAN 1000 IN VALUE.

THE EXPRESSION IS ALWAYS CONVERTED TO INTEGER VALUE.

THE EXECUTION OF THE STATEMENT 'GO TO φ ' MEANS 'EXECUTE NEXT THE STATEMENT WHOSE NUMBER IS EQUAL TO THE VALUE OF φ '. THUS, A STATEMENT SUCH AS 'GO TO K6' MEANS 'EXECUTE NEXT THE STATEMENT WHOSE NUMBER IS EQUAL TO THE (CURRENT) VALUE OF K6'. (THE VALUE OF K6 MAY BE CHANGED DURING THE EXECUTION OF THE PROGRAM; HENCE, WHENEVER THE STATEMENT 'GO TO K6' IS EXECUTED, THE SEQUENCE OF COMPUTATION MAY PROCEED TO ONE OF SEVERAL DIFFERENT PLACES IN THE PROGRAM DEPENDING ON THE CURRENT VALUE OF K6. —— THIS STRUCTURE SERVES AS A VARIABLE SEQUENCING 'SWITCH'.)

SYNTAX:

<GO TO STATEMENT> ::= GO TO <EXPRESSION>

MORE EXAMPLES:

A) CORRECT

```
GO TO K3+J9
```

B) INCORRECT

```
GO TO 1900
```

5.2. ITERATION STATEMENTS

EXAMPLES:

```
        2, I1, 0, 1, 20,
    2:  I3 ← I3 + I1 * Z0
```

THIS IS EQUIVALENT TO:

```
        I1 ← 0
    3:  GO TO 4 IF I1 > 20
    2:  I3 ← I3 + I1 * Z0
        I1 ← I1 + 1
        GO TO 3
    4:  ...
```

(HERE 3 AND 4 ARE STATEMENT NUMBERS NOT USED OTHERWISE IN THE PROGRAM.)

DESCRIPTION:

IN MOST PROGRAMS THERE ARE USUALLY GROUPS OF STATEMENTS WHICH ARE TO BE REPEATED MANY TIMES (ITERATED). USUALLY SOME VARIABLE IS TO BE REGULARLY MODIFIED FOR EACH ITERATION, AND THE ITERATION IS TO BE TERMINATED AFTER THE VARIABLE REACHES A CERTAIN VALUE. AN ITERATION STATEMENT HAS THE FORM

```
M:  K, V, A, B, C,
```

(ALL FIVE COMMAS MUST BE WRITTEN.) M IS THE LABEL OF THE ITERATION STATEMENT. K IS THE LABEL OF THE LAST STATEMENT OF THE BLOCK CONTROLLED BY THE ITERATION STATEMENT; IT MUST BE AN INTEGER. V IS THE VARIABLE WHOSE VALUE IS TO BE SYSTEMATICALLY ALTERED BY THE ITERATION STATEMENT; AND A, B, AND C ARE ALL EXPRESSIONS. A IS THE INITIAL VALUE; I.E., THE VALUE TO BE ASSIGNED TO V BEFORE THE CONTROLLED STATEMENTS ARE EXECUTED THE FIRST TIME. B IS THE INCREMENT: BETWEEN ANY TWO REPETITIONS OF THE CONTROLLED STATEMENTS V IS INCREMENTED BY B. C IS THE FINAL VALUE. THE BODY IS EXECUTED AS LONG AS V IS NOT GREATER THAN C. WHEN THIS CONDITION NO LONGER HOLDS THE ITERATION STATEMENT TERMINATES. THIS MAY BEST BE DESCRIBED BY AN EXAMPLE.

CONSIDER THE FOLLOWING STATEMENT SEQUENCE:

```
M:  K, V, A, B, C,
    . . .
K:  STATEMENT K
P:  STATEMENT P
```

THIS SEQUENCE IS EQUIVALENT TO THE FOLLOWING SEQUENCE, WHERE N IS A LABEL NOT USED ELSEWHERE IN THE PROGRAM:

```
M:  V ← A
N:  GO TO P IF V>C
    . . .
K:  STATEMENT K
    V ← V + B
    GO TO N
P:  STATEMENT P
```

THERE ARE SEVERAL POINTS TO NOTE HERE:

1) IF A>C, THE BODY IS NOT EXECUTED AT ALL, SINCE THE TEST IN STATEMENT N IS SATISFIED THE FIRST TIME IT IS MADE. THIS IS A CHANGE FROM 650-GATE, IN WHICH THE CONTROLLED STATEMENTS WERE ALWAYS EXECUTED AT LEAST ONCE.

2) AFTER THE COMPLETION OF THE ITERATION STATEMENT V WILL HAVE A VALUE ONE INCREMENT BEYOND THE VALUE IT HAD THE LAST TIME THE BODY WAS EXECUTED. THUS, IN THE EXAMPLE AT THE BEGINNING OF THIS SECTION I1 WILL BE 21 AT COMPLETION.

3) IF C-A IS AN INTEGRAL MULTIPLE OF B, THE BODY WILL, IN GENERAL, BE EXECUTED EXACTLY (C-A)/B + 1 TIMES. THIS WILL WORK SATISFACTORILY AS LONG AS A, B, AND C ARE INTEGER VALUED. THERE ARE, HOWEVER, EXCEPTIONS.
CONSIDER THE STATEMENT

```
2, X1, 0, .01, 1,
```

THE NUMBER .01 CAN ONLY BE APPROXIMATED INTERNALLY IN A BINARY MACHINE SUCH AS THE G-20. THUS, BECAUSE OF ROUNDOFF ERROR, IT IS POSSIBLE THAT THE NUMBER STORED IN THE G-20 EXCEEDS .01. (INDEED, THIS WILL BE THE CASE FOR ABOUT HALF OF ALL POSSIBLE CONSTANTS.) IN THAT CASE THE BODY WOULD BE EXECUTED ONLY 100 TIMES, INSTEAD OF THE 101 TIMES EXPECTED.
THE EASIEST WAY TO AVOID THIS DIFFICULTY IN FLOATING POINT ITERATIONS IS TO INCREASE C BY B/2. THUS, THE ABOVE STATEMENT MIGHT WELL BE REPLACED BY

```
2, X1, 0, .01, 1.005,
```

TO INSURE PROPER OPERATION.

4) THE CASE K=M IS ALLOWED, WHERE THE FINAL STATEMENT OF THE BODY IS THE ITERATION STATEMENT ITSELF. IN THAT CASE, THE INCREMENTING AND TESTING CONTAINED IN THE ITERATION STATEMENT ITSELF CONSTITUTE THE SCOPE OF ITERATION.

5) IT IS CLEAR THAT THE SCHEME DESCRIBED ABOVE WILL NOT WORK IF B<0, SINCE IN THIS CASE C WILL BE LESS THAN A AND THE BODY WILL, ACCORDING TO (1) ABOVE, BE EXECUTED ZERO TIMES. HOWEVER, THIS POSSIBILITY IS PROVIDED FOR. IF THE FIRST CHARACTER OF THE INCREMENT IS - (MINUS SIGN), THEN THE TEST IN STATEMENT N (SEE PAGE 13) IS V<C INSTEAD OF V>C.
IT IS THUS CLEAR THAT IF AN ACTUAL DECREMENT IS TO OCCUR AT RUN TIME, B AT COMPILE TIME MUST START WITH A MINUS SIGN. IN SUMMARY, THEN, THE TEST PRODUCED BY THE ITERATION STATEMENT IS ALWAYS

```
GO TO P  IF  (V-C) * SGN(B) > 0
```

WHERE SGN(B) IS DETERMINED AT TRANSLATE TIME AND IS +1 UNLESS THE FIRST CHARACTER OF B IS MINUS, IN WHICH CASE IT IS -1.

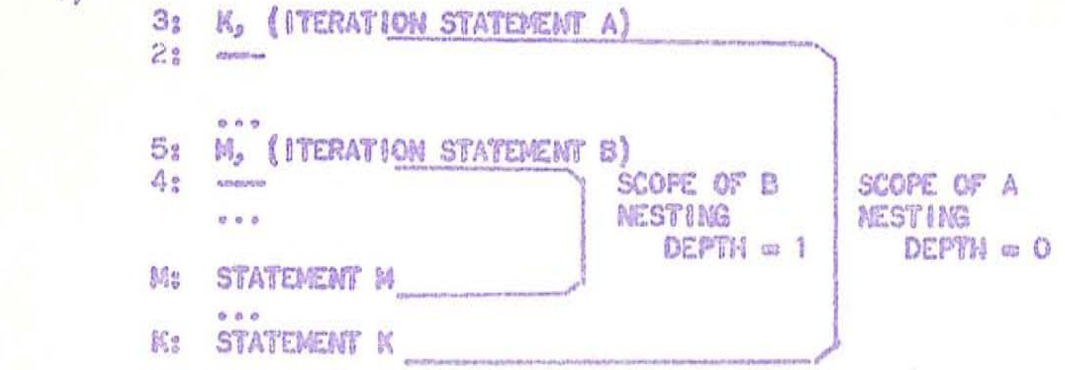
5.2.1. NESTING OF ITERATION STATEMENTS

THE BLOCK OF STATEMENTS SPECIFIED BY AN ITERATION STATEMENT FOR REPEATED EXECUTION IS CALLED THE SCOPE OF ITERATION.

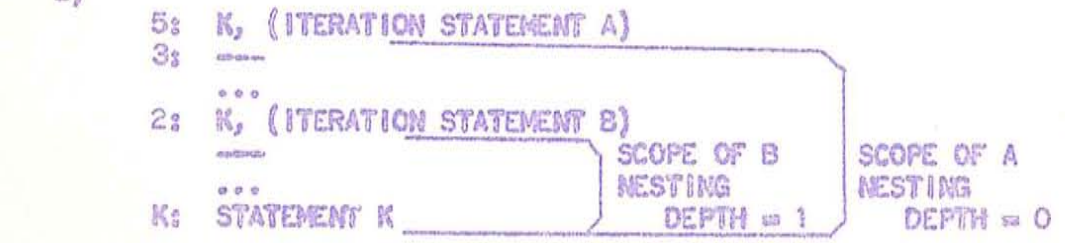
```
4:  K, V, A, B, C,          ITERATION STATEMENT
2:  STATEMENT 2 }
    . . .       }           SCOPE OF ITERATION
K:  STATEMENT K }
```

EXAMPLES:

1)



2)



DESCRIPTION:

SOME OF THE STATEMENTS WITHIN THE SCOPE OF ITERATION MAY BE ITERATION STATEMENTS. HOWEVER, IF THE ITERATION STATEMENT B (AS IN THE 'EXAMPLES' ABOVE) IS IN THE SCOPE OF ANOTHER ITERATION STATEMENT A, THEN THE SCOPE OF B MUST BE ENTIRELY WITHIN THE SCOPE OF A.

