



# algebraic compiler

## for the

## BENDIX G-15

## general purpose

## digital computer

		$\sigma$	<b>71</b>	JТ	E١	JΤ	-5									
Chapter		· ·														Page
1	The Algo Language .		•										•	•		. 1
2	Numerical Data				·		•		•				•	•		. 5
3	Typewriter Input-Outp	ut												٠		. 7
4	Control Statements .								٠.	•	•				•	. 9
5	Program Completion	•													•	. 14
6	Sub-programs			•	•								•		•	. 16
7	Arrays												•			. 23
	Index								•					•		. 27
	Typical Program		·					•			·	•				. 28

Algo follows the principles laid down for the universal algebraic computer language, Algol, proposed for international use by computer organizations in America and abroad.

The Bendix Corporation is happy to cooperate with the Association for Computing Machinery and with the other members of the Algol committee in being the first manufacturer to introduce a programming system patterned on their proposal.

### THE ALGO LANGUAGE



#### INTRODUCTION

The Algo language closely parallels Algebra and may be learned in a few hours. The similarity may be seen by examining a few relationships. For example, to add quantity x to quantity y, the relationship in Algo language is: x + y; similarly,

to subtract: x - y to divide: x / y to find the sine of x:  $\sin x$  to find the logarithm of x:  $\log x$ 

Thus, without a special knowledge of programming for electronic computers, anyone with a background of high school Algebra may express a problem in the Algo language for the G-15 computer. There are two steps to obtaining a complete solution for the problem written in Algo language. First, the problem is entered into the computer and is automatically transformed by the Algo routine into a program called the "object" program, expressed in the computer's internal language. Secondly, the computer solves the problem by executing the object program.

### EXAMPLE OF ALGO LANGUAGE PROGRAM

Problem

Find:  $\mathbf{u}_i = \mathbf{e}^{\mathbf{x}_i} + \mathbf{x}_i^5 + \sin \mathbf{x}_i$  where,

 $\mathbf{x}_{i+1} = \mathbf{x}_i + .01$ 

Print  $x_i$  and  $u_i$  in floating-point notation.

The programmer specifies at the time of execution the initial value of  $\mathbf{x}$  and the limit of the subscript i.

### Algo Language Program

001. TITLE Find u [i]

002. LIBRARY SIN (0101000)

003. SUBSCript i

004. BEGIN

005. x --- KEYBD

006. FOR i = 0 (1) KEYBD BEGIN

007.  $u = EXP x + x \uparrow 5 + SIN x$ 

008. PRINT (FL) = x

009. PRINT (FL) = u

010. CARR (2)

011. x = x + ...01 END

012. END

The lines of the Algo language program may be broken into two groups, declarations and statements. The declarations are announcements to Algo regarding the contents of the program. The statements describe actions taken by Algo either while compiling the object program or during the execution of the object program. In the example, the lines numbered from 001 thru 003 are declarations, the others are statements.

The components of a statement may be: identifiers which are names for the various quantities; numbers which represent quantities whose numeric values are fixed; and operators which indicate the relationship between two quantities.

In the sample program, Line 007 has two identifiers, u and x; four operators, exp, +,  $\uparrow$ , and sin; and one number, 5.

#### ALGO ALGEBRAIC STATEMENTS

As shown, the programmer writes x + y to express the addition of x to y. He then replaces or assigns the relationship x + y to another quantity z.

$$z = x + y$$

Table 1 lists the operators for the most common arithmetic operations, the definitions and an example of each.

Operator	Definition	Example
	Replaced by	a = b
$\mathcal{L}(\mathcal{L}, \mathcal{L})$	Add	a+b
	Subtract	a — b
*	Multiply	a * b
	Divide	a/b
1	To the exponent	a↑b

Table 1

#### **PARENTHESES**

As some algebraic statements may be incorrectly interpreted, the programmer should use PARENTHESES () whenever ambiguity may result in the statement of a problem.

## Example 1

A programmer requires the evaluation of the equation:

$$d = \frac{\alpha}{bc}$$

If the Algo Algebraic statement were written:

$$d = \alpha/b*c$$
,

the equation becomes:

$$d = \left(\frac{\alpha}{b}\right) c$$

which is not the original problem.

The correct Algo algebraic statement is:

$$d = \alpha/(b*c)$$

## Example 2

A program requires the equation:

$$y = x (\alpha + b) = \alpha x + bx$$

If the algebraic statement were written:

$$y = x * \alpha + b$$

the equation would become:

$$y = \alpha x + b$$

which is not the required relationship.

The correct Algo algebraic statement is:

$$y = x * (\alpha + b)$$
 or  $y = \alpha * x + b * x$ 

The programmer may nest up to 8 pairs of parentheses, one pair within the other.

## SPECIAL ARITHMETIC OPERATIONS

The operators previously described are the foundation for computation using Algo. However, most problems require more complex operations.

Another group of operators complete the list of arithmetic operations. Table 2 lists the operators, an algebraic equation and the Algo algebraic statement of each.

The quantities following the operators in the Algo algebraic statements must be enclosed in parentheses if composed of more than one variable or if preceded by  $\alpha$  "-" sign. If the quantity is a single variable, the parentheses may be omitted; however, a space must separate the variable and the operator.

Oper- ator	Equation	Algo Algebraic Statement
	c = a +  b  z = x  y  f = a -  b + c  a =  xy /z	c = a + ABS b $z = x*ABS y$ $f = a - ABS(b + c)$ $a = ABS(x*y)/z$
SQRT SQRT SQRT SQRT	$y = (xd)^{x_0} + x + 2ac$ $z = ac^{x_0}/bd$	y = -b + SQRT(x - a*c) $y = SQRT(x*d) + x + 2*a*c$ $z = a*SQRT c/(b*d)$ $a = x*y/(SQRT z*a)$
EXP EXP EXP FXP	$y = e^{x}$ $z = 1 + e^{\alpha x}$ $y = \frac{e^{\alpha x}}{a}$ $z = e^{-x}$	y = EXP x $z = 1 + EXP (a*x)$ $y = EXP (a*x)/a$ $z = EXP(-x)$
LOG LOG LOG	y — log <sub>e</sub> x	$y = LOG x$ $a = 1/(n+1) * LOG x \uparrow (n+1)$ $b = LOG (EXP x/(1 + EXP x))$

Table 2

In this manual, upper case letters designate the characters which Algo recognizes of alphabetic operators, multi-character identifiers, and declaration names. The other characters need not be typed. For example, in the word SUBSCript, the first five characters are recognized by Algo. The programmer may or may not type the last four characters. The programmer may type either upper or lower case alphabetic characters with no difference in effect.

If the quantity following the SQRT operator is negative, an error results. If the quantity following the LOG operator is zero or negative, an error results. Errors may be detected as described in the Algo Operating Instructions.

#### LIBRARY ROUTINES

The library contains a group of trigonometric routines which are not a part of the basic Algo system. The routines in the library require a library declaration which is discussed in Chapter 5. Table 3 lists the operators, an equation and an example of each.

Oper- ator	Equation	Algo Algebraic Statement
SIN	$y = a \sin \frac{x}{a}$	y = a * SIN (x/a)
SIN	$y = \frac{1}{b}\sin(a + bx)$	y = (1/b) * SIN (a + b*x)
COS	y = cos x	y = COS x
cos	$y = \cos(x + a)$	y = COS(x + a)
COS	$y = \cos(x + a^2 + ax)$	$y = \cos(x + a \uparrow 2 + a*x)$
ARCTN	z —arctn (x + y)	z = ARCTN(x + y)
ARCTN	$a=\arctan\left(x^{2}+2x+y\right)$	$a = ARCTN(x \uparrow 2 + 2 *x + y)$

Table 3

For the sine and cosine routines, the angles must be expressed in radians. The arctangent routine finds the angle in radians.

If the quantity following the operator of each routine contains more than one quantity, the quantity must be enclosed in parentheses. If the quantity is a single variable, a space must separate the operator and the quantity.

#### **IDENTIFIER**

An identifier is a name by which something may be recognized. Identifiers may be of any length; however, only the first five characters are recognized by Algo. The first character must always be an alphabetic. Successive characters may be alphabetic or numeric. Typical identifiers are: ALPHA, BETA, X, Y, Z, AB123, and C345D.

Any arbitrary identifier, such as,  $\alpha$ , b, x, y, gamma, or epsilon, may represent a variable. As in Algebra, a variable in an Algo algebraic statement is a quantity which is free to assume different values.

A constant is a quantity whose value is fixed. Normally, a C or K represent constants in algebraic statements. However, the programmer may use any identifier desired. For instance, the programmer may wish to represent the universal constant,  $\pi$ , with the identifier, PI. A programmer may also write a constant as a numeric value in the algebraic statement.

## EVALUATION OF ALGO ALGEBRAIC STATEMENTS

Between any two relationships, expressions using  $\uparrow$ , \*, and / are evaluated before expressions using + and -. For instance,

$$y = a - b/c$$

Algo first evaluates b/c and then a - b/c.

Algo evaluates relationships involving the special arithmetic functions, library routines, and the \( \) before relationships involving \* and /.

Algo first evaluates  $b \uparrow 2$  and then  $a * b \uparrow 2$ .

As the — operator assigns the quantity from the right side to the left side, Algo performs the — operation after all other operations in an algebraic statement.

Algo evaluates algebraic statements in the same manner in which a person evaluates an algebraic equation. For example, the equation:

$$\mathbf{y} = \mathbf{\alpha}_{\mathrm{i}} + \mathbf{x}^{2} + \mathrm{SIN} \, 3\pi$$

would be in an Algo statement:

$$y = \alpha [i] + x \uparrow 2 + SIN (3*PI)$$

To find the value of y, the mathematician would find  $x^2$  and sin  $3\pi$  and then add the values of  $\alpha_1$ ,  $x^2$ , and sin  $3\pi$ . In general, the same rules apply to Algo and the order in which Algo evaluates algebraic statements is called precedence.

