BENDIX COMPUTER DIVISION OF THE BENDIX CORPORATION
5630 Arbor Vitae Street, Los Angeles 45, California

TECHNICAL APPLICATIONS MEMORANDUM NO. 72

27 March 1961

TITLE: Programming notes for the ALGO system

PURPOSE: The information contained in this memorandum answers several questions which have been asked and which may arise concerning use of the ALGO system. Other points of interest are included.

EQUIPMENT AFFECTED: G-15D and optionally MTA-2

EFFECTIVE DATE: 27 March 1961

1. (Q) What form is required for declaring processes which use no inputs but use one or more outputs?

   (A) This question can best be answered by use of an example. Consider a procedure called ALPHA which uses no inputs and only one output called BETA. A dummy input must be used in the declaration of ALPHA.

```
15. PROCEdure ALPHA (JUNK = BETA)
```

   In this example, JUNK need not be mentioned again until the process call at which time another dummy must be used.

```
33. ALPHA (GARB = z)
```

   The above example is also true for a FUNCTion.

2. (Q) What use should be made of the NEG operator in the ALGO system?

   (A) Minus signs (-) must not be used immediately following opening parenthesis. The operator NEG must be used instead.

   Example: The following expression is to be computed.

   $x = \frac{y2}{-3z}$ This should be written as

```
17. x = y↑2 / (NEG 3 * z)
```

   and not as

```
17. x = y ↑2 / (- 3*z)
```

3. (Q) What rules govern the use of process calls as part of an algebraic statement?

   (A) To insure reliable operation of the ALGO system, process calls included as part of an algebraic statement should be the first elements on the right of the equal sign. More than one process call should not be included in an algebraic statement. If two process calls were desired in a single algebraic statement, make one of the process calls as a preceding statement and use the output variable in the following algebraic statement.

   Example: Process IOTA and GAMMA are to be used in a statement.

   IOTA and GAMMA have been declared.

```
 5.  IOTA (A, B = C)
 6.  BEGIN
 7.  C = A * KEYBD - B↑2
 8.  RETURN
 9.  END
10.  GAMMA (E, F = G)
11.  BEGIN
12.  G = E/2 + F↑2
13.  RETURN
14.  END
```

The statement below including both procedures must not be written

```
x = IOTA (w, v = u) - M↑2 + GAMMA (P, Q = T)
```

Instead, the call could be

```
25.  GAMMA (P, Q = T)
26.  x = IOTA (w, v = u) - M↑2 + T
```

4. (Q) Could the title of a program be typed out during object program run?

   (A) No. The actual title of a program is not retained following the operation of Package No. 2 of the compiling process due to the complexity of the ALGO system. Therefore the title could not be typed out during the object program run.

5. (Q) What means of detection and correction should be employed concerning scratch tape reading errors in Package No. 2 and Package No. 3?

   (A) a) Paper Tape Scratch Pad

   When Package No. 2 detects an error when reading the output of Package No. 1, it halts and rings a bell. The operator should rewind the scratch tape one block and cycle the compute switch. If the program continually detects an error, the operator should start over with Package No. 1.

   When Package No. 3 detects an error when reading the output of Package No. 2, it types R and halts. Reread the tape as above, or repeat the operation of Package No. 2.

   b) Magnetic tape scratch pad

   If Package No. 2 detects a read error, it halts and rings a bell. Cycle the compute switch and the block will be reread. If the read error continues, start over.

   If Package No. 3 detects a read error, it types R and halts. When this happens, the operator must use the updater to rewrite the Editor output blocks (see updater instructions) and go thru Package No. 2 again.

6. (Q) Is there a method of labeling keyboard inputs to an object program?

   (A) SPACE limitations prohibit the automatic labeling (via typewriter) of keyboard inputs to the object program. However, the programmer can easily write his own labels using ALGO language.

   Example:

```
15.  PRINT (FORMA) = 1 Ⓢ
16.  X = KEYBD Ⓢ
17.  PRINT (FORMA) = 2 Ⓢ
18.  Y = KEYBD Ⓢ
ETC.
```

KEYBD entries must be made in the same order in which they are written.