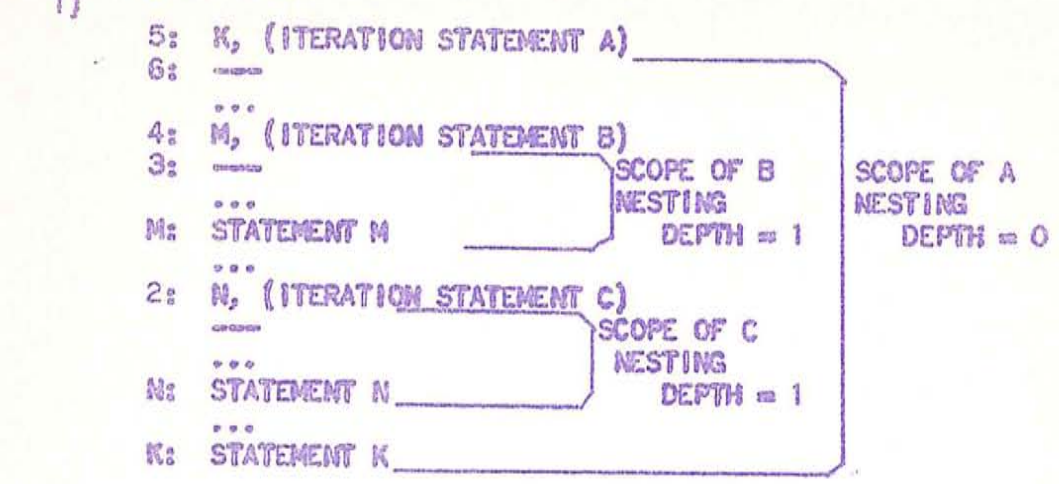
IN EXAMPLE 2), EVEN THOUGH THE SCOPES OF A AND B BOTH END ON STATEMENT K, THE ITERATION OF B IS INCREMENTED AND TESTED FIRST; HENCE, THE ITERATION OF B IS COMPLETED BEFORE THE ITERATION OF A IS INCREMENTED AND THE EXAMPLE IS CORRECT.

WHEN ITERATION STATEMENTS OCCUR IN THE SCOPE OF OTHER ITERATION STATEMENTS THEY ARE SAID TO BE 'NESTED'. THE 'NESTING DEPTH' OF AN ITERATION STATEMENT IS THE NUMBER OF ITERATION STATEMENTS IN WHOSE SCOPE IT APPEARS. THIS DEPTH MAY NOT EXCEED EIGHT.

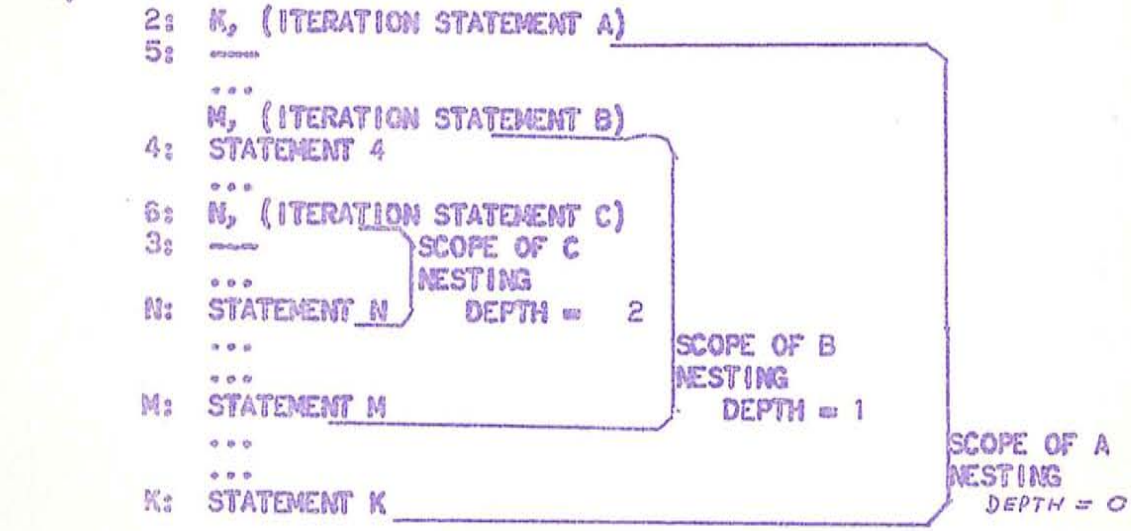
MORE EXAMPLES:

A) CORRECT

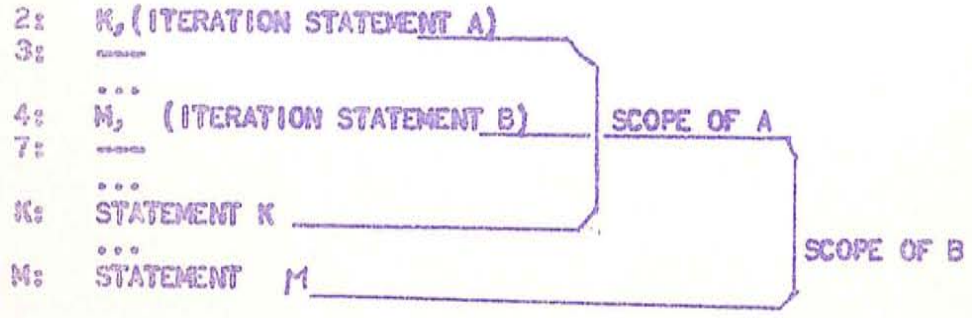
1)



2)



B) INCORRECT



SYNTAX:

<ITERATION> ::= <LABEL> ↓1<INTEGER>,<ITERATION PART>
<STATEMENT SEQUENCE> ↑1: <SIMPLE STATEMENT> |
↓2 <INTEGER>: ↑2 , <ITERATION PART>
<ITERATION PART> ::= <VARIABLE>,<EXPRESSION>,<EXPRESSION>,
<EXPRESSION>,
<STATEMENT SEQUENCE> ::= <CS>|<STATEMENT SEQUENCE> <STATEMENT>

MORE EXAMPLES:

1) THE PROGRAM SEGMENT,

```
Z1 ⇐ 0.
Z2 ⇐ 1.
4, K1, 1, 1, K2,
Z1 ⇐ Z1 + XK1
4: Z2 ⇐ Z2 * YK1
```

ACCUMULATES THE SUM OF THE K2 NUMBERS X1, X2, ...,XK2 AND THE PRODUCT OF THE K2 NUMBERS Y1, Y2, ..., YK2.

2) GIVEN A VECTOR X1, X2, ..., XK1 THE FOLLOWING PROGRAM SEGMENT COMPUTES THE SQUARE ROOT OF THE SUM OF THE LAST J1 ELEMENTS.

```
4: X0 ⇐ 0
   2,I1, K1-J1+1, 1, K1,
2: X0 ⇐ X0 + XI1
   X0 ⇐ SQRT.(X0)
```

3) EVALUATE THE POLYNOMIAL $C_N X^N + C_{N-1} X^{N-1} + \ldots + C_1 X + C_0$ USING THE FORMULA

$$(\ldots((C_N X + C_{N-1})X + C_{N-2})X + \ldots + C_1)X + C_0.$$

LETTING Y1 = VALUE OF THE POLYNOMIAL, X1 = X, K1 = N, AND C_N = CK1, C_{N-1} = C(K1-1),...,C_1 = C1,C_0 = C0, THE FOLLOWING PROGRAM SEGMENT WILL EVALUATE THIS POLYNOMIAL:

```
Y1 ⇐ 0
2, K2, K1, -1, 0,
2: Y1 ⇐ X1 * Y1 + CK2
```

4) A NEWTON'S METHOD SOLUTION OF THE EQUATION
F(X) = X↑3 + X - 1 = 0
COULD BE WRITTEN AS A SINGLE STATEMENT IF THE CRITERION FOR STOPPING THE ITERATION IS $|F(X)| < \epsilon$:

```
2: 1, X1, X0, -(X1↑3+X1-1.)/(3.*X1↑2+1.),
   X1-(X1↑3+X1-1.)÷Y1,
```

WHERE X0 IS THE INITIAL GUESS AND Y1 = ϵ .

6. COMMENT STATEMENT

ANY STATEMENT WHOSE FINAL CHARACTER IS THE LETTER 'C' WILL BE TREATED AS A COMMENTS STATEMENT. SUCH A STATEMENT PRODUCES NO CODE. HOWEVER, IF IT IS NUMBERED IT CAN BE USED TO TERMINATE AN ITERATION LOOP.

SYNTAX:

<COMMENT STATEMENT> ::= C|<BASIC CHARACTER> <COMMENT STATEMENT>

EXAMPLES:

```
GO TO 3 IF Y1 = 0.  C
CC
COMMENT C
```

7. COMPOUND STATEMENTS

EXAMPLES:

```
N:  I2 ← 5 ;  C3 ← D5 ;  X8 ← 0.
```

THIS IS COMPILED AS IF THE PROGRAMMER HAD WRITTEN

```
N:  X8 ← 0.
    C3 ← D5
    I2 ← 5
```

DESCRIPTION:

SEVERAL STATEMENTS MAY BE COMBINED TO FORM A SINGLE STATEMENT BY THE USE OF THE COMBINATION SYMBOL ';' (WHICH MAY BE READ AS 'AND').

WARNING: THE ORDER OF STATEMENT EXECUTION IS FROM THE RIGHTMOST LISTED ONE TO THE LEFTMOST LISTED ONE.

IT IS POSSIBLE TO COMBINE A COMMENTS STATEMENT WITH ANY OTHER STATEMENT. THUS, THE EFFECT OF A STATEMENT SUCH AS

```
N:  I1 IS N C ; READ IF I1 > 0 ; I1←I1+1 IF C1=X1
```

IS AS FOLLOWS:

IF C1 DIFFERS FROM X1, CONTROL PASSES TO THE NEXT LISTED STATEMENT. IF NOT, I1 IS INCREMENTED BY ONE AND THEN COMPARED TO ZERO. IF I1>0, THE READ SUBROUTINE IS ENTERED; IF NOT, CONTROL PASSES TO THE NEXT STATEMENT.