Table 4 lists the operators and their order of evaluation.

Order of	
Evaluation	Operators
	↑, SQRT, ABS, EXP, LOG and
Til Strike	Library Routines
2	*,/
3	+,+

Table 4

Where two operators of equal precedence follow one another, the first one encountered, when proceeding from left to right, is processed first. For example,  $\alpha*b/c$ , the relationship  $\alpha*b$  is evaluated followed by the evaluation of  $\alpha*b/c$ .

In Example 1, 
$$d = \frac{\alpha}{bc}$$

The programmer may write:  $d = \alpha/b/c$ , and the statement will be correctly interpreted by Algo.

## Example 3

Equation

Algo Language Statement

$$z = \alpha x + by \qquad z = \underbrace{a * x + b * y}_{1}$$

A close analysis of the problem shows the precedence.

- 1.  $\alpha*x$  is the first relationship on the right hand side of the = operator and is therefore evaluated first.
- 2. The + operator follows the α\*x relationship and serves to add b\*y to α\*x. However, b and y are related to one another by the \* operator which takes precedence over the + operator. Therefore, b\*y is the second evaluation.
- 3. The last evaluation is  $\alpha*x + b*y$  which is assigned to z.

## Example 4

Equation

Algo Language Statement

$$z = \alpha x^2 + 2by + c^2$$
  $z = a*x \uparrow 2 + 2*b*y + c \uparrow 2$ 

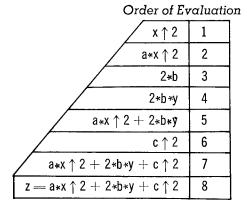


Figure 1

The order of evaluation is illustrated in Figure 1. The entire sequence of operations is as follows:

- 1. The first relationship on the right hand side of the operator, preceding from left to right, is  $\alpha_*x$ ; however, the next relationship is  $x \uparrow 2$ . From the rules previously discussed, the relationship  $x \uparrow 2$  is the first evaluation.
- 2. The next evaluation is  $\alpha *x \uparrow 2$ .
- The relationship 2\*b\*y follows and is related to α\*x ↑ 2 by the + operator. From Table 4, the \* operator has precedence over the + operator. Therefore, 2\*b and 2\*b\*y are evaluated and then added to α\*x ↑ 2.
- 4. The last two relationships are:  $c \uparrow 2$  and  $a*x \uparrow 2 + 2*b*y+c\uparrow 2$ . The  $c\uparrow 2$  relation is evaluated next.
- 5. Finally, the last relationship is  $a*x \uparrow 2 + 2*b*y + c \uparrow 2$  which is then assigned to z.

### NUMERICAL DATA



#### INTRODUCTION

Numerical data for Algo programs may be in fixed-point or floating-point notation. A fixed-point number is a number written in common decimal notation. A floating-point number is a number written in the scientific form of numerical notation, for example,  $.345 \cdot 10^4$ .

### FIXED-POINT NUMBERS

A fixed-point number may be an integer or a mixed number and may have as many as 14 digits in the entire number. The range of values is from  $10^{-14}$  up to  $10^{14}-1$ . The decimal point on the typewriter keyboard is represented by a small, hollow circle  $_{\rm o}$ , called a hollow point. The decimal point may occur anywhere in the number.

Integers

Integers are whole numbers, that is, numbers which have no fractional part. Typical integers are: 8, 23, 354, 5500, 7893000. As there is no fractional part, the programmer has the option of writing or omitting the decimal point.

Mixed Numbers

Mixed numbers have an integral part and a fractional part. If a number consists of only the fractional part, the number is considered to be a mixed number. Typical mixed numbers are: 12,5, 365,789, 553,1,64329.

Leading and trailing zeros need not be typed during data entry. The leading zero refers to the integral part of the number; the trailing zero refers to the fractional part of the number.

> 12<sub>.</sub>5 not 00000012<sub>.</sub>500000 365<sub>.</sub>789 not 00000365<sub>.</sub>789000 .683 not 0000000<sub>.</sub>6830000

#### FLOATING-POINT NUMBERS

A floating-point number has two parts: a mantissa and a characteristic. Two hollow dots,  $_{\infty}$ , represent the decimal point which separates the mantissa and the characteristic.

Mantissa	The mantissa is a decimal num-
	ber of the form: .6823; .12568943;
	.108.

Characteristic

The characteristic designates the power of 10 which multiplies the mantissa. To write the power of ten, say n, as a characteristic, the programmer adds the exponent to 50, (50 + n).

		Meaning in Common
Characteristic	50 + n	Notation
52	$50 \div 2$	$10^2 = 100$
48	50 + (-2)	$10^{-2} = .01$
50	50 + 0	$10^{\circ} = 1$
49	50 + (-1)	$10^{-1} = .1$

## FLOATING-POINT TO FIXED-POINT NUMBERS

To express a floating-point number as a fixed-point number, multiply the mantissa (decimal part) by the power of 10 indicated by the characteristic.

Floating-poi <b>nt</b>		Fixed-point
Number	Conversion	Number
52 <sub>00</sub> 125	$_{\circ}$ 125 $ imes$ 10 $^{2}$	12 <sub>°</sub> 5
51 <sub>00</sub> 6834	$_{\circ}6834  imes 10^{1}$	6 <sub>°</sub> 834
48 <sub>00</sub> 7385	$_{\circ}$ 7385 $ imes$ 10 $^{-2}$	。007385

## FIXED-POINT TO FLOATING-POINT NUMBERS

To express a fixed-point as a floating-point number:

If the number is an integer or a mixed number with both a fractional and integral part, count the number of significant places to the left of

the decimal point. The number of digits indicates the positive exponent of 10. Add the exponent to 50 to obtain the characteristic. Move the decimal point left in front of the first significant digit to obtain the fractional part, the mantissa.

If the number is a fractional number, count the number of zeros immediately to the right of the decimal point. The number of zeros indicates the negative exponent of ten. Add the negative exponent to 50 to obtain the characteristic. Move the decimal point right in front of the first nonzero digit for the fractional part, the mantissa.

Fixed- point Number	Conversion Method	Floating point Number
12,5	2 places to left of decimal point $.125 \times 10^2 = 50 + 2 + .125$	52,0125
17500	5 places to left of decimal point .17500 $ imes$ 105 $=$ 50 $+$ 5 $+$ .17500	55 <sub>00</sub> 175
。06835	$1$ zero to right of decimal point .6835 $ imes$ $10^{-1}$ $=$ 50 $ 1$ $+$ .6835	49 <sub>00</sub> 6835
。008305	2 zeros to right of decimal point .8305 $ imes$ 10 $^{-2}$ $=$ 50 $-$ 2 $+$ .8305	48,08305

### TYPEWRITER INPUT-OUTPUT



#### INTRODUCTION

Input and output may be in fixed-point or floating-point numerical notation. The input variable KEYBD permits input via the alphanumeric typewriter. The PRINT statement provides output via the alphanumeric typewriter. The PRINT statement may need a declaration, called FORMAT.

## TYPEWRITER INPUT

**KEYBD** 

KEYBD is an input variable which permits the introduction of a numerical value for a program variable. When assigned to a program variable, the KEYBD variable causes the computer to halt during the execution of the object program, to ring a bell, and to wait for the programmer to enter a value for the program variable.

## Example 5

#### Problem

Find the value of y in the equation:

$$y = \alpha x + b$$

where a, x, and b are to be supplied by the programmer during program execution. There are two possible methods of using the KEYBD variable.

Algo Program, One Method

a = KEYBD

x - KEYBD

b - KEYBD

y = a \* x + b

Algo Program, Another Method

$$y = KEYBD * KEYBD + KEYBD$$

### Example 6

## Problem

Find the slope M of a straight line which goes through (0, 1) and the point (x, y). Find the point P where the line crosses the x-axis.

$$M = \frac{y-1}{x}$$

$$P = -\frac{1}{M}$$

x and y are to be entered from the typewriter keyboard.

Algo Program

x == KEYBD

y - KEYBD

M = (y - 1)/x

P = -1/M

#### TYPEWRITER OUTPUT

PRINT () The PRINT statement causes the quantity to the right of the = operator to be typed in either floating-point or fixed-point notation.

If the type-out is in floating-point notation, the identifier in the parentheses is FL. The letter F precedes the type-out and the typewriter carriage is automatically moved to the next tab stop.

If the type-out is in fixed-point notation, the programmer must specify the form of the type-out. The form consists of the number of digits, periods, tabs or carriage returns and is called a format. A declaration identifies and specifies the form of the format. Chapter 5 discusses the format declaration.

The format identifier is enclosed in the parentheses and specifies the desired format for the fixed-point output.

## Example 7

The programmer wishes the output for Example 6 in floating-point notation.

Algo Program

x == KEYBD

y == KEYBD

M = (y - 1)/x

P = -1/M

PRINT (FL) — M

PRINT (FL) == P

### MECHANICAL OPERATIONS

The following statements cause Algo to perform mechanical operations during the execution of the object program.

BELLS (n)

PERIOd (n)

CARR (n)

TABS (n)

The number n in parentheses following each statement specifies the number of times the operation is performed. The quantity n must be a number from 1 to 15. The parentheses must be present for the statement to be interpreted correctly. The statements may occur anywhere in the Algo language program.

The BELLS statement causes a bell in the computer to ring. The PERIOd statement causes a period to be printed. The CARR and TABS statements concern movement of the typewriter carriage. CARR causes the carriage to be returned and the TABS causes the carriage to be moved to the next tab stop.