7. (Q) When typing in an algebraic statement which is too long for one line on the typewriter, can the operator hit the carriage return key or must he return the carriage manually?

   (A) If, during type-in in Package No. 1, the type-in goes beyond one line and the operator returns the carriage and completes his type-in on the second line, the carriage return code generated will be ignored unless it is in the middle of an identifier or operator. The hyphen (-) key cannot be used to separate words at the end of lines.

8. (Q) What rules govern the typing out of a carriage return or tab following the typing of a floating point number during the object program run?

   (A) If a number is too large for the format under which it is to be typed, the number will be typed in floating point. If there is a carriage return in the format which was too small, the type-out will be followed by a carriage return. If not, the floating point type-out will be followed by a tab. If (FL) is specified as the format, the type-out will be followed by a tab.

9. (Q) Could a dictionary check be made at the end of Package No. 1?

   (A) Package No. 1 separates and encodes all declarations, converts constants, forms formats and identifies statement types. It does not investigate identifiers or operands within the statement and therefore cannot make a dictionary check. This check is one of the functions of Package No. 2.

10. (Q) By what means (other then updating) can we change constants in the object program?

   (A) In order to change a constant when the object program is being run, the operator must know the location of the constant. If the constant is one of an array, the operator can determine its location in the following manner. During the second phase of compiling (Package No. 2), the computer typed the entry number of the constant array declaration and also the location assigned to the first constant of the array. All constants in the array follow sequentially.

   If the constant to be changed is not an element of a constant array, i.e. the constant is a numeric value contained in an algebraic statement, its location is determined by use of the following comments:

   a) assignment starts in location 7.
   b) 4-words are assigned for every format.
   c) 1-word is assigned for every constant.
   d) count the number of assignments preceding the constant to be changed.

e) this number plus 7 is the location of the constant to be changed.

Caution: Constants of 0, 1, or 2 and constants modifying subscripts cannot be changed and should not be counted in determining constant locations as they are not assigned locations.

Example: The following program is written.

```
 1.  TITLE  EXAMPLE  Ⓢ
 2.  FORMAT  FORM1 (S3DP2DT), FORM2 (S2DP)  Ⓢ
 3.  CONSTANT  CON1 (3)  Ⓢ
 4.
 4.  1.67  Ⓢ
 5.  55.. 1234  Ⓢ
 6.  42  Ⓢ
 7.  BEGIN  Ⓢ
 8.  X = KEYBD  *1.37  Ⓢ
 9.  Y = KEYBD/  .36  Ⓢ
10.  PRINT (FORM1) = ((X * CON1 [ 0 ]) + Y) * 8  Ⓢ
11.  ⋮
```

The programmer wishes to change the constant 8 in statement 10. To find its location he must count the number of previous assignments and add 7.

2 FORMATS	= 8
CONSTANT ARRAY	= 3
LN. 8 constant	= 1
LN. 9 constant	= 1
Total	13 + 7 = 20

Once the location of the constant is determined the following method is employed for changing it during the run of the object program.

a) Put program in the manual mode (Ⓢ c f).
b) Type minus (-) followed by constant location and tab Ⓢ. (At this point the present contents of the location will be typed out in floating point notation. This is a good check to see if this is the location the operator desires).
c) Put punch switch "on".
d) Type in new constant followed by tab Ⓢ
e) Turn off punch switch.
f) Type minus (-) followed by location of constant and tab Ⓢ (optional for checking)
g) The changed constant will be typed out to the operator in floating point form.
h) Return to program.

11. (Q) Could the F in floating point type-out be deleted?

(A) Yes. Two G-15 words must be changed in Line 04 of the Op-Package (No. 4) for the deletion of F from the floating point typeout.

	Decimal	Hex
04:85	w.u3.82.0.06.19	y7520x3
04:62	w.64.05.1.28.28	w00579w

These changes can be made via PPR or the tape corrector routine (ASP 72). Line 4 is the 11th block with check sum = 5xv7u3x. The corrections will not effect the check sum.

12. (Q) What method should be employed to process several ALGO programs with first the Editor (Package No. 1), then the Rator/Cator (Package No. 2) and finally with Packages 3 and 4?