SYNTAX:

```
<BASIC STATEMENT> ::= <GO TO STATEMENT>|<HALT STATEMENT>|
    <COMMENT STATEMENT>|<CONTINUABLE STATEMENT>
<CONTINUABLE STATEMENT> ::= <SUBSTITUTION STATEMENT> |
    <SUBROUTINE STATEMENT> | <INPUT-OUTPUT STATEMENT>
<SIMPLE STATEMENT> ::= <BASIC STATEMENT>|<COMPOUND STATEMENT>|
    <CONDITIONAL STATEMENT>
<COMPOUND STATEMENT> ::= <SIMPLE STATEMENT> |
    <COMPOUND STATEMENT> ; <CONTINUABLE STATEMENT>
```

MORE EXAMPLES:

A) CORRECT

```
    I1 ← 1
2:  GO TO 2 IF I1 < 5 ; I1 ← I1+1 ;  CI1 ← I1
```

THIS IS EQUIVALENT TO THE STATEMENTS

```
    2, I1, 1, 1, 5,
2:  CI1 ← I1
```

B) INCORRECT

```
N:  I1 ← I1+1 ; GO TO 4
```

(THIS WILL NOT CAUSE I1 TO BE INCREMENTED. ONLY THE TRANSFER TO STATEMENT 4 WILL BE EXECUTED.)

8. INPUT-OUTPUT STATEMENTS

8.1 INPUT STATEMENTS

TO READ INTO STORAGE EITHER NUMERIC OR ALPHABETIC DATA THE STATEMENT 'READ' IS WRITTEN. THIS STATEMENT WILL CAUSE THE NEXT 'READ GROUP' OF CARDS IN THE READ HOPPER OF THE G-20 TO BE READ INTO STORAGE. THE TYPE OF DATA IS INDICATED ON THE CARD AND EACH VALUE IS ACCOMPANIED BY ITS NAME; THEREFORE, NO OTHER INFORMATION IS NEEDED IN THE INPUT STATEMENT.

FOR DATA CARD FORMAT SEE
'HOW TO USE THE SYSTEM'.

SYNTAX:

<INPUT STATEMENT> ::= READ

8.2. OUTPUT STATEMENTS

CONSTANTS AND COMPUTED VALUES OF VARIABLES ARE AVAILABLE TO THE PROGRAMMER AS PRINTED OUTPUT.

THE OCCURRENCE OF AN OUTPUT STATEMENT IN THE PROGRAM WILL CAUSE THE PRINTING OF DESIGNATED VARIABLES AND CONSTANTS ALONG WITH THEIR CORRESPONDING CURRENT VALUES.

IT SHOULD BE NOTED THAT PRINTING OUTPUT GREATLY LESSENS THE SPEED OF COMPUTATION OF A PROGRAM. THEREFORE, PROGRAMMERS SHOULD TAKE CARE TO HAVE ONLY NECESSARY OUTPUT PRINTED.

(SEE 'HOW TO USE THE SYSTEM' FOR SOME SAMPLE OUTPUT.)

SYNTAX:

<OUTPUT STATEMENT> ::= <ALPHABETIC OUTPUT STATEMENT> |
<NUMERIC OUTPUT STATEMENT>
<ALPHABETIC OUTPUT STATEMENT> ::= A<NUMERIC OUTPUT STATEMENT>
<NUMERIC OUTPUT STATEMENT> ::= <OUTPUT ELEMENT> |
<NUMERIC OUTPUT STATEMENT> <OUTPUT ELEMENT>
<OUTPUT ELEMENT> ::= T<CONSTANT> | T<VARIABLE> |
T<VARIABLE>...<VARIABLE>

8.2.1. NUMERIC OUTPUT STATEMENTS

TO PRINT NUMERIC DATA THE STATEMENT
'Ta_1 Ta_2 ... Ta_k'

IS WRITTEN. HERE THE ELEMENTS OF THE SEQUENCE $a_1, a_2, \ldots, a_k$

MAY BE ANY VARIABLES (WITH ANY TYPES OF SUBSCRIPTS) OR ANY NUMERIC CONSTANTS, BUT NOT GENERAL EXPRESSIONS. K, WHICH IS THE NUMBER OF VARIABLES AND/OR CONSTANTS SPECIFIED IN THE STATEMENT, MUST BE LESS THAN 25.

SYNTAX:

<NUMERIC OUTPUT STATEMENT> ::= T<OUTPUT UNIT>|T<OUTPUT UNIT> <NUMERIC OUTPUT STATEMENT>

EXAMPLES:

A) CORRECT

```
TY4 TK(I1,I2) T4
```

(THIS WILL CAUSE THE PRINTING OF THE VALUES OF Y4, K(I1, I2), AND THE CONSTANT 4, ALONG WITH THE NAMES OF THE VARIABLES. THE NAME OF K(I1, I2) WILL BE K WITH THE CORRESPONDING SINGLE NUMERIC SUBSCRIPT.)

```
TZK8
```

(THIS STATEMENT, WITH K8=4, WILL CAUSE THE COMPUTER TO PRINT THE VALUE OF Z4 AND THE NAME 'Z4'.)

B) INCORRECT

```
T I1+I2
T (J1 * X3)
```

8.2.2. ALPHABETIC OUTPUT STATEMENTS

TO PRINT ALPHABETIC DATA THE STATEMENT

'AT β_1 T β_2 ... T β_K'

IS WRITTEN. HERE THE ELEMENTS OF THE SEQUENCE $\beta_1, \beta_2, \ldots, \beta_K$ MAY BE ANY FLOATING POINT VARIABLES OR ALPHABETIC CONSTANTS, AND THE NUMBER K MUST BE LESS THAN 25.

SYNTAX:

<ALPHABETIC OUTPUT STATEMENT> ::= A<NUMERIC OUTPUT STATEMENT>

EXAMPLES:

A) CORRECT

```
AT Y4 T$LIFT$
```

(THIS STATEMENT WILL PRINT THE VALUE OF Y4 AS A FOUR-CHARACTER ALPHABETIC CONSTANT AND THE ALPHABETIC CONSTANT 'LIFT'.)

```
ATY4 TX3 TZO
ATY1...Y10
AT $DIG$ T $GATE$ TC5
```

B) INCORRECT

```
ATJ1 TX3 T18
```

8.2.3. REGIONAL OUTPUT

EITHER OF THE ALPHABETIC OR NUMERIC OUTPUT STATEMENTS MAY SPECIFY A REGION TO BE PRINTED RATHER THAN A SINGLE VARIABLE. WRITING

'TVφ_1 ... Vφ_2 '

(THE THREE DOTS MUST BE WRITTEN.) WILL PRINT THE VALUES Vφ_1 , V(φ_1 +1), V(φ_1 +2),...,Vφ_2 IN SEQUENCE. HERE φ_1 AND φ_2 MAY BE ANY INTEGER EXPRESSION, BUT φ_2 MUST BE GREATER THAN φ_1 .

REGIONAL DESIGNATIONS, SINGLE VARIABLES, AND CONSTANTS MAY BE MIXED; BUT M+2N MUST BE LESS THAN 25, WHERE M= THE NUMBER OF SINGLE VARIABLES AND N= THE NUMBER OF REGIONS. EACH REGIONAL DESIGNATION COUNTS AS TWO SINGLE VARIABLES; HENCE, NO MORE THAN 12 REGIONS MAY BE SPECIFIED IN ONE OUTPUT STATEMENT.

EXAMPLES:

```
TX1 ...X20
TD5 ...D7
TC1...CJ1
TV(I1,1) ...V(I1,K1)
TX1 ...X4 TZ0
```

B) INCORRECT

```
TX3 ...X1
TC1...D10
```

9. CONDITIONAL STATEMENTS

EXAMPLES:

```
N:  GO TO 7 IF Y1 = 0.
N:  HALT IF I2 > I1
N:  Y15 ← 4.  IF I1 < 6
```

DESCRIPTION:

ANY ACTION MAY BE MADE CONDITIONAL UPON A GIVEN RELATION. INDEED, RELATION STATEMENTS MAY BE MADE CONDITIONAL, WITH THE ORDER OF DETERMINATION BEING FROM RIGHT TO LEFT.

FOR THE STATEMENT
'(STATEMENT) IF A R B'
A AND B MAY BE ANY EXPRESSIONS, AND R IS A RELATION WRITTEN AS ONE OF THE FOLLOWING CARD CODES:

RELATION	CARD CODE
=	=
≠	≠
<	<
≤	≤
>	>
≥	≥

IF THE RELATION HOLDS, THEN THE NORMAL SEQUENCE OF COMPUTATION IS ALTERED; AND, IF THE RELATION IS NOT SATISFIED THEN COMPUTATION PROCEEDS TO THE NEXT STATEMENT IN THE ORDER IN WHICH THEY WERE WRITTEN.

IN A STATEMENT SUCH AS

```
N:  I1 ← 3  IF C2 < 0  IF J3 = 5
```

THE PROGRAM WILL PERFORM THE SUBSTITUTION ONLY IF BOTH OF THE CONDITIONS ARE MET; OTHERWISE CONTROL WILL PASS TO THE NEXT STATEMENT.

SYNTAX:

```
<CONDITIONAL STATEMENT> ::= <SIMPLE STATEMENT> IF <RELATION>
<SIMPLE STATEMENT> ::= <BASIC STATEMENT>|<COMPOUND STATEMENT>|
    <CONDITIONAL STATEMENT>
<RELATION> ::= <EXPRESSION> <RELATION OPERATOR> <EXPRESSION>
<RELATION OPERATOR> ::= > | < | ≮ | ≯ | ≠ | =
```

MORE EXAMPLES:

A) CORRECT

```
N:  READ IF K1 ≮ 3
    GO TO 7 IF K1 < 3  IF K2 < 4  IF K3 = K4
    TZ1  IF  (I1+Y1)*Z13 ≯ X1↑Y2
N:  GO TO 7 IF I1 = 5 ; J5↑25 IF X1 ≠ 3.5
```

B) INCORRECT

```
N:  IF Y2 = Y1  GO TO 2
```

(THE TRANSFER 'GO TO 2' WILL BE EXECUTED FIRST AND THE CONDITIONAL STATEMENT WILL NEVER BE EXECUTED.)

10. HALT STATEMENT

EXAMPLES:

HALT

DESCRIPTION:

THE EXECUTION OF A 'HALT' STATEMENT WILL CAUSE CONTROL TO PASS OUT OF THE COMPILED PROGRAM TO THE SUPERVISORY MONITOR PROGRAM OR TO THE MACHINE OPERATOR. SUCH A STATEMENT THUS TERMINATES THE RUNNING PROGRAM.