### CONTROL STATEMENTS



#### INTRODUCTION

Algo processes the statements of an Algo language program one after the other in the order written. However, statements exist which either by-pass or return to a certain section of the program or which repeat a certain section a number of times. These statements are control statements.

#### GO TO AND STOP STATEMENTS

The statements which transfer control unconditionally are GO TO and STOP statements. Associated with GO TO statements are identifiers, called labels. To process a statement out of sequence, the programmer precedes the statement with an identifier and later writes a GO TO control statement.

Algo recognizes the statement by means of the identifier, called a label. A colon, : , must immediately follow the label. Several statements illustrating the use of a label follow:

A1: 
$$b = a \uparrow 2 + SIN z$$
  
START:  $x = KEYBD$   
BETA: PRINT (FL) =  $y$ 

The statement with the preceding label is a labelled statement. Algo processes the successive statements following the labelled statement in the normal sequence.

GO TO

The GO TO statement directs Algo to by-pass or return to a section of the Algo language program. A label without the following colon follows GO TO and identifies the section to be processed. To process out of sequence the labelled statements illustrated above, the programmer would write GO TO statements as follows:

Example 8 shows the use of labels and GO TO statements in a program.

STOP

The STOP statement halts the execution of the program. Computation will proceed if the programmer moves the Compute switch on the base of the typewriter to off and then returns it to GO.

The statement may be used as an indication of the progress of the program. Another use is to temporarily halt compution while the programmer replaces a tape magazine.

### IF STATEMENTS

IF

An IF statement is the comparing of one quantity to another quantity. If the condition expressed in the IF statement is true, the compiler proceeds to the next successive statement. If the condition expressed in the IF statement is false, the compiler continues to the next successive statement.

There are three operators associated with IF statements. Table 5 lists the operators, their meaning and ilustrates their use.

Symbol	Meaning Example
	Equal to a b
	Less than a < b
	Greater than $a>b$

Table 5

The IF statement consists of the IF operator followed by a quantity related to another quantity. A space must follow the IF.

IF 
$$x = a \uparrow 2 + b$$
  
IF  $x < z + z \uparrow 2$   
IF  $x > LOG ALPHA + BETA \uparrow 2$ 

The quantity to the left of the operator may be a mathematical expression as well as the quantity to the right of the operator.

For example:

IF 
$$x \uparrow 2 + 2*x + b = a*z + y$$
  
IF  $c \uparrow 2 + EXP z < d \uparrow 2 + COS PHI$   
IF  $y + z - x > a \uparrow 2 + b*a + b \uparrow 2$ 

However, the quantity to the left of the relational operator may not contain more than one quantity if the quantity to the right is a single quantity. For instance,

IF 
$$a + b < c$$

may not be written; but

IF 
$$c > \alpha + b$$

may be written.

## BEGIN AND END STATEMENT PARENTHESES

The statement parentheses, BEGIN and END, provide a means of bracketing a segment of a program. The BEGIN corresponds to the left hand parenthesis, (, and the END corresponds to the right hand parenthesis, ).

The statement parentheses have the effect of treating  $\alpha$  group of statements as one statement,  $\alpha$  complex statement. The "true" condition of an IF statement may be made  $\alpha$  complex statement by use of the BEGIN and END statement parentheses.

The BEGIN statement parenthesis always occurs at the end of the statement immediately preceding the complex statement. The END statement parenthesis follows the last statement of a complex statement and is written on the same line as the statement. For example,

006. IF 
$$x < y$$
 BEGIN  
007.  $c = LOG x + x \uparrow 3$   
008.  $d = EXP c + c \uparrow 2$  END  
009.  $c = LOG y + y \uparrow 3$   
010.  $d = SQRT c$ 

In statement 006, the BEGIN, which follows the IF statement, indicates that the next statement consists of more than one statement. The END in statement 008 terminates the complex statement which consists of 007 and 008. Therefore, when  ${\bf x}$  is less than  ${\bf y}$ , statements 007 and 008 will be processed. When  ${\bf x}$  is equal or greater than  ${\bf y}$ , Algo skips one statement for the IF statement; the one skipped statement is a complex statement consisting of statements 007 and 008, and proceeds to statement 009.

Statement parentheses may be nested 4 deep.

Example 8

Problem

Find: 
$$f(x)$$

IF 
$$x < d$$
,  $f(x) = \alpha x^2 - bx + c$ 

IF 
$$x = d$$
,  $f(x) = 0$ 

IF 
$$x > d$$
,  $f(x) = cx^2 - bx + \alpha$ 

Algo Program

009. A1: 
$$x = KEYBD$$

010. IF 
$$x < d$$
 BEGIN

011. 
$$FX = a*x \uparrow 2 - b*x + c$$

013. IF 
$$x = d$$
 BEGIN

014. 
$$FX = 0$$

016. 
$$FX = c*x \uparrow 2 - b*x + a$$

## DISCUSSION OF EXAMPLE 8

The following discussion gives  $\alpha$  complete analysis of Example 8. As the statements 005 through 008 proceed in  $\alpha$  straightforward manner, our discussion will not spend any more time explaining them.

Statement 009 is a labelled statement. Note that the colon immediately follows the label A1. The reason for labelling statement 009 may be seen by looking at statement 020 which is a GO TO statement. After solving for one value of  $\mathbf{x}$ , the program returns to statement 009 and waits for another value of  $\mathbf{x}$  to be entered. These two statements permit the programmer to solve the problem for many values of  $\mathbf{x}$  during program execution.

Statement 010 is an IF statement which compares x to d. When x is less than d, the "true" condition, the object program proceeds to the next successive statement. Because of the statement parentheses, BEGIN and END, the compiler treats statements 011 and 012 as one statement. The compiler after evaluating FX, statement 011, proceeds to statement 012 which is a GO TO statement by-passing statements 013 through 016 and going directly to statement 017.

When x < d is false, the object program skips one statement and goes to statement 013. Note the one skipped statement is a complex statement consisting of statements 011 and 012. Statement 013, the false condition of statement 010, is another IF statement which compares x to d and which essentially asks the question, "Is x equal to d?"

When x=d, the "true" condition for statement 013, the object program proceeds to the next successive statement which is a complex statement consisting of statements 014 and 015. Statement 015 directs Algo to statement 017. Statement 016, the false condition, is by-passed because of the GO TO statement, 015.

When  $\mathbf{x}=\mathbf{d}$  is false, the object program skips one statement and proceeds to statement 016. The skipped statement is a complex statement consisting of statements 014 and 015. Statement 016 is a statement which evaluates FX for  $\mathbf{x}$  greater than  $\mathbf{d}$ . There is no IF statement involved, as  $\mathbf{x}<\mathbf{d}$  and  $\mathbf{x}=\mathbf{d}$  have been eliminated;  $\mathbf{x}$  can only be greater than  $\mathbf{d}$ .

After executing statement 012, 015 or 016, the object program proceeds to type out x and FX in floating-point notation. The object program executes two carriage returns, statement 019, and proceeds to statement 020. Statement 020 is a GO TO statement which directs Algo to return to statement 009 and accept another value of x.

## ENTRY NUMBERS

The numbers to the left of Algo statements in Example 8 are entry numbers. The compiler assigns the entry numbers as the programmer enters the Algo language program. Entry numbers have the form DDD and start with 001 and continue to 511.

## FOR STATEMENTS, SUBSCRIPT AND ARRAY DECLARATIONS

Often a programmer desires to execute one statement several times for different values of a variable. The FOR statement provides the means of repetitively executing a statement or group of statements.

FOR

The FOR statement repeats a part of the program a given number of times. The statement consists of the FOR operator and an identifier related to three quantities designated as base, difference, and limit. The form is:

FOR Identifier = Base (Difference) Limit

For each value of the identifier the FOR statement causes the statement immediately following to be repeated until the specified limit of the identifier is exceeded. When the limit is exceeded, Algo bypasses the iterated statement. Through the use of the BEGIN and END statement parentheses, the iterated statement may be a complex statement.

The number of times that the statement following the FOR statement is executed may be determined as follows:

$$Number of Executions = \frac{Limit - Base + Difference}{Difference}$$

If the quotient is a fractional number, then the number of executions is equal to the integer part of the number.

Example 9

Problem

Find: 
$$\mathbf{u}_i = \mathbf{e}^{\mathbf{x}_i} + \mathbf{x}_i^2$$
  
fox  $\mathbf{x}_1 = -2$  to  $\mathbf{x}_i = 10$   
where  $\wedge \mathbf{x} = 1.5$ 

Algo Program

FOR 
$$x = -2 (1.5) 10$$
 BEGIN  
 $u = EXP x + x \uparrow 2$   
PRINT (FL) =  $u$  END

Initially, x equals -2 and is incremented by 1.5 for each iteration. Algo repeats the complex statement following the FOR statement 9 times.

There are two methods of using a FOR statement. In one method, illustrated in Example 9, the FOR statement operates on a program variable and in the other method, on a program subscript.

In Algebra, a subscripted variable  $A_i$  where i varies from j to j+n is an array, that is, a list of quanti-

ties. In Algo, arrays may be either data or constant arrays. Both subscripts and arrays require a declaration to be used in an Algo program.

**SUBSCripts** 

The declaration announces that the identifier, or identifiers, following is for use as a subscript for program variables or for use as a counter with FOR statements. The numerical values of subscripts are always non-negative integers.

The subscript may be used to control the number of iterations through a segment of a program or to form a counter for the number of times a section of the program is executed.

A succession of up to 20 single identifiers separated by commas may follow SUBSCript.

SUBSCript i, j, k, l, m

When an identifier, declared a subscript, is used with a program variable in subsequent statements, the declared subscripts are enclosed in brackets  $[\ ]$ .

## Example 10

The equation:  $u_i = e^{x_i} + x_i^2$  is to be evaluated for i values of x.