(A) a) The ALGO system must be on paper tape with paper tape output.

b) Load the Editor and enter the first program. When the first program has been edited, the Editor will type out "LOAD No. 2". The programmer may now type Ⓢ c f, and the Editor will permit the type-in of the next program.

This method may be continued until all the programs have been processed by the Editor.

c) Load the Rator/Cator with compute switch on breakpoint. When "MOUNT No. 1 OUTPUT" is typed out, the programmer will mount one of the Editor output tapes. This program will then be processed. When completed, the Rator/Cator may now be reloaded on breakpoint. When so instructed, the programmer will mount the next Editor output.

This method may be continued until all the edited programs have been processed by the Rator/Cator

d) Load the Lyzer/Lator (Package No. 3) with compute switch on breakpoint.

When so instructed, the programmer will mount one of the program outputs from the Rator/Cator. Further processing will continue in the normal manner as described in the ALGO Operating Instruction Manual. Following the completion of each program (i.e. processed by the Lyzer/Lator and the Op-Package, Package No. 4), the Lyzer/Lator may be loaded on breakpoint. The programmer will then mount the next Rator/Cator output.

13. (Q) How does the programmer write machine language subroutines using inputs and/or outputs in the ALGO system?

(A) Subroutines which will use inputs and outputs (i.e. alphanumeric subroutine, Decimal Tape conversion subroutine, magnetic tape subroutine) must know two things:

a) Starting location of the storage for the input or output.
b) Length of the block of data coming in or going out.

A method of denoting to the subroutine what these values are can best be shown by an example.

In the following example, the programmer wishes to enter data (numeric or alphanumeric) into the computer; perhaps convert it into binary floating point form and store it in data arrays.

Call the subroutine INPUT with check sum 0200000 and starting loc. at 00. The LIBRARY declaration would be:

```
  2.  LIBRARY   INPUT (0200000)
```

Now some data storage must also be declared. Two blocks of information are required. Call one block of 76 words A and the other block of 130 words B.

```
  3.  DATA   A(76), B(130)
```

The statement to call the subroutine for use on the 1st block would be

```
  15.   A  [0]= INPUT (76)
```

The statement to call the subroutine for use on the 2nd block would be

```
  16.   B  [0]= INPUT (130)
```

The statements written in this manner result in the length of the block in the ALGO accumulator MQ_0 or MQ_1. Store this information and by using the following sequence of commands, the location of the 1st word of the table is found.

All subroutines operate out of line 05.

```
:80  u.82.82.0.22.28   Pick up N.C.
:82  u.84.84.2.28.29   Set ovfl if store
:84  u.86.86.0.29.31   is it store command?
:86  w.89.90.0.22.18\  No. Go get next command
:90  w.94.03.0.05.29 }
:94  -7w0u804        }
:03  u.05.05.0.31.31/
:87  w.89.91.0.22.26   Yes Put in PN 1
:91  w.u5.07.3.23.31   CH to PN, Rest To ID
:07  w.09.28.2.26.28   CH to AR
:28  u.34.29.2.28.29   Shift to source
:29  w.31.33.2.25.29   Add WD + Junk
:33  u.35.E.3.05.29    - Junk   E = Exit to Program
:34  7z036y4
```

At this point the accumulator contains: w.wd.00.0.CH.00 of the 1st word. With this information and the stored length of the block, the programmer may write his machine language program and exit keeping in mind all of the conditions for writing machine language subroutines as outlined in the ALGO Operating Instructions.

14. (Q) Can decimal data from paper tape be used with ALGO?

(A) Yes. Three methods of using decimal tape input to an ALGO object program are described below.
   a) The object program accepts data from tape in binary floating point form. If the data is in decimal, the programmer may write a machine language subroutine for converting it as outlined in 13 above.
   b) Another method would be to use Intercom 500 with the flex input subroutine to convert the decimal tape to a binary tape and use the binary tape as input to the ALGO object program.
   c) Another method is described in Answer No. 15.

15. (Q) Can data be prepared in decimal floating point notation on punched tape off line on a flexowriter and read into memory by the ALGO read statement?

