Main Roads Dept.

BENDIX G-15 COMPUTER • PROGRAMMING SYSTEM

algebraic compiler

for the

BENDIX G-15

general purpose

digital computer

CONTENTS

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 by 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: $u_i = e^{x_i} + x_i^5 + \sin x_i$
where,
$x_{i+1} = x_i + .01$
Print x_i and u_i in floating-point notation.

The programmer specifies at the time of execution the initial value of 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 ↑ 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, +, ↑, 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
+	Add	a + b
−	Subtract	a − b
*	Multiply	a * b
/	Divide	a / b
↑	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:

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

If the Algo Algebraic statement were written:

d = a/b*c,

the equation becomes:

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

which is not the original problem.

The correct Algo algebraic statement is:

d = a/(b*c)

Example 2

A program requires the equation:

$$y = x(a + b) = ax + bx$$

If the algebraic statement were written:

y = x*a + b

the equation would become:

$$y = ax + b$$

which is not the required relationship.

The correct Algo algebraic statement is:

y = x * (a + b) or
y = a*x + b*x

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

Operator	*Equation*	*Algo Algebraic Statement*
ABS	$c = a + \lvert b \rvert$	c = a + ABS b
ABS	$z = x \lvert y \rvert$	z = x*ABS y
ABS	$f = a - \lvert b + c \rvert$	f = a − ABS(b + c)
ABS	$a = \lvert xy \rvert / z$	a = ABS(x*y)/z
SQRT	$y = -b + (x - ac)^{1/2}$	y = −b + SQRT(x − a*c)
SQRT	$y = (xd)^{1/2} + x + 2ac$	y = SQRT (x*d) + x + 2*a*c
SQRT	$z = ac^{1/2}/bd$	z = a*SQRT c/(b*d)
SQRT	$a = xy/z^{1/2}a$	a = x*y/(SQRT z *a)
EXP	$y = e^x$	y = EXP x
EXP	$z = 1 + e^{ax}$	z = 1 + EXP (a*x)
EXP	$y = \frac{e^{ax}}{a}$	y = EXP (a*x)/a
EXP	$z = e^{-x}$	z = EXP(−x)
LOG	$y = \log_e x$	y = LOG x
LOG	$a = \frac{1}{n+1}(\log_e x)^{n+1}$	a = 1/(n + 1) *LOG x ↑ (n + 1)
LOG	$b = \log_e \frac{e^x}{1+e^x}$	b = LOG (EXP x/(1 + EXP x))

Table 2

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 an example of the Algo algebraic statement for each equation.

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 a " − " sign. If the quantity is a single variable, the parentheses may be omitted; however, a space must separate the variable and the operator.

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.

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, a, 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, π, with the identifier, PI. A programmer may also write a constant as a numeric value in the algebraic statement.

LIBRARY ROUTINES

The Algo system has, as an added facility, an expandable, machine-language library which permits the programmer to add to the number of operations automatically performed by Algo.

The library now contains a group of trigonometric routines to find the sine, cosine, and arctangent of a quantity. Table 3 lists the operators and an equation involving the trigonometric function and an example of the Algo algebraic statement of the equation.

Operator	*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 ↑ 2 + a*x)
ARCTN	$z = \text{arctn} (x + y)$	z = ARCTN (x + y)
ARCTN	$a = \text{arctn} (x^2 + 2x + y)$	a = ARCTN(x↑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.

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

LIBRAry — Every library routine must have an identifier which follows LIBRAry in the declaration. Following the identifier is a code word enclosed in parentheses. The code word has the form: 0abc000, where abc differs for each library routine. The form of the declaration is:

```
LIBRAry  Identifier (0abc000)
```

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

```
SIN     0101000
COS     0168000
ARCTN   0164000
```

To declare the sine routine, the declaration would be:

```
LIBRAry  SIN (0101000)
```

More than one routine may be declared in a library declaration.

```
LIBRAry  COS (0168000), ARCTN (0164000)
```

The programmer should note that the routine identifier, used in the declaration, becomes the operator in the Algo algebraic statements.

The library declaration must be the second element in an Algo language program. The example on page 1 illustrates the placement of the library declaration. The directions to add machine language routines to the library are given in the "Algo Operating Instructions" manual.

EVALUATION OF ALGO ALGEBRAIC STATEMENTS

Between any two relationships, expressions using ↑ *, 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 /.

$$y = a*b \uparrow 2$$

Algo first evaluates b ↑ 2 and then a*b ↑ 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:

$$y = a_i + x^2 + \text{SIN } 3\pi$$

would be in an Algo statement:

$$y = a[i] + x \uparrow 2 + \text{SIN } (3*PI)$$

To find the value of y, the mathematician would find x^2 and sin 3π and then add the values of a_i, x^2, and sin 3π. 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*
1	↑, SQRT, ABS, EXP, LOG and Library Routines
2	*, /
3	+, −
4	=

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, a*b/c, the relationship a*b is evaluated followed by the evaluation of a*b/c.