The declaration would be:

SUBSCript i

Subsequent statements might be:

 $u[i] = EXP x[i] + x[i] \uparrow 2$ 

PRINT (ALPHA) = u [i]

DATA

A data array is a list of subscripted variables, each of which is free to assume a numeric value. The DATA declaration reserves space in memory for the array. The declaration also identifies the array and the size of the array. The form of the declaration is:

#### DATA identifier (n)

The number specifying the size of the array must be enclosed in parentheses. For instance, an array ALPHA has 5 elements. The declaration would be:

DATA ALPHA (5)

A data declaration may specify more than one array. To declare two arrays, MU and NU, each of which has 10 elements, the declaration would be:

DATA MU (10), NU (10)

CONSTant

The declaration identifies an array of constants and the size of the array. The form of the declaration is:

CONSTant Identifier (n)

The number in parentheses specifies the number of elements in the array.

When the Algo language program is entered into the computer, the computer halts after the declaration is typed and waits for the programmer to type each element of the array.

Normally, a subscript declaration must accompany an array declaration. To refer to a particular element of an array, the programmer writes the array identifier and encloses in brackets the number of the element. The computer numbers the elements from zero; therefore, to refer to the sixth element of an array BETA, the programmer writes: BETA [5].

#### FOR STATEMENT - METHOD 1

In the first method the identifier to the left of the equals operator is a subscript used as a counter. The base would then be the initial value of the counter; the difference would be the increment by which the base is increased; and the limit is the value to which the base may be increased. The numerical values for the base, difference, and limit of the subscript must be non-negative integers.

The quantities for the base, difference and limit of the FOR statement may be a variable, a subscript, a constant or the input variable KEYBD.

Base	Difference	Limit
variable	variable	variable
subscript	subscript	subscript
constant	constant	constant
KEYBD	KEYBD	KEYBD

Table 6

The quantities for the base, difference and limit in Table 6 may occur in any combination.

## Example 11

Problem

Find:

$$\begin{aligned} \mathbf{u} &= \mathbf{e}^{\mathbf{x}} \mathbf{i} + \mathbf{x}_{\mathbf{i}}^{2} \\ &\text{for } \mathbf{x}_{1} = \mathbf{0} \\ &\text{and } \triangle \mathbf{x} = .01 \\ &\text{to } \mathbf{x}_{n} = 1.00 \end{aligned}$$

A total of 101 values for u.

## Algo Program

SUBSCript i

DATA u (101)

$$x = 0$$

FOR i = 0 (1) 100 BEGIN

 $u [i] = EXP x + x \uparrow 2$ 
 $x = x + .01$  END

STOP

### FOR STATEMENT - METHOD 2

Using the second method, the identifier is a variable. The base is the initial value of the variable; the difference is the increment by which the base is increased; and the limit is the maximum amount to which the base is increased.

The programmer may specify either a variable, subscript counter, or constant for the three values B, D, and L.

		THE RESIDENCE OF THE PROPERTY	er concentration of the concen
Charles and the last of the la	CONTRACTOR OF STATE OF		
			mit -
LANCE LANCE	1 11772	erence Li	TOTAL
Base			A A A A A A A A A A A A A A A A A A A
TO THE RESERVE OF THE PARTY.		and the second of the second o	Charles and the second
erape and the second process.			The Part House Control of the Contro
		4 4	Market State of the Control of the C
		able vo	rriable
varial	ne vuu	UDIE: V	41114W
		And the second second second second	tell a beautiful to the second
GREEK AND METERS AND	and the second s		
		script su	ıbscript
conce		((1111) - 51	
subsc			
THE RESERVE OF STREET			
	A CONTRACTOR OF STATE		
const	4m+ ~~~	stant co	onstant
	Company of the Compan		
CONTRACTOR OF THE PARTY OF THE			

Table 7

The KEYBD variable may not be used when the identifier is a variable. The quantities for base, diffence, and limit in Table 7 may occur in any combination.

## Example 12

Problem

Evaluate the equation:

$$u_i = e^{x_i} + x_i^2$$

for increments of x = .01.

The programmer wishes to supply the base and limit of x at the time of program execution.

Algo Program

$$B = KEYBD$$

FOR 
$$x = B (0.01) L$$
 BEGIN

$$u = EXP x + x \uparrow 2$$

Example 13

Problem

Find:

$$\sum_{i=1}^{3} \mathbf{A}_{i} \mathbf{B}_{i}$$

Algo Program

002. DATA ALPHA (3), BETA (3)

003. SUBSCript i

004. SUM = 0

005. FOR i = 0 (1) 2

006. SUM = SUM + ALPHA [i] \* BETA [i]

007. PRINT (FL) = SUM

## PROGRAM COMPLETION



#### INTRODUCTION

In the analysis of a problem, before starting the detailed work, a person organizes the general information, such as, the precision of calculations, the project name, and the necessary reference material. The programmer, preparing an Algo language program, also organizes the general information and presents the information to Algo in the form of declarations. In the Algo language program, the declarations precede the program statements.

TITLE

The programmer supplies the title which identifies a program for future use. A program to solve for the roots of a quadratic equation might have a title as follows:

TITLE QUADRATIC

A program to solve for the surface temperatures of  $\alpha$  jet aircraft might have  $\alpha$  title:

TITLE Surface Temperature — Jet Aircraft

A title may not include the characters: (, ), and = .

LIBRAry

As the analyst requests a reference book from the library, the declaration tells Algo that a routine contained in a special library is necessary to the program.

The identifier of the routine follows LIBRAry in the declaration. Following the identifier is a code word enclosed in parentheses. The code word has the form: Oabc000, where abc differs for each library routine.

The trigonometric routines in Chapter 1 need a library declaration. The identifier for each routine and the code word are:

SIN 0101000 COS 0168000 ARCTN 0164000

Note the identifier used to declare the routine becomes the operator in the Algo algebraic statements.

To declare the sine routine, the declaration would be:

LIBRAry SIN (0101000)

More than one routine may be declared in the library declaration. For instance,

LIBRAry COS (0168000), ARCTN (0164000)

The library declaration must be the second element in an Algo language program.

Machine language routines may be added to the library. The directions are given in the Algo Operating Instructions.

**FORMAt** 

The FORMAt declaration specifies the form in which the programmer desires the data to be typed for fixed-point numerical output. The programmer indicates the form by the characters S, D, P, T, and C.

S indicates the sign of the number;

D indicates a digit;

P indicates a period;

C indicates a carriage return;

T indicates that the carriage is to be moved to the next tab stop.

The characters D, P, C, and T may be preceded by a number from 2 to 14. The number specifies the number of digits, periods, carriage returns or tabs. A single character does not need a number. A tab or carriage return may not precede a digit or digits. There may be up to 26 characters in a format.

The form of the declaration is:

FORMAt Identifier (format characters)

The declaration may contain several formats.

## Example 14

The programmer wishes output to be a mixed number with 5 places to the left of the decimal point and

3 places to the right of the decimal point. The output is to be followed by a tab.

#### Declaration

FORMAt ALPHA (S5DP3DT)

#### BEGIN AND END STATEMENTS

The BEGIN and END statements indicate the boundaries of a mathematical process. The BEGIN follows the program declarations and precedes all other statements. The END statement is the last statement in a program. The BEGIN and END statements are two distinct statements differing from the BEGIN and END statement parentheses which are part of the algebraic statements.

The BEGIN statement signals Algo that the information following is the arithmetic and operational portion of the program. The END statement indicates that the Algo language program is completed. If the BEGIN statement is omitted, Algo will detect an error while compiling the object program. If the END statement is omitted, Algo will not process the program.

The Algo Operating Instructions discuss the errors. Sub-programs must also have the BEGIN and END statements as discussed in Chapter 6.

## Example 15

Example 7 becomes a complete program with the insertion of the BEGIN and END statements and the declarations.

### Algo Program

- 001. TITLE SLOPE
- 002. FORMAT GAMMA (S3DP2DT)
- 003. BEGIN
- 004. x = KEYBD
- 005. y = KEYBD
- 006. M = (y 1)/x
- 007. P = -1/M
- 008. PRINT (GAMMA) M
- 009. PRINT (GAMMA) = P
- 010. END

## SUB-PROGRAMS



#### INTRODUCTION

In solving a complex problem without the aid of an electronic computer, standard formulas for the solution of portions of the problem may be found in reference books and the formulas substituted in the problem equations. A similar situation occurs in programming. Once a program has been written, it may be incorporated as a sub-program in another program.

An Algo language program is a process, that is, a series of actions and operations which solve a problem. An Algo sub-program is also a process, specifically, a subordinate process.

For his problem, the programmer may choose from two types of subordinate processes, Procedures and Functions. The difference between the two is the method of communication between the subordinate process and the program. The programmer must decide which type is the best method for the requirements of his problem. The differences between Procedures and Functions are discussed in detail in a later section.

Prior to use, the programmer announces the subordinate process in the general information of the program. At the same time, he writes the elements of the subordinate process.

#### SAMPLE PROGRAM

The following example is a simple application of the use of subordinate processes.

Problem

Evaluate for several values of x:

$$\mathbf{z} = \frac{\mathbf{\alpha}^2 + \mathbf{c}\mathbf{y} + \mathbf{b}^2}{\mathbf{y}}$$

where, y = ax + b

The programmer will write a subordinate process to evaluate:

$$y = \alpha x + b$$

and will call the subordinate process LINEAr.

Algo Language Program

001. TITLE GBS

002. FORMAt ALPHA (S3DP2DT)

003. PROCEdure LINEAr (f, g, h = i)

004. BEGIN

005. i = f\*g + h

006. RETURn

007. END

008. BEGIN

009. a — KEYBD

010. b = KEYBD

011. c = KEYBD

012. A1: x == KEYBD

013. LINEAr (a, x, b = y)

014.  $z = (a \uparrow 2 + c \star y + b \uparrow 2)/y$ 

015. PRINT (ALPHA) == z

016. GO TO A1

017. END

The line numbered 003 is the process declaration; the lines 003 through 007 are the process elements; line 006 is a control statement; and line 013 is a process call statement. Each of these will be discussed in detail in the accompanying sections.