A PROGRAM MUST HAVE AT LEAST ONE AND IT MAY HAVE ANY NUMBER OF 'HALT' STATEMENTS. HOWEVER, ONE AND ONLY ONE MUST BE EXECUTED TO TERMINATE PROGRAM OPERATION. (ABSENCE OF A READ GROUP OR SOME OTHER ERROR CONDITION MUST NOT BE USED TO TERMINATE OPERATION.)

SYNTAX:

<HALT STATEMENT> ::= HALT

11. SYSTEM STATEMENTS

THE STATEMENTS DESCRIBED IN THIS SECTION ARE NOT PART OF THE COMPUTATION BUT MERELY PROVIDE INFORMATION TO THE TRANSLATOR.

THESE STATEMENTS MUST BE THE FIRST STATEMENTS OF ANY PROGRAM.

11.1 STATEMENTS

A) STATEMENT DICTIONARY

EVERY STATEMENT WITH A NON-ZERO NUMBER CREATES AN ENTRY IN THE 'STATEMENT DICTIONARY'. THE STATEMENT

'N IS HIGHEST STATEMENT NUMBER'

WILL CAUSE THE TRANSLATOR TO SET UP A DICTIONARY OF N ENTRIES CORRESPONDING TO THE STATEMENT NUMBERS 1, 2,..., N. HERE N MUST BE AN INTEGER $\prec$ THE LARGEST STATEMENT NUMBER APPEARING IN THE PROGRAM; HOWEVER, EVERY NUMBER $\succ$ N NEED NOT ACTUALLY BE USED.

SYNTAX:

<STATEMENT-NUMBER STATEMENT> ::= <INTEGER> IS HIGHEST STATEMENT NUMBER <CS>

B) ABSOLUTE CONSTANTS

NORMALLY, STORAGE SPACE IS PROVIDED FOR 100 ABSOLUTE CONSTANTS (ABCONS). IF THE PROGRAMMER KNOWS THAT MORE OR LESS THAN 100 ABCONS WILL BE GENERATED, THE STATEMENT

'K ABCONS'

WILL SIGNAL THE TRANSLATOR TO SAVE STORAGE LOCATIONS FOR PRECISELY K ABCONS.

HERE K MUST BE AN INTEGER $\prec$ THE ACTUAL NUMBER OF ABCONS GENERATED.

EACH DISTINCT CONSTANT WRITTEN IN A STATEMENT GENERATES AN ABCON. (HOWEVER, 4 AND -4 GENERATE THE SAME CONSTANT.) EACH OUTPUT STATEMENT GENERATES (M+2N+1) ABCONS, WHERE M IS THE NUMBER OF SINGLE VARIABLES AND N IS THE NUMBER OF REGIONS. FURTHER, EACH USE OF THE 'L' NOTATION GENERATES AN ABCON. OTHERS ARE GENERATED WHEN MATRIX NOTATION IS USED.

THE STATEMENT 'K ABCONS', NEED NOT APPEAR IN THE PROGRAM.

SYNTAX:

<ABCON STATEMENT> ::= <INTEGER> ABCONS <CS>

C) PRINT OFF

THE OUTPUT OF A PROGRAM WILL ALWAYS BE PRINTED, AND THE PROGRAM ITSELF WILL ALSO BE LISTED AS IT IS COMPILED UNLESS THE PROGRAMMER HAS INCLUDED A 'PRINT OFF' STATEMENT.

SYNTAX:
<PRINT-OFF STATEMENT> ::= PRINT OFF <CS>

11.2. VARIABLE STORAGE

EXAMPLES:
1) DIMENSION J(2) X(15) D(200)
2) DIMENSION X(100,K6, J2) J(40, 5, 15) Z(50)

DESCRIPTION:
IMMEDIATELY FOLLOWING THE STATEMENTS IN 11.1 MUST BE THE STATEMENT
'DIMENSION $V_1(N_1, R_1, B_1) \ldots V_K(N_K, R_K, B_K)$'
(THE COMMAS MUST BE WRITTEN BUT NOT THE DOTS.) EACH V_ℓ IS ONE OF THE VARIABLE LETTERS (C, D, G, H, I, J, K, X, Y, AND Z).
EACH VARIABLE LETTER USED IN THE PROGRAM MUST APPEAR IN THE DIMENSION STATEMENT. HOWEVER, NOTE THAT THE LETTER 'B' IS NOT PERMITTED. FOR EACH ℓ, N_ℓ MUST BE $\geq$ THE LARGEST SUBSCRIPT EVER USED IN THE PROGRAM FOR V_ℓ. A V_ℓ-REGION WILL BE RESERVED FOR EACH VARIABLE CLASS APPEARING IN THIS STATEMENT.
IF PART OF THE V_ℓ-REGION IS TO BE USED AS A MATRIX, THEN R_ℓ IS THE NUMBER OF COLUMNS OF THE MATRIX AND B_ℓ IS THE SUBSCRIPT OF THE BASE ELEMENT. THE R_ℓ AND B_ℓ MAY BE EITHER INTEGERS OR INTEGER VARIABLES WITH CONSTANT SUBSCRIPTS. (E.G., K1 BUT NOT KJ1).
FOR EACH OF THE V_ℓ NOT USED AS A MATRIX THE NUMBERS R_ℓ AND B_ℓ SHOULD BE OMITTED. I.E., '$V_\ell(N_\ell)$' IS ALL THAT IS WRITTEN.

THE NUMBER N_ℓ MUST BE A POSITIVE INTEGER <16,384. (IN A LONG PROGRAM, N_ℓ SHOULD NOT BE MUCH LARGER THAN THE ACTUAL NUMBER OF V_ℓ VARIABLES USED, SINCE THERE MAY NOT BE ENOUGH UNRESERVED SPACE LEFT FOR THE PROGRAM ITSELF.)

SYNTAX:
<DIMENSION STATEMENT> ::= DIMENSION <DIMENSION ELEMENT> |
<DIMENSION STATEMENT> <DIMENSION ELEMENT>
<DIMENSION ELEMENT> ::= <NAME TYPE>(<INTEGER>) |
<NAME TYPE>(<INTEGER>,<B OR C>,<B OR C>)
<B OR C> ::= <INTEGER> | <INTEGER NAME TYPE> <INTEGER>

MORE EXAMPLES:
A) CORRECT
DIMENSION Z(50) J(40,5,15) X(100,J6,J2)
(THIS SPECIFIES THAT THE MAXIMUM Z SUBSCRIPT WHICH WILL OCCUR IN THE COMPUTATION IS => 50, AND Z IS NOT USED AS A MATRIX DESIGNATOR. THE MAXIMUM J SUBSCRIPT IS => 140, AND THE MAXIMUM X SUBSCRIPT IS => 100.

THERE ARE NO OTHER VARIABLES USED IN THE PROGRAM WITH THE POSSIBLE EXCEPTION OF 'B'. THE J MATRIX HAS 5 COLUMNS AND J(1,1) = J15. THE X MATRIX HAS J6 COLUMNS AND X(1,1) = XJ2.

NOTE: IN THIS CASE THE VALUES OF J6 (NUMBER OF COLUMNS) AND J2 (BASE SUBSCRIPT) MUST BE SET BY THE PROGRAM. (THEY MUST EITHER BE COMPUTED IN A SUBSTITUTION STATEMENT OR BE READ AS INPUT DATA. THEY MAY ALSO BE CHANGED DURING THE PROGRAM, SO THE PROGRAMMER CAN CHANGE BOTH THE LOCATION AND SIZE OF THE MATRIX DURING EXECUTION.)

B) INCORRECT

```
DIMENSION Z(J1) J(K1+2,5,15)
```

12. M-STATEMENTS

EXAMPLES:

```
STUDENT PROGRAM
I HOPE THIS WORKS      M
K5 = 7   M
```

DESCRIPTION:

A STATEMENT WHOSE RIGHTMOST CHARACTER IS 'M' WILL BE PRINTED BUT WILL BE OTHERWISE IGNORED. ALTHOUGH A COMMENT STATEMENT MAY BE LABELLED AND HENCE SERVE AS A TRANSFER POINT, THESE STATEMENTS MAY NOT BE SO TREATED, SINCE EVEN THE LABEL IS IGNORED.

M-STATEMENTS MAY OCCUR ANYWHERE IN THE PROGRAM.

SYNTAX:

<M-STATEMENT> ::= M | <BASIC CHARACTER> <M-STATEMENT>

13. END STATEMENT

EXAMPLES:

```
PROGRAM END
```

DESCRIPTION:

THE LAST STATEMENT OF EACH PROGRAM MUST BE AN END STATEMENT.

EVERY PROGRAM MUST CONTAIN ONE AND ONLY ONE END STATEMENT.

SYNTAX:

<END STATEMENT> ::= PROGRAM END <CS>

HOW TO USE THE SYSTEM

1. PREPARATION OF STATEMENT CARDS

PROGRAM STATEMENT CARDS FOR THE G-20 MUST BE PUNCHED IN THE FOLLOWING FORMAT:

STATEMENT NUMBER	- COLUMNS 9-11
STATEMENT	- COLUMNS 15-67
IDENTIFICATION (OR BLANK)	- COLUMNS 69-80

ALL OTHER CARD COLUMNS MUST BE BLANK.

EACH STATEMENT MAY CONTAIN UP TO 159 CHARACTERS ON AS MANY CARDS AS DESIRED. IN COUNTING CHARACTERS, 20-GATE WILL IGNORE ALL BLANK COLUMNS (EXCEPT IN ALPHABETIC CONSTANTS).

IF A STATEMENT CAN NOT FIT ENTIRELY ON ONE CARD, IT MAY BE CONTINUED ON SUCCEEDING CARDS OF THE SAME FORMAT. 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 RIGHTMOST CHARACTER ON THAT CARD. 'N' DOES NOT HAVE TO BE PUNCHED IN ANY PARTICULAR COLUMN.

THE LAST CARD OF EACH STATEMENT CARRIES NO SPECIAL DESIGNATION.

STATEMENT NUMBERS MUST BE PUNCHED AS THREE-DIGIT INTEGERS; HOWEVER, LEADING ZEROS MAY BE OMITTED. THE STATEMENT NUMBER NEED ONLY BE PUNCHED ON THE FIRST CARD OF A STATEMENT.