(A) Yes. A special use of the Decimal to Binary Conversion Routine is in conjunction with flexowriter prepared data. Data may be prepared in decimal floating point notation on punched tape off line on a flexowriter and read into memory by the ALGO read statement. These data are then converted to binary floating point by the conversion routine.

Data is punched in four-word groups (i.e., CR word tab word tab word tab word tab /) in floating point decimal notation, (i.e., EEDDDDDS where EE is the decimal exponent, DDDDD is the decimal mantissa, and S is the sign). If the number of data words is not a multiple of four, a filler word (i.e., 0000000 tab) is punched a sufficient number of times in the first four-word group in order to make the number of data words a multiple of four. If the number of data words is a multiple of four, no filler word is required. When the last four-word group of data is punched, a stop code is punched in place of a "/".

Example No. 1

Given three data words, prepare flexowriter tape. Data tape is punched as follows:

```
C/R
0000000 tab EEDDDDD tab EEDDDDD tab EEDDDDD tab stop 2
```

Example No. 2

Given eight data words prepare flexowriter tape. Data tape is punched as follows:

```
C/R
EEDDDDD tab EEDDDDD tab EEDDDDD tab EEDDDDD tab / C/R
EEDDDDD tab EEDDDDD tab EEDDDDD tab EEDDDDD tab stop 2
```

Attached is an example ALGO source program, data input and output which demonstrates the special use of the Decimal to Binary Conversion Routine.

16. (Q) It has been stated in the Programming manual that entry numbers start with 001 and continue consecutively to 511. Please clarify.

(A) The compiler assigns the entry number as the programmer enters the ALGO language program. Entry numbers start with 001 and continue consecutively. The number of entry numbers that the compiler assigns for a program will depend upon the complexity of the statements. The ALGO compiler has provisions for handling up to 511 entry numbers, however, this does not mean that the compiler will process 511 statements.

ALGO SOURCE PROGRAM

```
TITLE DEMONSTRATION OF FLEX LIBRARY ROUTINE Ⓢ
LIBRARY FLEX (0202000) Ⓢ
FORMAT TAG (S5DP5DC) Ⓢ
SUBSCRIPT I, J Ⓢ
DATA ALPHA(35) Ⓢ
BEGIN Ⓢ
START: CARR(3)  Ⓢ
READ(P) ALPHA Ⓢ
I=0 Ⓢ
J=34 Ⓢ
LOOP: HOLD=ALPHA [J] Ⓢ
ALPHA [J] =FLEX(ALPHA [I]) Ⓢ
ALPHA [I] =FLEX(HOLD) Ⓢ
I=I+1 Ⓢ
J=J-1 Ⓢ
IF J>I Ⓢ
GO TO LOOP Ⓢ
IF J=I Ⓢ
ALPHA [I]=FLEX(ALPHA [I]) Ⓢ
LAST: BELLS(5) Ⓢ
FOR I=0(1)34 Ⓢ
PRINT(TAG)=ALPHA [I] Ⓢ
END Ⓢ


   5       865
START       2
 OOP       3
 OLD     864
LAST       4


 12
864

 2    .0120013
 3    .0220024
 4    .0470048
```

DATA INPUT

```
0000000     5514161     5515361     5516637   /
5518010     5519480     5521046     5522709   /
5524466     5526316     5528258     5530293   /
5532421     5534630     5536909     5539359   /
5541679     5544169     5546739     5549400   /
5552152     5554995     5557929     5560951   /
5564059     5567253     5570532     5573897   /
5577350     5580893     5584526     5588250   /
5592064     5595990     5310005     5010424   /
```

OUTPUT

```
12 Ⓢ

14161.00000
15361.00000
16637.00000
18010.00000
19480.00000
21046.00000
22709.00000
24466.00000
26316.00000
28258.00000
30293.00000
32421.00000
34630.00000
36909.00000
39359.00000
41679.00000
44169.00000
46739.00000
49400.00000
52152.00000
54995.00000
57929.00000
60951.00000
64059.00000
67253.00000
70532.00000
73897.00000
77350.00000
80893.00000
84526.00000
88250.00000
92064.00000
95990.00000
  100.05000
       .10424
```

ACTION BY: All personnel concerned

PREPARED BY: E. J. Records

APPROVED BY: T. Yamashita