#### PROCESS DECLARATIONS

The process declarations precede the statements of a program and may precede the subscript and array declarations. In the Sample Program, the process declaration for the Procedure Linear on line 003 precedes the statements of the program, lines 008 through 017.

The process declaration serves three purposes:

to specify the type, either Procedure or Function;

to identify the particular process; and

to identify any input-output variables.

The programmer writes a process declaration in one of the three forms below:

PROCEdure or FUNCTion Identifier (Input = Output)

PROCEdure or FUNCTion Identifier (= Output)

PROCEdure or FUNCTion Identifier

In the Sample Program, the process declaration on line 003 specifies a procedure whose title is Linear and whose input-output variables are f, g, h, and i. The quantities f, g, and h are the input quantities necessary to solve the problem of the subordinate process. The quantity i is the output quantity calculated by the subordinate process and transmitted to the program.

The input-output variables of the process declaration are the formal names of the parameters of the sub-ordinate process. Parentheses must always enclose the input-output variables in the process declaration. A comma must separate multiple input variables from one another and multiple output variables from one another. If the subordinate process has only output parameters, the — operator must be present.

The number of input parameters of a subordinate process does not have to equal the number of output parameters. For instance,

PROCEdure MU (x = y, z)

FUNCTion NU ( $\alpha, b = c$ )

PROCEdure RHO ( $\alpha, b, c, d = z, y, x$ )

In this manual, a "program" refers to a complete process which may include subordinate processes. A "master process" refers to the specific process which provides the inputs and uses the outputs of the subordinate process. A "process" refers to either a master or a subordinate process.

## PROCESS ELEMENTS, RETURN STATEMENTS

As a subordinate process is really a small program contained in a larger one, all the elements necessary to a program must be present in the subordinate process. The one exception is the library declaration. All library routines, used in either subordinate or master processes, must be identified in the library declaration on line 002 of the program.

Every subordinate process must have a BEGIN statement following its declarations, an END statement as the last statement, and a RETURn statement. In the Sample Program, the elements of the subordinate process are on lines 003 through 007. Note that the process declaration is one of the elements of the subordinate process as is the TITLE of a program.

RETURn

The RETURn statement directs Algo to exit from a subordinate process and to re-enter the master process. The section on Process Call Statements discusses the re-entry point.

The RETURn statement is normally the next to the last statement in a process. (The END statement is the last statement.) If the RETURn statement occurs at some other point in the process, the next to the last statement must be a GO TO statement. There may be more than one RETURn statement in a process.

The elements of a subordinate process follow the declaration and must be complete before the statements of the master process. The Sample Program illustrates the placement of the elements of both the master and subordinate processes.

## Example 16

Problem

Declare and write a procedure BETA which finds the values of  $x \le 2.5$ 

y = LOG x and for values of x > 2.5 y = 0

The statements of the subordinate process could be:

PROCEdure BETA (x = y)
BEGIN

IF  $x > 2_{\circ}5$  BEGIN y = 0RETURN END y = LOG xRETURN

END

# CHARACTERISTICS OF PROCEDURES AND FUNCTIONS

The primary difference between procedures and functions is the method of communication between the individual process and the master process. Because of the difference, procedures are designated as independent processes and functions are designated as dependent processes in the Algo language.

**PROCEdure** 

Communication between the master process and a procedure is normally limited to the procedure parameters. These parameters are the input-output variables of the PROCEdure declaration and are defined in the procedure.

In general, information contained within the procedure is not available to the master process. However, the procedure and master process may use the declared subscripts and formats of one another.

## Example 17

PROCEdure ALPHA (x, y = z)

**BEGIN** 

 $h = x \uparrow 2 + y$ 

 $i = y \uparrow 2 + x*y + x$ 

z = h + i

RETURn

**END** 

The master process may communicate with Procedure Alpha only through the input-output parameters  $\mathbf{x}$ ,  $\mathbf{y}$ , and  $\mathbf{z}$  via a process call statement. The quantities  $\mathbf{h}$  and  $\mathbf{i}$  are at no time available to the master process.

At the time of execution the procedure parameters are replaced by program variables via a process call statement.

**FUNCTion** 

Communication between the master process and a function is not limited to the function parameters. Information generated within the master is available to the function and information generated in the function is available to the master.

The master process must provide program variables for the function parameters identified in the declaration. Algo at the time of execution replaces the function parameters with the program variables.

A function and a master process may

use the declared formats, subscripts, and arrays of one another. The master must call the function at least once before using the quantities of the function in the statements of the master process.

## Example 18

If Example 17 were declared a Function, the quantities h and i would be available to the program. If the master needed the quantities h and i in an expression, Algo would supply the values of h and i calculated in the Function.

## Algo Program

Only the statements pertaining to FUNCTion ALPHA are illustrated.

005. FUNCTion ALPHA (x, y = z)

006. BEGIN

007.  $h = x \uparrow 2 + y$ 

008.  $i = y \uparrow 2 + x * y + x$ 

009. z = h + i

010. RETURn

011. END

012. BEGIN

013. a — KEYBD

014. b = KEYBD

015. c = KEYBD

016. d = KEYBD

017. ALPHA (a, b == e)

018. f = SQRT e

019. g = EXP h + LOG c + d

. . . . . .

The statement on line 019 uses the quantity h from the Function. Note that the master process has previously called the Function, statement 017.

The Function may also use the quantities of the master process. For instance, the programmer could have written for statement 007:

$$h = x \uparrow c + y$$

where c would indicate the power of x.

## PROCESS CALL STATEMENTS

A process call statement directs Algo to perform a subordinate process. There are two types of process call statements, one for subordinate processes which have input-output or only output parameters and another for subordinate processes which have no formal parameters.

A process call statement refers to a subordinate process by its identifier. The process call statement also contains the quantities in parentheses which Algo supplies for the formal parameters.

In the Sample Program, the process call statement on line 013 refers to the Procedure by its name, LINEAr. The statement also contains the quantities  $\alpha$ , x, and b which Algo supplies to the subordinate process.

The quantities in parentheses in a process call statement must occur in the sequence specified in the subordinate process declaration. For instance, in the Sample Program (see Page 16) the quantities  $\alpha$ , x, and b replace and occur in the same sequence as the formal parameters f, g, and g, respectively. If the programmer wrote:

LINEAr (a, b, 
$$x = y$$
)

Algo would assign:

a to f

b to q, and

x to h.

The evaluation would become:

$$y = \alpha b + x$$

In a process call statement, the quantities in parentheses may be program variables, numbers, or an element of an array. However, the array itself may not be used. A space must always separate the identifier and the opening parenthesis.

## Example 19

Problem

An Algo program evaluates the equations:

$$\mathbf{A} = \mathbf{e}^{2\mathrm{u}} + \mathbf{e}^{-\mathrm{u}}$$

$$B = v^2 + \log v$$

and has a Procedure SIGMA which evaluates:

$$z = x^2 + y$$

Only the statements and declarations pertaining to the PROCEdure SIGMA are illustrated.

## Algo Program

007. PROCEdure SIGMA (x, y = z)

008. BEGIN

009.  $z = x \uparrow 2 + y$ 

010. RETURn

011. END

012. BEGIN

013. U = KEYBD

014. V = KEYBD

015. C = EXPU

016. D = EXP(-U)

017. E == LOG V

018. SIGMA (C, D == A)

019. SIGMA (V, E == B)

END

When a subordinate process has only outputs, the process call statement is written without the input quantities.

BETA 
$$(== y)$$

The = operator must be present within the parentheses.

For subordinate processes having no declared parameters, the programmer writes a DO statement.

DO A DO statement directs Algo to perform a subordinate process. The identifier of the process follows the DO. For instance, a program has a procedure to ring the bell a number of times. The process call statement might be:

DO RING

A space must separate the DO and identifier.

The return statement in the subordinate process directs Algo to the statement immediately following the process call statement. In the Sample Program, Algo re-enters the master process at Statement 014.

The programmer may also use a process call statement as part of an algebraic statement. The subordinate process called in this manner may have more than one input, but may have only one output. In the Sample Program (see Page 16), the program-

mer may omit line 013, the process call statement. Instead, he may include the process call as part of the algebraic statement, line 014.

014. 
$$z = (a \uparrow 2 + c * LINEAr (a, x, b = y) + b \uparrow 2)/v$$

The process call is written only once although the quantity y appears twice in the statement. The process call must occur the first time the quantity is used.

The RETURn statement in the subordinate process causes Algo to re-enter the master process. The re-entry will be in the expression which contained the process call.

#### **NESTED PROCESSES**

A program may have subordinate processes which are nested one within the other up to 3 deep. A program may have a total of 9 subordinate processes. In our discussion, the nested processes will be regarded as being on different levels. Level 1 will indicate the outer process; level 2, the middle process; and level 3, the inner process.

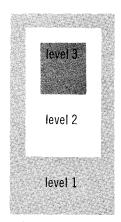
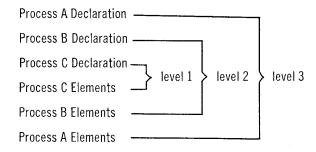


Figure 2

In Figure 2, level 2 contains level 3, and level 1 contains levels 2 and 3. Therefore, a process on level 1 becomes the master for a process on level 2 and a process on level 2 becomes the master of a process on level 3.