EXCEPT WHEN APPEARING IN AN ALPHABETIC CONSTANT, BLANKS ARE ALWAYS IGNORED IN STATEMENTS. THEREFORE, THEY MAY BE USED AS DESIRED FOR EASIER READING OF PUNCHED CARDS.

THE LAST STATEMENT CARD MUST CARRY THE STATEMENT,

'PROGRAM END'

IF COMPILATION OF A 20-GATE PROGRAM IS SUCCESSFULLY COMPLETED, EXECUTION OF THE COMPILED PROGRAM WILL BE BEGUN, STARTING WITH THE STATEMENT WHOSE NUMBER IS 1.

MANY SPECIAL CHARACTERS APPEAR IN THIS MANUAL WHICH ARE NOT AVAILABLE ON THE KEYPUNCHES. THESE CHARACTERS CAN BE PUNCHED (NOT TOO INCONVENIENTLY) BY USING THE MULTIPLE-PUNCH FACILITY OF THE KEYPUNCH. IT IS EXPECTED THAT ARRANGEMENTS WILL SOON BE MADE TO PUNCH THESE CHARACTERS DIRECTLY. UNTIL THIS IS DONE, HOWEVER, USERS MAY USE CERTAIN ALPHABETIC CHARACTERS (AS IN 650-GATE) INSTEAD OF THE SPECIAL CHARACTERS. THIS WILL BE ONLY A TEMPORARY SITUATION, BECAUSE AS SOON AS THERE IS A MEANS OF PUNCHING THE SPECIAL CHARACTERS DIRECTLY, THE ALPHABETIC CHARACTERS WILL NO LONGER BE INTERCHANGEABLE. THIS WILL MAKE THE LANGUAGE MORE FLEXIBLE,

HOWEVER, SINCE THESE LETTERS WILL BE USED TO PROVIDE MORE VARIABLE CLASSES. THE SPECIAL CHARACTERS AND THEIR CORRESPONDING ALPHABETIC EQUIVALENTS ARE LISTED BELOW.

₁₀	E	POWER OF TEN IN ABCONS AND DATA CARDS
;	M	COMPOUND STATEMENT SEPARATOR
↑	P	EXPONENTIATION OPERATOR
<	Q	LESS THAN
≤	R	LESS THAN OR EQUAL TO
≠	S	NOT EQUAL TO
=	U	EQUAL TO, AS IN A RELATION (BUT SEE BELOW)
>	V	GREATER THAN
≥	W	GREATER THAN OR EQUAL TO
←	=	SUBSTITUTION STATEMENT

THE '=' SIGN IS AN UNUSUAL CASE. CURRENTLY ALL OCCURRENCES OF '=' WILL BE INTERPRETED AS MEANING SUBSTITUTION, AND RELATIONAL EQUALITY WILL HAVE TO BE PUNCHED 'U'. FOR ALL OTHER CASES EITHER OPTION LISTED ABOVE MAY BE USED. THE SPECIAL CHARACTERS ARE RECOMMENDED, SINCE PROGRAMS SO PUNCHED WILL CONTINUE TO WORK (EXCEPT, POSSIBLY, FOR =). PROGRAMS PUNCHED USING THE ALPHABETIC REPRESENTATION WILL PROBABLY HAVE TO BE REPUNCHED EVENTUALLY -- PROBABLY BY SEPTEMBER.

G-20 ALPHABET - 63 CHARACTER

SYMBOL	INTERNAL	CARD CODE	SYMBOL	INTERNAL	CARD CODE
SPACE	00	NO PUNCH	0	40	0
A	01	12 1	1	41	1
B	02	12 2	2	42	2
C	03	12 3	3	43	3
D	04	12 4	4	44	4
E	05	12 5	5	45	5
F	06	12 6	6	46	6
G	07	12 7	7	47	7
H	10	12 8	8	50	8
I	11	12 9	9	51	9
J	12	11 1	≡	52	0 7 8 *
K	13	11 2	.	53	12 3 8
L	14	11 3	+	54	12
M	15	11 4	-	55	11
N	16	11 5	*	56	11 4 8
O	17	11 6	/	57	0 1
P	20	11 7	=	60	3 8
Q	21	11 8	∨	61	12 7 8 *
R	22	11 9	≠	62	11 2 8 *
S	23	0 2	>	63	12 6 8 *
T	24	0 3	<	64	11 5 8 *
U	25	0 4	$	65	11 3 8
V	26	0 5	≥	66	11 6 8 *
W	27	0 6	,	67	4 8 **
X	30	0 7	(	70	0 4 8
Y	31	0 8	)	71	0 5 8 *
Z	32	0 9	[	72	0 6 8 *
|	33	2 8 *	]	73	12 4 8
←	34	6 8 *	↓	74	7 8 *
→	35	11 7 8 *	↑	75	12 2 8 *
¬	36	12 5 8 *	≤	76	0 2 8 *
'	37	0 3 8	≠	77	5 8 *

THE INTERNAL REPRESENTATIONS ABOVE ARE OCTAL INTEGERS.

* MUST BE PUNCHED BY USING MULTIPLE-PUNCH BUTTON.
** PUNCHED BY ' ON NEW KEY-PUNCHES OR EXTRA MINUS SIGN ON OLD KEY-PUNCHES.

2. INPUT DATA CARDS

A SET OF DATA CARDS, READ BY THE TRANSLATED PROGRAM UPON THE EXECUTION OF A SINGLE 'READ' STATEMENT, IS CALLED A 'READ GROUP'.

THE PROGRAM WILL SEQUENTIALLY READ THE INPUT CARDS IN SUCH A READ GROUP, EACH CARD OF WHICH HAS THE INFORMATION CHARACTERS IN COLUMNS 1-72; AND WILL STORE THE INFORMATION INDICATED ON EACH CARD UNTIL THE END-OF-GROUP CHARACTER, '@', IS ENCOUNTERED. THIS CHARACTER MAY APPEAR IN ANY OF COLUMNS 1-73. AFTER THE CONTENTS OF THE CARD CONTAINING THIS CHARACTER ARE STORED, THE PROGRAM CONTINUES ON TO THE NEXT STATEMENT IN PROGRAM SEQUENCE.

THE FIRST APPEARANCE OF THE CHARACTER '/', ANYWHERE IN COLUMNS 1-72, MAY BE USED TO TERMINATE READ INFORMATION ON ANY CARD BUT THE LAST CARD IN A READ GROUP. ALL CHARACTERS FROM THAT COLUMN ON TO THE RIGHT WILL BE IGNORED AND THE NEXT CARD WILL BE READ.

INPUT DATA CARDS FOR 20-GATE MUST BE PUNCHED IN THE FORMAT DEFINED BY THE FOLLOWING SYNTAX:

<DIGIT> ::= 0|1|2|3|4|5|6|7|8|9
<INTEGER> ::= <DIGIT> | <INTEGER> <DIGIT>
<DECIMAL NUMBER> ::= <INTEGER> | <INTEGER>. | .<INTEGER> | <INTEGER>.<INTEGER>
<EXPONENT PART> ::= @<INTEGER> | @<ADDING OPERATOR> <INTEGER> | <EMPTY>
<NUMBER> ::= <DECIMAL NUMBER> <EXPONENT PART>
<NUMERIC VALUE> ::= <NUMBER>|<ADDING OPERATOR> <NUMBER>
<ALPHABETIC CONSTANT> ::= <ANY BASIC CHARACTER OTHER THAN $> | <ALPHABETIC CONSTANT> <ALPHABETIC CONSTANT>
<VALUE> ::= <NUMERIC VALUE>|$<ALPHABETIC CONSTANT>$
<NAME LETTER> ::= B|C|D|G|H|I|J|K|X|Y|Z
<NAME> ::= <NAME LETTER> <INTEGER>
<FIN> ::= <CS>|/
<INPUT> ::= <NAME>=<VALUE>|<NAME> <ADDING OPERATOR> <NUMBER>|<NAME>$<ALPHABETIC CONSTANT>$|<INPUT>,<VALUE>
<INPUT CARD> ::= <INPUT> <FIN>
<FILE> ::= <INPUT CARD>|<INPUT CARD> <FILE>
<READ GROUP> ::= <INPUT>@|<FILE> <READ GROUP>

EACH 'READ' CALL IN 20-GATE WILL READ INTO MEMORY ONE READ GROUP.

THE INPUT DATA CARDS FOR A PROGRAM ARE PLACED IMMEDIATELY AFTER ITS STATEMENT CARDS. THUS THE FIRST CARD OF THE FIRST READ GROUP TO BE READ BY A PROGRAM MUST BE THE CARD FOLLOWING THE PROGRAM'S END STATEMENT.

THE CHARACTER ',' (NOTE <INPUT>) IS USED TO REPRESENT THE LAST VARIABLE ENCOUNTERED WITH THE SUBSCRIPT INCREASED BY 1.
THUS, '$C1=10.,-3.' STORES +10 IN C1 AND -3 IN C2.

AN ALPHABETIC CONSTANT USED IN A READ GROUP MAY HAVE MORE THAN FOUR CHARACTERS. IF SUCH IS THE CASE, THE CONSTANT WILL BE STORED IN SUCCESSIVE FLOATING POINT LOCATIONS, FOUR CHARACTERS PER WORD (COUNTING FROM LEFT TO RIGHT).
IF THE NUMBER OF CHARACTERS IS NOT A MULTIPLE OF FOUR, THE LAST VARIABLE STORED WILL HAVE ITS CHARACTERS RIGHT-JUSTIFIED.

THE LETTER 'E' MAY BE USED INSTEAD OF '$_{10}$' IF DESIRED. THE '$_{10}$' IS PREFERRED, HOWEVER.

SPACES MAY BE USED FREELY; THEY WILL BE IGNORED (EXCEPT, OF COURSE, IN ALPHABETIC INFORMATION).