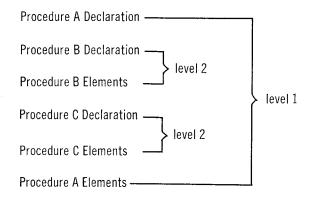
The declarations for nested processes correspond to the order of the levels. The declaration for a process on level 1 is written first; level 2 is written second; and level 3 is written last. However, as all the elements of a subordinate process are written prior to the statements of its master, the order in which the elements of nested processes are written is the reverse of the process declaration.



Note that the elements of Process B contain the elements of Process C and the elements of Process A contain the elements of Process B.

## Example 20

A program has three Procedures A, B, and C. Procedures B and C are on level 2 and are subordinate processes of Procedure A. The process declarations and elements would occur as follows:



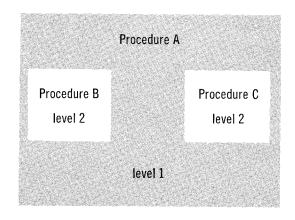
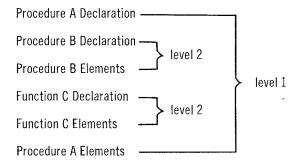


Figure 3

In a program, if there is more than one subordinate process on the same level, procedures must be declared before functions. In a nest of subordinate processes only the innermost process may be a function. A function may not be a subordinate process of another function.

In Example 20, Procedure C may be a Function. The declarations and elements would occur as follows:



## Example 21

A program has 5 subordinate processes, ABLE, BAKER, CHARLie, DOG, and EASY. Processes ABLE, CHARLie, and EASY are on level 1. ABLE and CHARLie are Procedures and EASY is a Function. BAKER and DOG are Functions on level 2 and are subordinate processes of Procedures ABLE and CHARLie, respectively.

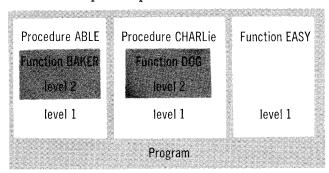
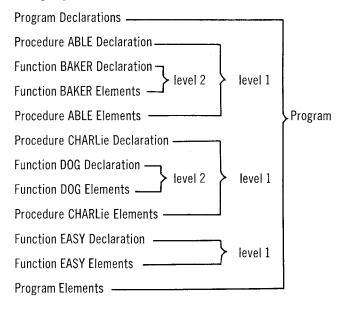


Figure 4

The program skeleton would be:



## PROCESS CALL STATEMENTS FOR NESTED PROCESSES

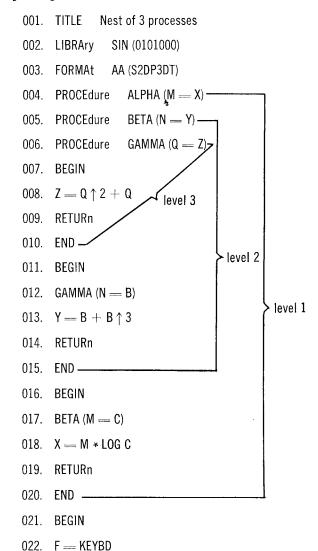
The process call statements for nested processes have the same form as the call statements for a single process.

The programmer may direct Algo to any process by writing a process call statement in the master process. For instance, a program has three procedures, ALPHA, BETA, and GAMMA, each nested one within the other. In the program, the programmer may write a process call statement for either ALPHA, BETA, or GAMMA as needed. However, the elements for Procedures ALPHA and BETA may contain a process call statement.

## Example 22

The following program is a simple program with 3 nested procedures.

## Algo Program



023. 
$$U = EXP F + ALPHA (F = G)$$

026. 
$$W = U + V$$

## DISCUSSION OF EXAMPLE 22

Statements 001 thru 006 are the declarations of the Program. Statements 006 thru 010 are the elements of Procedure GAMMA; statements 005 thru 015 are the elements of Procedure BETA; and statements 004 thru 020 are the elements of Procedure ALPHA Statements 021 thru 031 are the statements of the master process.

The elements of Procedure ALPHA contain a process call statement for Procedure BETA. The statement on line 017 also provides the quantities M and C as the input-output parameters for Procedure BETA. The quantity M is the formal input parameter for ALPHA. Thus, when Algo provides a program variable for M, the quantity also becomes the input to Procedure BETA. Similarly, the elements of Procedure BETA contain a process call statement for Procedure GAMMA. The statement, on line 012, provides the quantities N and B as the input-output parameters for Procedure GAMMA. The quantity N is the formal input parameter for BETA. Thus, the input quantity, supplied to BETA by ALPHA, becomes the input to GAMMA.

The elements of the master process have two process call statements 023 and 025. Statement 023 calls for Procedure ALPHA and provides the quantity F for the input parameter. Algo assigns the output of Procedure ALPHA to the variable G.

Calling for Procedure ALPHA has the effect of executing all three processes. Algo passes the input quantity F to the inner Procedure GAMMA through the process call statements in Procedures ALPHA and BETA.

$$F^2 + F = B$$
Output — Procedure GAMMA

 $F^2 + F + (F^2 + F)^3 = C$ 
Output — Procedure BETA

 $F (LOG (F^2 + F + (F^2 + F)^3)) = G$ 
Output — Procedure ALPHA

Statement 025 calls for Procedure GAMMA by-passing Procedures ALPHA and BETA. Algo provides the quantity H as the input parameter.

#### **ARRAYS**



#### INTRODUCTION

An array is a series or list of values which occur in a given sequence. Algo provides for "data" arrays and "constant" arrays. A data array is a list of subscripted variables, each of which is free to assume a numeric value. A constant array is a list of numbers. To declare an array, the programmer writes either a DATA or CONSTant declaration.

Both types of arrays may be one dimensional or two dimensional. A vector is an example of a one dimensional array and a matrix is an example of a two dimensional array. Each individual quantity in an array is an element. The total number of elements is the magnitude of the array.

In a two dimensional array, the elements arranged horizontally are rows and the elements arranged vertically are columns. The magnitude is equal to the number of rows times the number of columns.

B <sub>11</sub>	$B_{12}$	B <sub>13</sub>	$C_{i}$
$B_{21}$	$B_{22}$	$B_{23}$	$C_2$
$B_{31}$	$-{f B}_{32}$ .	B <sub>33</sub>	$C_3$
$\mathbf{B}_{41}$	B <sub>42</sub>	B <sub>43</sub>	$C_i$
ľwo Dime	ension	ıl Array	$\mathbf{C}_{5}$
			One Dimensional Arro

Table 8

In Table 8, B is a two dimensional array whose magnitude is 12. Array C is a one dimensional array whose magnitude is five.

DATA

The DATA declaration reserves space in memory for an array. The declaration identifies the array and the size of the array. The form of the declaration is:

DATA Identifier (n) or (m, n)

The number or numbers specifying the size of the array must be enclosed in parentheses.

A single number indicates a one dimensional array and specifies the magnitude of the array. In Table 8, array C is a one dimensional array of magnitude 5. The declaration would be:

DATA C (5)

Two numbers, m and n, separated by a comma indicate a two dimensional array. The first number indicates the number of rows in the array, and the second number indicates the number of columns. In Table 8, B is a two dimensional array. The declaration would be:

DATA B (4, 3)

A data declaration may specify more than one array. To declare both arrays B and C from Table 8, the declaration would be:

DATA B (4, 3), C (5)

CONSTant

The declaration identifies an array of constants and the size of the array.

The form of the declaration is:

CONSTant Identifier (n) or (m, n)

The number or numbers specifying the

size of the array must be enclosed in parentheses.

A single number indicates a one dimensional array and specifies the magnitude of the array. Two numbers indicate a two dimensional array; the numbers m and n indicate the number of rows and the number of columns, respectively.

When the Algo language program is entered into the computer, the computer halts after the declaration is typed and waits for the programmer to type each element of the array. The elements must be typed by column.

Normally, a subscript declaration must accompany an array declaration. If the array is two dimensional, parentheses must enclose the subscript identifiers. There may be up to 10 pairs of identifiers in a subscript declaration. Both single and paired subscripts may be present in the declaration. The form is:

SUBSCript i, j, (k, l), (m, n), q, r

To refer to a particular element of a one dimensional array, the programmer writes the array identifier followed by a number which is equal to the number of the element minus one. He encloses the number in brackets. To refer to the sixth element of an array

### A, he writes A [5].

To refer to a particular element of a two dimensional array, the programmer writes the array identifier followed by two numbers enclosed in brackets. The first number is equal to the number of the row minus one in which the element is located. The second number is equal to the number of the column minus one, multiplied by the total number of rows.

### Identifier [A, B]

where

A = i - 1 and i is the row number of the particular element.

B = m(j-1) and j is the column number of the particular element and m is the total number of rows in the array.

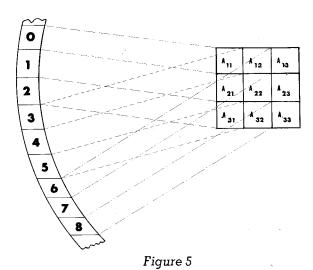
To refer to the element  $B_{32}$  from Table 8, the programmer would write: B [2, 4].

The reason for subtracting one may be seen when examining the way the array enters the computer memory.

Consider a  $3 \times 3$  matrix A:

Row 1	$\mathbf{A}_{11}$	${\tt A}_{12}$	$\mathbf{A}_{13}$
Row 2	${f A}_{21}$	${\bf A}_{22}$	$\mathbf{A}_{23}$
Row 3	${f A}_{31}$	${\bf A}_{32}$	$\mathbf{A}_{33}$
	Column 1	Column 2	Column 3

The array enters the computer memory column by column. The first element in column 1 enters some memory location assigned by Algo. The programmer need have no knowledge of the memory location and may consider it zero. The second element of the array enters the next sequential location which may be considered as one. Each successive element of the array then enters the next sequential memory location. Figure 5 illustrates the placement of the elements of the matrix on the memory drum of the computer. The number of elements in matrix A is 9. However, by numbering the elements from 0 to correspond to the memory placement, the magnitude of matrix A may now be considered as 8.



Matrix  $A_{ij}$  — Allocation in Memory

As stated previously, to refer to a particular element of an array, the programmer may write the array identifier followed by two numbers. However, the sum of these numbers will also give the proper location of the element.

Identifier [A + B]

where,

$$A = i - 1$$

$$B = m(j-1)$$

For element  $A_{23}$  of Matrix A from Figure 5, the programmer may write: A [7]. The location of element  $A_{23}$  is 7 in Figure 5. The programmer now has the option of referring to a particular element in one of two ways.

### Example 23

Problem

Multiply a vector B by a matrix A. Store results as vector C. Both A and B are entered from the keyboard. Vector C will be printed. The matrix A is a 3 by 3 matrix. The formula is:

$$\mathbf{C}_{i} = \sum_{j=1}^{3} \mathbf{A}_{ij} \times \mathbf{B}_{j}$$

Algo Program

001. TITLE MATRIX

002. FORMAt ETA (S3DP4DC)

003. DATA A (3, 3), B (3), C (3)

004. SUBSCript (I, J), K

005. BEGIN

006. FOR I == 0 (1) 2 BEGIN

007. FOR J = 0 (3) 8

008. A [I, J] = KEYBD

009. CARR (2) END

010. FOR K = 0 (1) 2

011. B [K] = KEYBD

012. CARR (2)

013. FOR I = 0 (1) 2 BEGIN

014. S = 0

015. K = 0

016. FOR J = 0 (3) 8 BEGIN

017. S = S + A[I, J] \* B[K]

018. K = K + 1 END

019. C[I] = S END

020. FOR I = 0 (1) 2

021. PRINT (ETA) - C [I]

022. END

## DISCUSSION OF EXAMPLE 23

Statements 001 and 002 are the title and format declarations. Statement 003 is an array declaration. The declaration identifies arrays A, B, and C as Data arrays. The array A is a two dimensional array of 3 rows and 3 columns. Arrays B and C are one dimensional arrays each of whose magnitude is 3. Vector C is the result of computation and the declaration reserves memory space for the array.

Statement 004 is a subscript declaration. The subscripts for the matrix A are enclosed in parentheses indicating paired subscripts. Statement 005 BEGIN indicates the beginning of the program statements.

Statement 006 is a FOR statement which establishes a counter for the subscript I. As a statement parenthesis, BEGIN, follows the FOR statement, the next successive statement for the iterative processing of I is a complex statement consisting of statements 007, 008, and 009. Statement 009 has the END statement parenthesis. Statement 007 is another FOR statement which establishes the counter for the subscript J. For each value of J, statement 008 is repeated until J exceeds the limit 8. Statement 008 is an input statement which permits the entrance of the values

for the Matrix A from the typewriter keyboard at the time the object program is executed.

Statement 009 directs the computer to execute two carriage returns. The statement parenthesis END indicates the end of the complex statement for statement 006.

To visualize what the statements 006 through 009 accomplish, first examine Figure 5. When I equals the base 0, J equals 0, 3, and 6. When J exceeds 8, statement 009 is processed and the program returns to statement 006. Therefore, when I=0 and J=0, the first element of matrix A is entered into memory location 0. To determine the memory location, add together the values of I and J. While I still equals 0, J is incremented and the second value for matrix A is entered into memory location 3. (I + J = 0 + 3).  $\boldsymbol{J}$  is again incremented to 6 and a third element of matrix A enters memory location 6. (I + J = 0 + 6). Incrementing J again, exceeds the limit 8 which directs the program to statement 009. By holding the value of I and incrementing the value for J, the elements of row 1 of the matrix were entered into the computer memory. By incrementing J by 3, the elements in row 1 entered into the proper memory locations as if the elements were entered by columns.

After executing 2 carriage returns, I is incremented by 1 and statements 007 and 008 are repeated. When I=1, the elements in row 2 of matrix A enter the memory locations 1, 4, and 7 for J=0, 3, and 6, respectively. Again, the elements enter the proper memory locations which correspond to the position of the elements in the columns of the matrix. The process is repeated a third time for I=2 and then proceeds to statement 010 when I=3, that is, exceeds 2.

Note that by incrementing J while holding the value for I, the programmer types the matrix A by row and enters the elements by column in the computer memory.

Statements 010 and 011 provide the means for entering the vector B into memory. For each value of K, a value of B enters memory until K=3 at which time statement 012 is executed.

The reason for assigning the subscript K to B may be seen by examining Statement 007. The subscript I is increased by 3 whereas the subscript for B must increase by only 1.

Statement 013 is the FOR statement which establishes a counter for I. The BEGIN statement parenthesis indicates that the statement for the iterative process is a complex statement. Statement 014 sets S equal to 0 where S is the sum. Statement 015 sets

the subscript K equal to zero.

Statement 016 is a FOR statement establishing a counter for the subscript J. The BEGIN statement parenthesis indicates the statement for the iterative process is a complex statement. Statement 017 is an algebraic statement which directs the computer to replace S with S +  $A_{i,j} \cdot B_k$ . Statement 018 is an algebraic statement which adds 1 to the value of K. The END statement parenthesis indicates the completion of the complex statement for statement 016.

Statement 019 is an algebraic statement which relates S to  $C_i$ . The END statement parenthesis signals the end of the complex statement for statement 012.

When I equals the base 0, the program has three iterations for J and K. For each iteration there is a new value for S, K, and J.

For the first iteration of I,

$$S = A_{11}B_1 + A_{12}B_2 + A_{13}B_3$$

When J exceeds the limit 8, the program processes Statement 019.

$$C_i = S = A_{11}B_1 + A_{12}B_2 + A_{13}B_3$$

The program then increments I by 1, resets S and K to zero, and repeats the iteration for J.

For the second iteration of I,

$$S = A_{21}B_1 + A_{22}B_2 + A_{23}B_3$$

Where J exceeds the limit 8, the program processes statement 019.

$$C_i = S = A_{21}B_1 + A_{22}B_2 + A_{23}B_3$$

The program then increments I by 1, resets S and K to zero, and repeats the iteration for J.

For the third iteration of I,

$$S = A_{31}B_1 + A_{32}B_2 + A_{33}B_3$$

When J exceeds the limit 8, the program processes statement 019.

$$C_i = S = A_{31}B_1 + A_{32}B_2 + A_{33}B_3$$

Now when I is incremented, I exceeds the limit 2 and the program proceeds to statement 020.

Statement 020 is a FOR statement for the subscript I. Statement 021 is an output statement which directs the computer to type the value of  $C_i$  in the format ETA. The statement is executed for three values of I. Statement 022 is the program closing statement .

#### TAPE INPUT-OUTPUT

Two statements, READ and WRITE, govern tape input-output. The statements are for use with data and constant arrays.

READ (P) The statement causes a block of data punched on paper tape to be read into the G-15 memory. An identifier follows the closing parenthesis and identifies the data on tape. In Example 23 the statement to read array A from tape would be:

Note the absence of the = operator.

The data on tape must be in floating-point binary form. The data is read in blocks of 100 words. If an array has 100 elements or less, there must be one block of tape; between 100 and 200 elements, there must be two blocks of tape; between 200 and 300 elements, there must be three blocks of tape.

WRITE (P) The statement causes an array to be punched on tape. Numbers are punched in floating-point binary form.

The array is identified by an identifier which follows the P. In Example 23, the PRINT statement may be replaced by a WRITE statement whose form would be:

WRITE (P) C

Note the absence of the = operator.

No FOR statement is necessary and the array identifier is not subscripted. The data is punched in blocks of 100 words.

## INDEX

The numbers in parentheses refer to Examples, Tables, and Figures illustrating each topic.