EXAMPLES:

THE READ GROUP

```
C1=123., 78.395 E 5,-.005 /  READ ALPHA, BETA, GAMMA
,1.3E-8   I79-39J1+4./       READ  ZETA,  I,  J
G491-123456.7890123b+38,+75    *  READ G  AND G1
```

WILL STORE THE FOLLOWING VALUES INTO THE GIVEN LOCATIONS:

C1	+123
C2	+78395b2
C3	-5b-3
C4	+13b-9
I79	-39
J1	+4
G491	-1234567890123*b31
G492	+75

THE READ GROUP

```
X1 $THIS IS RUN 5 OF PROBLEM 123.    $,
$DATA SET 39 --$     X20 = $JULY 28,1961$ *
```

WILL STORE THE FOLLOWING VALUES

X1	$THIS$
X2	$ IS $
X3	$RUN $
X4	$5 OF$
X5	$ PRO$
X6	$BLEM$
X7	$ 123$
X8	$. $
X9	$DATA$
X10	$ SET$
X11	$ 39 $
X12	$ --$
X20	$JULY$
X21	$ 28,$
X22	1961

3. PRINTED OUTPUT

IF THE TWO READ GROUPS USED IN THE EXAMPLES ABOVE (UNDER INPUT DATA CARDS) ARE READ INTO STORAGE, THE OUTPUT STATEMENTS

```
TC1...C4 T179 TJ1 TG491 TG492
AT X1...X24
```

WILL PRODUCE THE FOLLOWING PRINTED OUTPUT:

```
C1     +.1229999999999  10+03    C2     +.7839500000001  10+07
C3     -.4999999999998  10-02    C4     +.1299999999999  10-07
I79    -.3900000000000  10+02    J1     +.4000000000001  10+01
G491   -.1234567890121  10+44    G492   +.7500000000000  10+02
THIS IS RUN 5 OF PROBLEM 123.   DATA SET 39   ---
JULY 28,1961
```

NOTE THAT SOME OF THESE VALUES ARE NOT EXACT. FOR NON-INTEGRAL VALUES, THIS IS A NORMAL RESULT OF THE NECESSARY DECIMAL-BINARY AND BINARY-DECIMAL CONVERSIONS. FOR VALUES WHICH CAN BE EXPRESSED AS INTEGERS, HOWEVER, THE ERRORS ARE SOLELY A RESULT OF THE BINARY-DECIMAL OUTPUT CONVERSION. THUS, FOR THE ABOVE CASE, J1 IS EXACTLY 4 AND C1 IS EXACTLY 123 IN STORAGE.

APPENDIX A - ERROR CODES

MANY ERRORS IN THE USE OF THE LANGUAGE ARE DETECTED BY THE TRANSLATOR. EACH ERROR TYPE IS GIVEN BY A NUMBER, AS LISTED BELOW, AND THE NUMBER IS PRINTED ON THE OUTPUT LISTING WITH AN APPROPRIATE MESSAGE. IF AN ERROR IS DETECTED IN A SYSTEM STATEMENT, TRANSLATION COMES TO AN IMMEDIATE HALT. HOWEVER, ANY ERRORS FOUND IN STATEMENTS AFTER THE DIMENSION STATEMENT ARE MERELY LISTED, AND TRANSLATION CONTINUES. NO MORE THAN ONE ERROR WILL BE FOUND IN A STATEMENT. IF ANY ERRORS ARE DETECTED IN THE STATEMENTS OF A PROGRAM, THE RESULTING CODE WILL BE INCOMPLETE AND WILL NOT BE RUN.

IF AT RUN TIME AN ILLEGAL VALUE IS USED AS A SUBROUTINE ARGUMENT, THE PROGRAM WILL HALT AND AN ERROR INDICATION OF THE FORM

RUN ERROR READ OR
RUN ERROR LOG

ETC. WILL BE PRINTED. IF AN ERROR OCCURS DURING EXPONENTIATION

RUN ERROR X↑A

WILL BE PRINTED.

TRANSLATION ERROR CODES, 20-GATE

2. A VARIABLE LETTER HAS BEEN USED THAT DID NOT APPEAR IN THE DIMENSION STATEMENT, OR A NUMERIC SUBSCRIPT HAS BEEN USED WHICH IS LARGER THAN THE NUMBER OF VARIABLES RESERVED IN THE SYSTEM STATEMENTS. OR STMT. NO'S.

3. THE ACTUAL (OR IMPLIED) PARENTHESIS NESTING HAS EXCEEDED 9.

4. THE SUBROUTINE ENTRY TABLE CAPACITY OF 40 HAS BEEN EXCEEDED.

5. THE ASSEMBLED PROGRAM HAS EXCEEDED THE AVAILABLE STORAGE.

7. A STATEMENT HAS A MISSING OR AN EXTRA '$' SIGN.

9. A STATEMENT CONTAINS MORE THAN 139 CHARACTERS.

10. A STATEMENT NUMBER IS LARGER THAN THE ONE IN THE SYSTEM STATEMENT.

11. THE SAME STATEMENT NUMBER HAS BEEN USED TWICE.

12. TOO MANY THREE-ADDRESS CODES HAVE BEEN GENERATED FROM A STATEMENT.

13. THE STATEMENT CONTAINS MORE LEFT PARENTHESES THAN RIGHT.

14. MORE THAN 100 ABCONS HAVE BEEN GENERATED, OR MORE THAN SPECIFIED IN AN 'ABCON' SYSTEM STATEMENT.

25. THE STATEMENT IS IMPROPERLY FORMED; MISSING OPERATION, PARENTHESIS, ETC.

27. THE STATEMENT HAS AN IMPROPER COMBINATION OF OPERATION SYMBOLS, E.G., */.

29. THE STATEMENT HAS AN IMPROPER MATRIX SUBSCRIPT, TYPE VARIABLE, OR USE OF 'L'.

30. AN ITERATION STATEMENT CONTAINS LESS THAN FIVE COMMAS.

31. THE ITERATION NESTING DEPTH HAS EXCEEDED EIGHT, OR THE NESTING IS IMPROPER.

32. MORE THAN 24 ARGUMENTS HAVE BEEN USED IN A TYPE STATEMENT OR MORE THAN 25 ARGUMENTS HAVE BEEN USED IN A SUBROUTINE CALL.

33. THE STATEMENT HAS MORE RIGHT PARENTHESES THAN LEFT.

34. THE 'PROGRAM END' STATEMENT HAS OCCURRED BEFORE ALL ITERATION LOOPS HAVE BEEN TERMINATED.

~~35. THE PROGRAM DOES NOT CONTAIN A 'HALT' STATEMENT.~~

38. A CHARACTER HAS BEEN USED WHICH IS ILLEGAL IN THE GATE LANGUAGE.

39. A CHARACTER LEGAL TO GATE HAS BEEN USED IN AN IMPROPER PLACE IN A STATEMENT.

SYSTEM STATEMENT ERROR CODES

52. THE LAST CHARACTER OF A SYSTEM STATEMENT IS NOT 'C' OR 'M' OR 'F', AND THE FIRST CHARACTER IS NOT AN INTEGER OR 'D' OR 'P'.

53. THE FIRST CHARACTER OF A SYSTEM STATEMENT IS AN INTEGER WHICH IS NOT FOLLOWED BY 'A' OR 'I'.

54. NO PROGRAM END CARD

56. THE 'DIMENSION' STATEMENT IS INCORRECTLY FORMED.

57. A SUBSCRIPTED SUBSCRIPT HAS BEEN USED IN A 'DIMENSION' STATEMENT (AS C(100,I,J1,1)).

58. AN INDEX USED IN 'DIMENSION' STATEMENT I, J, AND K VARIABLES IS TOO LARGE.
(E.G., DIMENSION C(100,15,7) I(3))

59. TOO MUCH SPACE HAS BEEN RESERVED BY SYSTEM STATEMENTS.

MACHINE ERRORS

60. COMPUTER FAILURE.

61. CARD READER FAILURE.

62. PRINTER FAILURE.

63. MAGNETIC TAPE FAILURE.

RELOCATION ERRORS

81. A SUBROUTINE HAS BEEN CALLED WHICH IS NOT IN THE LIBRARY.

82. THERE IS NO LONGER ENOUGH SPACE FOR THE REQUIRED SUBROUTINES.

84. INDIRECT REFERENCES HAVE CAUSED MORE THAN 40 SUBROUTINES TO BE CALLED.

APPENDIX B

SUBROUTINES CURRENTLY AVAILABLE TO THE PROGRAMMER ARE THE FOLLOWING:

READ	
SIN	(SINE OF AN ANGLE IN RADIANS)
COS	(COSINE OF AN ANGLE IN RADIANS)
LOG	(LOGARITHM TO THE BASE E)
EXP	(EXPONENTIAL - BASE E)
SQRT	(SQUARE ROOT)
ARTN	(ARCTANGENT - RESULT IN RADIANS)

OTHERS WILL BE AVAILABLE SOON.

APPENDIX C - DIFFERENCES BETWEEN 20-GATE AND 650-GATE

1. THE CARD FORMAT FOR STATEMENT CARDS HAS BEEN CHANGED, ALTHOUGH CARDS PREPARED FOR THE 650 MAY BE USED IF THEY ARE PUNCHED WITH SOME INFORMATION IN COLUMNS 1 TO 4. SEE PART III, SECTION 1. AFTER SEPTEMBER 1961 IT IS UNLIKELY THAT THE OLD FORMAT WILL CONTINUE TO BE PERMITTED.

2. MANY CHARACTERS HAVE BEEN CHANGED. FOR EXAMPLE, '↑' IS TO BE USED INSTEAD OF 'P'. ALL OLD CHARACTERS ARE STILL ACCEPTABLE, HOWEVER, ALTHOUGH BY SEPTEMBER IT IS LIKELY THAT THEY WILL NOT BE. SEE PART III, SECTION 1 FOR FURTHER DETAILS.

3. THE FORMAT FOR DATA CARDS IS CHANGED COMPLETELY. SEE PART III, SECTION 2, FOR DISCUSSION.

4. THE SYSTEM STATEMENT 'K USED IN SUBROUTINES' IS NO LONGER REQUIRED. INDEED, IT IS NO LONGER LEGAL.

5. THE TRANSLATION OF ITERATION STATEMENTS HAS BEEN CHANGED SO THAT THE BODY WILL BE EXECUTED ZERO TIMES IF THE TEST FAILS WITH THE INITIAL VALUE OF THE CONTROLLED VARIABLE. ON THE 650, THE BODY WAS ALWAYS EXECUTED AT LEAST ONCE.

6. THE USE OF ALPHABETIC CONSTANTS HAS CHANGED. ONLY FOUR CHARACTERS MAY OCCUR IN A CONSTANT. (IF MORE THAN FOUR ARE USED, THE RIGHTMOST FOUR WILL BE USED.) FURTHER, ALPHABETIC INFORMATION MUST BE STORED IN FLOATING POINT VARIABLES, AS INTEGER VARIABLES WILL NOT CONTAIN ENOUGH INFORMATION.

7. STATEMENT NUMBERS NEED NOT APPEAR ON TYPE STATEMENTS. THERE IS NOTHING ON THE G-20 COMPARABLE TO THE STORAGE ENTRY SWITCHES ON THE 650, AND ALL OUTPUT STATEMENTS ARE ALWAYS EXECUTED, REGARDLESS OF THEIR STATEMENT NUMBER.