TITLE 14, 15 (E15)  DO (See Statements — Control)  END (See Parentheses, Statements)  Entry Numbers 10 (E8), 11  Evaluation, Order of 3 (T4), 4 (E3, E4)  Fixed-point Numbers 5  Floating-point Numbers 5  FOR (See Statements — Control)  Identifiers  Constants 3  Labels 9, 10 (E8)  Variables 3  IF (See Statements — Control)  KEYBD (See Statements — Input)  Labels (See Identifier)  Library (See Declarations)  Mechanical (See Operations)  DO 19  FOR 11 (E9), 12, 13 (E11, E12, E13)  GO TO 9, 10 (E8)  IF 9, 10 (E8)  RETURn 17 (E16)  STOP 9, 13 (E11)  Input  KEYBD 7 (E5, E6)  READ 26  Output  PRINT 7, 7 (E7)  WRITE 26  Process Call 19 (E19)  SUBSCripts (See Declaration)  Sub-programs (See Process)  TABS (See Operation — Mechanical)  Trigonometric (See Operation)  Variables (See Identifiers)	Alphabetic 2, 3 Alphanumeric (See Identifiers) Arithmetic (See Operations) Arrays (See Declarations) BEGIN (See Parentheses, Statements) BELLS (See Operation — Mechanical) Brackets 12 (E10) CARR (See Operation — Mechanical) Constants (See Identifier) Control (See Statements) Conversion Fixed-point to floating-point 5 Floating-point to fixed-point 5 Declarations Array, CONSTant One Dimensional 12 Two Dimensional 23 Array, DATA One Dimensional 12 Two Dimensional 23, 24 (E23) FORMAt 14, 15 (E14, E15) LIBRAry 14, 21 (E22) Process, FUNCTion 16,17 Process, PROCEdure 16, 17 SUBSCript Single 12 (E10)	Nested Processes (See Process) Operations Arithmetic 1 (T1) Mechanical 8 Special Arithmetic 2 (T2) Trigonometric 3 (T3) Output (See Statements — Output) Parentheses Nested 2 Use 1, 2 (E1, E2) BEGIN and END 10 (E8) PERIOd (See Operation — Mechanical) PRINT (See Statements — Output) Process Call (See Statement — Process Call) Declaration (See Declaration) Elements 17 FUNCTion, Characteristics of 18 (E18) Nested 20 (E20) PROCEdure, Characteristics of 17, 18 (E17) READ (See Statements — Input) RETURN (See Statements — Control) Statements Algebraic 1 BEGIN and END 15 (E15) Control
Entry Numbers 10 (E8), 11 Evaluation, Order of 3 (T4), 4 (E3, E4) Fixed-point Numbers 5 Floating-point Numbers 5 FOR (See Statements — Control) FUNCTion (See Process) GO TO (See Statements — Control) Identifiers Constants 3 Labels 9, 10 (E8) Variables 3 IF (See Statements — Control) KEYBD (See Statements — Input) Labels (See Identifier) LIBRAry (See Declarations)  IF 3, 10 (E8) RETURN 17 (E16) STOP 9, 13 (E11) Input KEYBD 7 (E5, E6) READ 26 Output PRINT 7, 7 (E7) WRITE 26 Process Call 19 (E19) STOP (See Statements — Control) SUBSCripts (See Declaration) Sub-programs (See Process) TABS (See Operation — Mechanical) Trigonometric (See Operation)	DO (See Statements — Control)	
Fixed-point Numbers 5 Floating-point Numbers 5 FOR (See Statements — Control) FUNCTion (See Process) GO TO (See Statements — Control) Identifiers Constants 3 Labels 9, 10 (E8) Variables 3 If (See Statements — Control) KEYBD (See Statements — Input) Labels (See Identifier) LIBRAry (See Declarations)  STOP 9, 13 (E11) Input KEYBD 7 (E5, E6) READ 26 Output PRINT 7, 7 (E7) WRITE 26 Process Call 19 (E19) STOP (See Statements — Control) SUBSCripts (See Declaration) Sub-programs (See Process) TABS (See Operation — Mechanical) Trigonometric (See Operation)	Entry Numbers 10 (E8), 11	the state of the s
Floating-point Numbers 5 FOR (See Statements — Control) FUNCTion (See Process) GO TO (See Statements — Control) Identifiers Constants 3 Labels 9, 10 (E8) Variables 3 IF (See Statements — Control) KEYBD (See Statements — Input) Labels (See Identifier) LIBRAry (See Declarations)  Input KEYBD 7 (E5, E6) READ 26 Output PRINT 7, 7 (E7) WRITE 26 Process Call 19 (E19) STOP (See Statements — Control) SUBSCripts (See Declaration) Sub-programs (See Process) TABS (See Operation — Mechanical) Trigonometric (See Operation)		
FUNCTion (See Process)  GO TO (See Statements — Control)  Identifiers  Constants 3  Labels 9, 10 (E8)  Variables 3  If (See Statements — Control)  KEYBD (See Statements — Input)  Labels (See Identifier)  LIBRAry (See Declarations)  READ 26  Output  PRINT 7, 7 (E7)  WRITE 26  Process Call 19 (E19)  STOP (See Statements — Control)  SUBSCripts (See Declaration)  Sub-programs (See Process)  TABS (See Operation — Mechanical)  Trigonometric (See Operation)	Floating-point Numbers 5	
GO TO (See Statements — Control)  Identifiers Constants 3 Labels 9, 10 (E8) Variables 3  IF (See Statements — Control) KEYBD (See Statements — Input) Labels (See Identifier) LIBRAry (See Declarations)  Output PRINT 7, 7 (E7) WRITE 26 Process Call 19 (E19) STOP (See Statements — Control) SUBSCripts (See Declaration) Sub-programs (See Process) TABS (See Operation — Mechanical) Trigonometric (See Operation)		
Constants 3 Labels 9, 10 (E8) Variables 3  IF (See Statements — Control) KEYBD (See Statements — Input) Labels (See Identifier) LIBRAry (See Declarations)  WRITE 26 Process Call 19 (E19) STOP (See Statements — Control) SUBSCripts (See Declaration) Sub-programs (See Process) TABS (See Operation — Mechanical) Trigonometric (See Operation)		
Constaints 3 Labels 9, 10 (E8)  Variables 3  IF (See Statements — Control)  KEYBD (See Statements — Input)  Labels (See Identifier)  LIBRAry (See Declarations)  Process Call 19 (E19)  STOP (See Statements — Control)  SUBSCripts (See Declaration)  Sub-programs (See Process)  TABS (See Operation — Mechanical)  Trigonometric (See Operation)		
Variables 3  IF (See Statements — Control)  KEYBD (See Statements — Input)  Labels (See Identifier)  LIBRAry (See Declarations)  STOP (See Statements — Control)  SUBSCripts (See Declaration)  Sub-programs (See Process)  TABS (See Operation — Mechanical)  Trigonometric (See Operation)		Process Call 19 (E19)
KEYBD (See Statements — Input)  Labels (See Identifier)  LIBRAry (See Declarations)  Sub-programs (See Process)  TABS (See Operation — Mechanical)  Trigonometric (See Operation)	Variables 3	
Labels (See Identifier)  LIBRAry (See Declarations)  TABS (See Operation — Mechanical)  Trigonometric (See Operation)	IF (See Statements — Control)	
LIBRAry (See Declarations)  Trigonometric (See Operation)		TABS (See Operation — Mechanical)
(0 7)	LIBBAry (See Declarations)	Trigonometric (See Operation)
		Variables (See Identifiers)

### TYPICAL PROGRAM

```
FIND: The Solution of the Differential Equation Dy/Dx = y + \sin x + \text{polynomial } (x)

where,

Polynomial (x) = A_4 + A_3x + A_2x^2 + A_1x^3 + A_0x^4

x_{\text{MIN}} = 0

\Delta x = .001

x_{\text{MAX}} = 1
```

PRINT VALUES OF X AND Y FOR INCREMENTS OF X = .01.

#### ALGO PROGRAM

THE PROGRAM IS SHOWN IN THE FORM IN WHICH IT IS TYPED INTO THE G-15 COMPUTER. THE OPERATOR TYPES THE SYMBOL TO INDICATE THE END OF EACH LINE. THE COMPUTER THEN AUTOMATICALLY RETURNS THE CARRIAGE, TYPES OUT THE NEXT ENTRY NUMBER, AND WAITS FOR THE OPERATOR TO TYPE THE NEXT LINE.

```
1. TITLE DIFFERENTIAL EQUATION SOLUTION S
 2. LIBRARY SIN (0101000) S
 3. FORMAT ESW(DP4DT), RNG(DP4DC) S
 4. PROCEDURE POLYX (R = SUM) S
 5. CONSTANT A(5) S
 6. .5 S
    ۰4 S
 7.
 8. .3 S
 9. .2 S
10.
    。1 S
11. SUBSCRIPT I S
12. BEGIN S
13. SUM = A[0] $
14. FOR I = 1(1)4 §
15. sum = sum * R + A[I] $
16. RETURN S
17. END S
18. BEGIN S
19. START: DX = .001 §
20. XPRINT = .01 S
21. x = 0 §
22. Y = 0 §
23. FY: Y = Y + DX * (Y + SIN X + POLYX (X = SHIRL))  §
24. x = x + Dx §
25. IF X < XPRINT - .005 S
26. GO TO FY S
27. PRINT (ESW) = x $
28. PRINT (RNG) = Y \circ
29. XPRINT = XPRINT + .01 S
30. IF x < 1 §
31. GO TO FY S
32. BELLS (5) S
33. END S
```



BOSTON	
CAPITOL 7-0450	607 BOYLSTON STREET BOSTON 16, MASSACHUSETTS
	LOS ANGELES
	291 S. LA CIENEGA BOULEVARD BEVERLY HILLS, CALIFORNIA
A STATE OF THE STA	OLEANDER 5-9610
CLEVELAND	
	55 PUBLIC SQUARE CLEVELAND 13, OHIO
CHERRY 1-7789	
	CHICAGO
	919 N. MICHIGAN AVENUE CHICAGO 11, ILLINOIS
	Michigan 2-6692
DENVER	
	655 BROADWAY BUILDING SUITE 910 DENVER 3, COLORADO
ALPINE 5-1403	
	DALLAS
	1511 BRYAN STREET DALLAS I, TEXAS
	RIVERSIDE 7-8805
DETROIT	
	12950 WEST EIGHT MILE ROAD DETROIT 37, MICHIGAN
JORDAN 6-8789	
	NEW YORK
	205 EAST 42ND STREET ROOM 1205 NEW YORK 17, NEW YORK OREGON 9-6990
	URESUN 9-0390
KANSAS CITY	
VALENTINE 1-8681	3430 BROADWAY KANSAS CITY 11, MISSOURI
	CANADA DEVICES OF CANADA D. D. POV. FOR OTTAWA A ONTARIO
	COMPUTING DEVICES OF CANADA P.O. BOX 508 OTTAWA 4, ONTARIO TALBOT 8-2711
	OTTAWA
SAN FRANCISCO	1330 BROADWAY SUITE 1121 OAKLAND 12, CALIFORNIA
GLENCOURT 2-3664	1550 BROADWAT SOLICITED TO CARLAND TO CARLAN
	OTHER
	COUNTRIES  BENDIX INTERNATIONAL 205 EAST 42ND STREET NEW YORK 17
	MURRAY HILL 3-1100
WASHINGTON	NEW YORK
TADIMOTOR	1000 CONNECTICUT AVENUE, N.W. WASHINGTON 6, D.C.
STERLING 3-0311	

Bendix Computer Division LOS ANGELES 45, CALIFORNIA