8. IT IS AN ERROR, DETECTED BY THE TRANSLATOR, IF THE SAME NUMBER IS USED MORE THAN ONCE AS A STATEMENT LABEL. ON THE 650, THIS CONDITION WAS NOT DETECTED.

9. CALLS FOR OUTPUT WILL ALWAYS PRODUCE PRINTED OUTPUT ON THE PRINTER. THERE IS, AT PRESENT, NO PROVISION FOR PUNCHING CARDS. SINCE WE HAVE NO WAY OF PRINTING CARDS ONCE THEY ARE PUNCHED, THIS IS NO HARDSHIP.

10. THE LIMITS ON THE POSSIBLE VALUES OF VARIABLES ARE CHANGED. FOR FLOATING POINT VARIABLES GREATER TOLERANCE IS PERMITTED, BUT INTEGER VARIABLES ARE LIMITED TO BEING LESS THAN 2↑21 (2,097,152).

APPENDIX D - SYNTAX

```
<PROGRAM> ::= <NAME HEAD> <SYSTEM STATEMENTS>
    <DIMENSION STATEMENT> <CS> <PROGRAM BODY> <END STATEMENT>
<NAME HEAD> ::= <M-STATEMENT SPECIFYING PROGRAMMER'S NAME> <CS>
<SYSTEM STATEMENTS> ::= <PRINT-OFF STATEMENT>ABCON STATEMENT>|
    <STATEMENT NUMBER STATEMENT>|<M-STATEMENT>|
    <COMMENT STATEMENT>|<SYSTEM STATEMENTS> <SYSTEM STATEMENTS>
<PRINT-OFF STATEMENT> ::= PRINT OFF <CS>
<ABCON STATEMENT> ::= <INTEGER> ABCONS <CS>
<STATEMENT-NUMBER STATEMENT> ::= <INTEGER> IS HIGHEST STATEMENT
    NUMBER <CS>
<DIMENSION STATEMENT> ::= DIMENSION <DIMENSION ELEMENT> |
    <DIMENSION STATEMENT> <DIMENSION ELEMENT>
<DIMENSION ELEMENT> ::= <NAME TYPE>(<INTEGER>) |
    <NAME TYPE>(<INTEGER>,<B OR C>,<B OR C>)
<B OR C> ::= <INTEGER> | <INTEGER NAME TYPE> <INTEGER>
<PROGRAM BODY> ::= <STATEMENT> | <STATEMENT> <PROGRAM BODY>
<END STATEMENT> ::= PROGRAM END <CS>
<BASIC STATEMENT> ::= <GO TO STATEMENT>|<HALT STATEMENT>|
    <COMMENT STATEMENT>|<CONTINUABLE STATEMENT>
<CONTINUABLE STATEMENT> ::= <SUBSTITUTION STATEMENT> |
    <SUBROUTINE STATEMENT> | <INPUT-OUTPUT STATEMENT>
<COMPOUND STATEMENT> ::= <SIMPLE STATEMENT> |
    <COMPOUND STATEMENT> ; <CONTINUABLE STATEMENT>
<CONDITIONAL STATEMENT> ::= <SIMPLE STATEMENT> IF <RELATION>
<SIMPLE STATEMENT> ::= <BASIC STATEMENT>|<COMPOUND STATEMENT>|
    <CONDITIONAL STATEMENT>
<STATEMENT> ::= <LABEL> <SIMPLE STATEMENT> <CS> |
    <ITERATION> <CS> | <M-STATEMENT> <CS>
<LABEL> ::= <UNSIGNED INTEGER> : | <EMPTY>
<ITERATION> ::= <LABEL> ^1<INTEGER>,<ITERATION PART>
    <STATEMENT SEQUENCE> ^1: <SIMPLE STATEMENT> |
    ^2 <INTEGER>: ^2 , <ITERATION PART>
<ITERATION PART> ::= <VARIABLE>,<EXPRESSION>,<EXPRESSION>,
    <EXPRESSION>,
<STATEMENT SEQUENCE> ::= <CS>|<STATEMENT SEQUENCE> <STATEMENT>
<SUBSTITUTION STATEMENT> ::= <VARIABLE>=<EXPRESSION>
<SUBROUTINE STATEMENT> ::= <FUNCTION DESIGNATOR>
<GO TO STATEMENT> ::= GO TO <EXPRESSION>
<HALT STATEMENT> ::= HALT
<COMMENT STATEMENT> ::= C|<BASIC CHARACTER> <COMMENT STATEMENT>
<M-STATEMENT> ::= M | <BASIC CHARACTER> <M-STATEMENT>
<INTEGER NAME TYPE> ::= I|J|K
<NAME TYPE> ::= C|D|G|N|X|Y|Z|<INTEGER NAME TYPE>
<VARIABLE> ::= <NAME TYPE> <SUBSCRIPT> | B<SIMPLE SUBSCRIPT>
<SIMPLE SUBSCRIPT> ::= <VARIABLE> | <INTEGER> | (<EXPRESSION>)|
    <UNARY OPERATOR> <PRIMARY>
<SUBSCRIPT> ::= <SIMPLE SUBSCRIPT>|(<EXPRESSION>,<EXPRESSION>)
```

<DIGIT> ::= 0|1|2|3|4|5|6|7|8|9
<INTEGER> ::= <DIGIT> | <INTEGER> <DIGIT>
<DECIMAL NUMBER> ::= <INTEGER> | <INTEGER>. | .<INTEGER> |
<INTEGER>.<INTEGER>
<EXPONENT PART> ::= ₁₀<INTEGER> | ₁₀<ADDING OPERATOR> <INTEGER> |
<EMPTY>
<NUMBER> ::= <DECIMAL NUMBER> <EXPONENT PART>
<PRIMARY> ::= <NUMBER> | <VARIABLE> | <FUNCTION DESIGNATOR> |
L <VARIABLE>|L <CONSTANT>|(<SUBSTITUTION STATEMENT>)|
(<EXPRESSION>)|<UNARY OPERATOR> <PRIMARY>
<FACTOR> ::= <PRIMARY> | <FACTOR>↑<PRIMARY>
<TERM> ::= <FACTOR> | <TERM> <MULTIPLYING OPERATOR> <FACTOR>
<EXPRESSION> ::= <TERM> | <EXPRESSION> <ADDING OPERATOR> <TERM>
<UNARY OPERATOR> ::= + | - | A | ↓
<ADDING OPERATOR> ::= + | -
<MULTIPLYING OPERATOR> ::= * | /
<RELATION> ::= <EXPRESSION> <RELATION OPERATOR> <EXPRESSION>
<RELATION OPERATOR> ::= > | < | ≮ | ≯ | ≠ | =
<FUNCTION DESIGNATOR> ::= <FUNCTION NAME>.(<PARAMETER PART>)
<PARAMETER PART> ::= <EXPRESSION>|<PARAMETER PART>,<EXPRESSION>
<FUNCTION NAME> ::= <LETTER>|<DIGIT>|<FUNCTION NAME> <LETTER>|
<FUNCTION NAME> <DIGIT>
<INPUT-OUTPUT STATEMENT> ::= <INPUT STATEMENT>|
<OUTPUT STATEMENT>
<INPUT STATEMENT> ::= READ
<OUTPUT STATEMENT> ::= <ALPHABETIC OUTPUT STATEMENT> |
<NUMERIC OUTPUT STATEMENT>
<ALPHABETIC OUTPUT STATEMENT> ::= A<NUMERIC OUTPUT STATEMENT>
<NUMERIC OUTPUT STATEMENT> ::= <OUTPUT ELEMENT> |
<NUMERIC OUTPUT STATEMENT> <OUTPUT ELEMENT>
<OUTPUT ELEMENT> ::= T<CONSTANT> | T<VARIABLE> |
T<VARIABLE>...<VARIABLE>
<CONSTANT> ::= <NUMBER> | $<ALPHABETIC CONSTANT>$
<ALPHABETIC CONSTANT> ::= <ANY BASIC CHARACTER OTHER THAN $> |
<ALPHABETIC CONSTANT> <ALPHABETIC CONSTANT>
<LETTER> ::= A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|
V|W|X|Y|Z
<BASIC CHARACTER> ::= <LETTER>|<DIGIT>|₀|₁|₂|₃|₄|₅|₆|₇|₈|₉|
/|=|≠|<|$|>|;|(|)|↓|↑|→|^|∨|[|]|:|'
<CS> ::= <CARD SHIFT>

APPENDIX E. DESCRIPTION ON SYNTAX USED IN MANUAL

BACKUS NORMAL FORM

A DESCRIPTIVE MANUAL FOR AN ARTIFICIAL LANGUAGE SUCH AS GATE OR ALGOL MUST ACCOMPLISH TWO OBJECTIVES: (1) IT MUST SPECIFY, GIVEN A STRING OF CHARACTERS, WHETHER OR NOT THAT STRING IS A LEGAL STATEMENT IN THE LANGUAGE; AND (2) IT MUST SPECIFY PRECISELY THE MEANING OF ANY STATEMENT RECOGNIZED AS BEING LEGAL. IN THE PAST, MANUALS HAVE FULFILLED THESE OBJECTIVES BY GIVING DESCRIPTIONS, IN ENGLISH, OF THE LANGUAGE AND ITS MEANING. AS COMPUTER LANGUAGES HAVE BECOME MORE COMPLEX, HOWEVER, SUCH DESCRIPTIONS HAVE BECOME LESS AND LESS ADEQUATE. EVEN WELL WRITTEN DESCRIPTIONS HAVE FREQUENTLY BEEN FOUND, ON CLOSE EXAMINATION, TO LEAVE AMBIGUOUS CERTAIN POINTS ABOUT THE LANGUAGE BEING DEFINED.

THE DEFINITION OF A LANGUAGE MUST FIRST SPECIFY (BY LISTING IT) THE ALPHABET OF THE LANGUAGE. IT MUST THEN SHOW HOW THE LARGER ELEMENTS OF THE LANGUAGE ARE MADE UP FROM THE ALPHABET. IT IS HERE THAT THE DIFFICULTY ARISES, SINCE THE ALPHABET OF THE LANGUAGE BEING DEFINED, CALLED THE 'OBJECT LANGUAGE', OVERLAPS THE ALPHABET OF THE LANGUAGE IN WHICH THE DEFINITION IS WRITTEN, REFERRED TO AS THE 'META-LANGUAGE'. THUS THE READER FREQUENTLY CANNOT TELL WHETHER A GIVEN PIECE OF TEXT IS AN EXAMPLE OF THE OBJECT LANGUAGE OR IS A PART OF THE DESCRIPTIVE META-LANGUAGE.

WITH THESE CONSIDERATIONS (AND OTHERS) IN MIND, J. W. BACKUS [1] HAS DEVISED A TECHNIQUE FOR ACCOMPLISHING OBJECTIVE (1) OF THE FIRST PARAGRAPH. HE HAS DEFINED A LANGUAGE, WHICH HAS COME TO BE CALLED BACKUS NORMAL FORM AND IS HERE ABBREVIATED BNF, TO BE USED AS A META-LANGUAGE FOR THE CLEAR AND UNAMBIGUOUS DEFINITION OF OBJECT LANGUAGES. INDEED, BNF WAS DEVISED SPECIFICALLY FOR THE PURPOSE OF DESCRIBING THE LANGUAGE ALGOL 60, AND THE MOST CONSPICUOUS USE OF BNF IS IN THE ALGOL 60 REPORT [2]. THE FOLLOWING PARAGRAPHS OF THIS PAPER CONTAIN A DESCRIPTION OF BNF WITH SOME EXAMPLES OF ITS USE. SINCE A KNOWLEDGE OF BNF WILL BE ASSUMED IN MANY FUTURE PUBLICATIONS OF THE CARNEGIE TECH COMPUTATION CENTER, THIS PAPER IS RECOMMENDED READING FOR USERS OF THIS FACILITY.

IN BNF THE FOLLOWING FOUR META-LINGUISTIC SYMBOLS ARE INTRODUCED:

< > | ::=

THESE ARE CALLED 'META-LINGUISTIC SYMBOLS' SINCE THEY MUST NOT BE INCLUDED IN THE ALPHABET OF THE OBJECT LANGUAGE. IT IS IN THE USE OF THESE META-LINGUISTIC SYMBOLS THAT BNF IS CAPABLE OF PROVIDING CLEAR DEFINITIONS OF AN OBJECT LANGUAGE.

STRINGS OF CHARACTERS ENCLOSED IN THE BRACKETS < > ARE CALLED META-LINGUISTIC VARIABLES. A META-LINGUISTIC VARIABLE IS THE NAME GIVEN TO A CLASS OF OBJECTS IN THE OBJECT LANGUAGE. THE MARK | MAY BE READ AS 'OR', AND THE MARK ::= (TO BE REGARDED AS A SINGLE SYMBOL) MAY BE READ AS 'IS DEFINED AS'. THE USE OF THESE

SYMBOLS IS BEST ILLUSTRATED BY EXAMPLE.

EXAMPLE 1: CONSIDER:

<DIGIT> ::= 0|1|2|3|4|5|6|7|8|9

THE ABOVE META-LINGUISTIC FORMULA MAY BE READ AS :
'A MEMBER OF THE META-LINGUISTIC CLASS DIGIT IS DEFINED AS 0 OR 1 OR 2 OR 3 OR 4 OR 5 OR 6 OR 7 OR 8 OR 9.'
THUS THE OCCURRENCE OF <DIGIT> IN A META-LINGUISTIC FORMULA STANDS FOR ANY ONE DECIMAL DIGIT.

AN ESPECIALLY USEFUL FEATURE OF BNF - INDEED, ITS REASON FOR EXISTANCE - IS THAT A DEFINITION MADE USING THIS META-LANGUAGE MAY BE RECURSIVE. A RECURSIVE DEFINITION, FOR OUR PURPOSES, MAY BE THOUGHT OF AS ONE IN WHICH THE OBJECT BEING DEFINED IS INCLUDED, EITHER DIRECTLY OR INDIRECTLY, IN ITS DEFINITION. HERE AGAIN, ILLUSTRATIVE EXAMPLES BEST SET FORTH THE POINT.

EXAMPLE 2: AGAIN LET US CONSIDER:

<INTEGER> ::= <DIGIT>|<INTEGER> <DIGIT>

THE META-LINGUISTIC FORMULA IN THIS EXAMPLE MAY BE READ AS:
'A MEMBER OF THE META-LINGUISTIC CLASS INTEGER IS DEFINED AS A MEMBER OF THE META-LINGUISTIC CLASS DIGIT OR A MEMBER OF THE META LINGUISTIC CLASS INTEGER FOLLOWED BY A MEMBER OF THE META-LINGUISTIC CLASS DIGIT.'
MORE BRIEFLY, ONE MIGHT SAY:
'AN INTEGER IS EITHER A DIGIT OR AN INTEGER FOLLOWED BY A DIGIT.'
THE RECURSIVE NATURE OF THE DEFINITION OF INTEGER IS SEEN IN OBSERVING THAT THE META-LINGUISTIC VARIABLE INTEGER OCCURS ON BOTH THE LEFT SIDE AND THE RIGHT SIDE OF THE FORMULA.

EXAMPLE 3: LET US NOW CONSIDER A MORE COMPLEX CASE.

<AB> ::= (|[|<AB>(|<AB> <DIGIT>

THE META-LINGUISTIC FORMULA ABOVE DEFINES THE CLASS AB. THE READER SHOULD SATISFY HIMSELF THAT THE FOLLOWING ARE MEMBERS OF AB

[((((1(37(
(12345(
(((
[86

AND THAT THE FOLLOWING ARE NOT MEMBERS OF AB

213
[(([
[[

EXAMPLE 4: THE FOLLOWING EXAMPLE IS EVEN MORE COMPLEX,

ALTHOUGH NO NEW RECURSIVENESS IS INTRODUCED.

<DIGIT> ::= 0|1|2|3|4|5|6|7|8|9
<INTEGER> ::= <DIGIT>|<INTEGER> <DIGIT>
<DECIMAL NUMBER> ::= <INTEGER>|<INTEGER>.|.<INTEGER>|
<INTEGER>.<INTEGER>
<EXPONENT PART> ::= <EMPTY>|₁₀<INTEGER>|
₁₀<ADDING OPERATOR> <INTEGER>
<NUMBER> ::= <DECIMAL NUMBER> <EXPONENT PART>
<ADDING OPERATOR> ::= +|-
<EMPTY> ::=

THE META-LINGUISTIC CLASS 'EMPTY' REFERS TO THE STRING CONTAINING NO CHARACTERS - THE EMPTY STRING.

THIS LAST EXAMPLE ILLUSTRATES THE APPLICATION OF BNF TO THE DEFINITION OF NUMBER. AN EXAMINATION OF THE EXAMPLE WILL SHOW THAT THE CLASS 'NUMBER' INCLUDES ALL STRINGS OF SYMBOLS USUALLY REGARDED AS BEING UNSIGNED DECIMAL NUMBERS.

FOR USE IN THIS MANUAL, A MODIFICATION OF 'BNF' HAS BEEN MADE FOR THE PURPOSE OF DESCRIBING ITERATION STATEMENTS. THE NOTATION

~1<OBJECT>

IN A META-LINGUISTIC DEFINITION AND THE NOTATION

^1

APPEARING SUBSEQUENTLY REFER TO THE SAME VALUE OF THE META-LINGUISTIC VARIABLE <OBJECT>. THE INTEGER '1' IDENTIFIES ONE OF (POSSIBLY) SEVERAL SUCH CORRESPONDENCES. IN THE DEFINITION OF <ITERATION> THIS ARTIFICE IS USED TO INDICATE THAT THE END OF THE SCOPE OF AN ITERATION STATEMENT OCCURS WHEN THE SAME INTEGER OCCURS AS A STATEMENT LABEL AS WAS USED IN THE ITERATION STATEMENT. THERE IS NO WAY TO DO THIS IN 'BACKUS NORMAL FORM'.

IN SOME OF THE META-LINGUISTIC FORMULAE THE VARIABLE <CS>, STANDING FOR 'CARD SHIFT', AND THE PUNCTUATION ':' ARE USED. THESE ARE ALSO ARTICIFICES. 'BNF' IS CAPABLE OF DESCRIBING ONLY STRINGS OF CHARACTERS. IN GATE, HOWEVER, IT IS NOT ENOUGH TO DESCRIBE STRINGS. THE FACT THAT SOME INFORMATION MUST GO INTO CERTAIN SPECIFIED CARD COLUMNS GIVES THE TRANSLATOR IMPLICIT INFORMATION. THUS IN THE SYNTAX A COLON INDICATES THAT THE OBJECT TO ITS LEFT IS A STATEMENT LABEL AND THAT THE OBJECT TO ITS RIGHT IS A STATEMENT BODY. SIMILARLY, 'CS' INDICATES THE END OF A PHYSICAL STATEMENT, AS INDICATED BY THE END OF A CARD (WITHOUT AN 'N' CONTINUATION SYMBOL).

BIBLIOGRAPHY:

[1] J. W. BACKUS, THE SYNTAX AND SEMANTICS OF THE PROPOSED INTERNATIONAL LANGUAGE OF THE ZURICH ACM-GAMM CONFERENCE. ICIP PARIS, JUNE 1959.

[2] J. W. BACKUS, ET AL, REPORT ON THE ALGORITHMIC LANGUAGE ALGOL 60. COMMUNICATIONS OF THE ASSOCIATION FOR COMPUTING MACHINERY, VOL. 3, NO. 5, MAY 1960. (COPIES OF THIS REPORT ARE AVAILABLE AT THE COMPUTATION CENTER.)

20-GATE Subroutine: RAND Random Number

A call for this subroutine will produce the next of a sequence of (pseudo-) random numbers. The sequence is defined by the recurrence relation

$$x_{n+1} \equiv x_n * 5^9 \pmod{8^7}$$

where each x_i satisfies

$$0 \leq x_i < 8^7$$

and x_0 is any odd integer. The user will get as output random numbers linearly distributed on the range [0, W-1], where W is the second parameter of the subroutine.

A call for the subroutine has the form RAND.(LΔ, W) . Here Δ is to be any variable and is to contain the current x_i. W may be any expression whose value is positive. The routine will replace Δ by a new value, and the value of the routine will be an integer in [0, W-1].

To use this routine the programmer should initialize Δ (or each Δ he is using) to any odd positive integer. Then he need never refer to Δ again except in the calls for RAND.

Example: The following statements will store into K0 to K13 a set of random numbers which are linearly distributed on the ranges X1 to X2. It is assumed that J1 has been set appropriately.

```
3, J2, 0, 1, I3,
3:   KJ2 ← RAND. ( L J1 , X2 - X1 ) - X1
```