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

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

Example 3

Equation	*Algo Language Statement*
z = ax + by	z = a*x + b*y (3; 1; 2)

A close analysis of the problem shows the precedence.

1. a*x is the first relationship on the right hand side of the = operator and is therefore evaluated first.
2. The + operator follows the a*x relationship and serves to add b*y to a*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 a*x + b*y which is assigned to z.

Example 4

Equation	*Algo Language Statement*
$z = ax^2 + 2by + c^2$	z = a*x ↑ 2 + 2*b*y + c ↑ 2

Order of Evaluation

x ↑ 2	1
a*x ↑ 2	2
2*b	3
2*b*y	4
a*x ↑ 2 + 2*b*y	5
c ↑ 2	6
a*x ↑ 2 + 2*b*y + c ↑ 2	7
z = a*x ↑ 2 + 2*b*y + c ↑ 2	8

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 a*x; however, the next relationship is x ↑ 2. From the rules previously discussed, the relationship x ↑ 2 is the first evaluation.
2. The next evaluation is a*x ↑ 2.
3. The relationship 2*b*y follows and is related to a*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 a*x ↑ 2.
4. The last two relationships are: c ↑ 2 and a*x ↑ 2 + 2*b*y + c↑2. The c↑2 relation is evaluated next.
5. Finally, the last relationship is a*x ↑ 2 + 2*b*y + c ↑ 2 which is then assigned to z.

NUMERICAL DATA

TYPEWRITER INPUT-OUTPUT

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, $.345 \cdot 10^4$.

Typewriter input-output may be in either fixed-point or floating-point notation. An input variable, called KEYBD, permits input via the alphanumeric typewriter. The PRINT statement provides output via the alphanumeric typewriter. Fixed-point output requires a declaration, called FORMAt, which specifies the form of the output data.

A program may also contain statements which cause the computer to perform mechanical operations such as printing periods or ringing the bell.

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 decimal point on the typewriter keyboard is represented by a small, hollow circle ₀, called a hollow point. The decimal point may occur anywhere in the number. The range of values is from 10^{-14} up to $10^{14} - 1$.

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: $12_{\circ}5$, $365_{\circ}789$, $_{\circ}553$, $1_{\circ}64329$.

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.

$-12_{\circ}5$	not	$-0000012_{\circ}5000000$
$365_{\circ}789$	not	$000365_{\circ}78900000$
$_{\circ}683$	not	$_{\circ}68300000000000$

FLOATING-POINT NUMBERS

A floating-point number has two parts: a mantissa and a characteristic. Two hollow dots, ₀₀, represent the decimal point which separates the mantissa and the characteristic.

Mantissa — The mantissa is a decimal number 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).

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

Typical floating-point numbers are:

$-52_{\circ\circ}125$
$51_{\circ\circ}6834$
$48_{\circ\circ}7385$

In fixed-point notation, the above numbers are:

$-12_{\circ}5$
$6_{\circ}834$
$_{\circ}007385$

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. The programmer may write the KEYBD variable in an Algo algebraic statement.

Example 5

Problem

Find the value of y in the equation:

$$y = ax + b$$

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
```

Algo accepts each KEYBD variable in the sequence in which it occurs in the program.

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
```

At the time the program is executed, the programmer enters the value for each KEYBD variable. The "+" sign must not precede a positive number.

TYPEWRITER OUTPUT

PRINT ( ) — The PRINT statement causes a quantity to be typed in either floating-point or fixed-point notation. The programmer must specify the form of the type-out by enclosing a format identifier in parentheses in the statement. The form of the type-out is:

PRINT (format identifier) = output

The quantity to the right of the = operator may be an algebraic expression.

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. The format identifier is enclosed in the parentheses after PRINT 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
```

Example 8

The programmer may write both Examples 6 and 7 as follows:

```
x = KEYBD
y = KEYBD
M = (y - 1)/x
PRINT (FL) = M
PRINT (FL) = -1/M
```

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.

A format must not specify more than 14 each of the D, P, C or T. For instance, the programmer may not specify 10DP10D in a format. He may specify 7DP7D or 10DP4D, 4DP10D, or any variation he desires.

The form of the declaration is:
FORMAt Identifier (format characters)
The declaration may specify several formats.

Example 9

In Examples 7 and 8 the programmer desires the output in fixed-point notation with 1 digit to the left of the decimal point and 3 digits to the right of the decimal point. He also wants a tab after the value of M and a carriage return after the value of P.

Algo Program:

```
.....
FORMAt ALPHA (SDP3DT), BETA (SDP3DC)
.....
x = KEYBD
y = KEYBD
M = (y - 1)/x
PRINT (ALPHA) = M
PRINT (BETA) = -1/M
.....
```

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.

INTRODUCTION

The structure of a program consists of the arithmetic and input-output statements and their necessary declarations. In addition, every program must have a TITLE declaration which identifies the program and BEGIN and END statements which indicate the boundaries of the mathematical portion of the program.

The Algo system has provisions for control statements, arrays, and sub-programs, discussed in Chapters 4, 5, and 6, respectively.

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 a jet aircraft might have a title:

```
TITLE    Surface Temperature — Jet Aircraft
```

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

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 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 10

Example 9 becomes a complete program with the insertion of the BEGIN and END statements and the TITLE declaration.

Algo Program

```
001. TITLE    SLOPE
002. FORMAt   ALPHA (SDP3DT), BETA (SDP3DC)
003. BEGIN
004. x = KEYBD
005. y = KEYBD
006. M = (y - 1)/x
007. P = -1/M
008. PRINT (ALPHA) = M
009. PRINT (BETA) = P
010. END
```

ENTRY NUMBERS

The numbers to the left of Algo statements in Example 10 are entry numbers. The compiler assigns the entry numbers as the programmer enters the Algo language program. Entry numbers start with 001 and continue consecutively to 511.

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.

A label precedes the statement to be processed out of sequence and enables Algo to recognize the statement. Labels follow the rules previously defined for identifiers. A colon, :, must immediately follow the label.

Statements, illustrating the use of a label, follow:

```
A1: b = a ↑ 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:

```
GO TO  A1
GO TO  START
GO TO  BETA
```

Example 11 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 skips the next successive statement. Algo processes each successive statement one after the other in the normal sequence.

There are three operators associated with IF statements. Table 5 lists the operators, their meaning and illustrates 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 ↑ 2 + b
IF x < z + z ↑ 2
IF x > LOG ALPHA + BETA ↑ 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 ↑ 2 + 2*x + b = a*z + y
IF c ↑ 2 + EXP z < d ↑ 2 + COS PHI
IF y + z − x > a ↑ 2 + b*a + b ↑ 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 > a + b
```

may be written.

If the single quantity to the left of the relational operator has an alphanumeric operator, such as, LOG, SQRT, or SIN, the quantity to the right may not be a single quantity.

```
IF EXP a < .01
```

may not be written; but

```
IF .01 > EXP a
```

may be written.

BEGIN AND END STATEMENT PARENTHESES

The BEGIN and END parentheses provide a means of bracketing a segment of a program and have the effect of treating a group of statements as one statement, a complex statement. The BEGIN corresponds to the left hand parenthesis (, and the END corresponds to the right hand parenthesis,).

The BEGIN and END parentheses differ from the BEGIN and END statements which are two distinct statements. The BEGIN and END parentheses are always part of the Algo statements. The BEGIN parenthesis always occurs at the end of the statement immediately preceding the complex statement. The END parenthesis follows the last statement of a complex statement and is written on the same line as the complex statement.

The "true" condition of an IF statement may be made a complex statement by use of the BEGIN and END parentheses. For example:

```
006. IF x < y  BEGIN
007. x = y
008. y = 3.14  END
009. c = LOG x + x ↑ 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 x is less than y, Algo processes statements 007, 008, 009 and 010. When x is equal or greater than y, Algo skips one statement for the IF statement and proceeds to statement 009; the one skipped statement is a complex statement consisting of statements 007 and 008.

BEGIN and END parentheses may be nested 4 deep.

Example 11

Problem

Find: f(x)

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

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

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

Algo Program

```
001. TITLE  FX COMPARISON
002. FORMAt  DON (S3DP2DT)
003. BEGIN
004. a = KEYBD
005. b = KEYBD
006. c = KEYBD
007. d = KEYBD
008. A1: x = KEYBD
009. IF x < d  BEGIN
010. FX = a*x ↑ 2 - b*x + c
011. GO TO A2  END
012. IF x = d  BEGIN
013. FX = 0
014. GO TO A2  END
015. FX = c*x ↑ 2 - b*x + a
016. A2: PRINT (DON) = x
017. PRINT (DON) = FX
018. CARR (2)
019. GO TO A1
020. END
```

DISCUSSION OF EXAMPLE 11

The following discussion gives a complete analysis of Example 11. As the statements 001 through 007 proceed in a straightforward manner, our discussion will not spend any more time explaining them.

Statement 008 is a labelled statement. Note that the colon immediately follows the label A1. The reason for labelling statement 008 may be seen by looking at statement 019 which is a GO TO statement. After solving for one value of x, the program returns to statement 008 and waits for another value of x to be entered. These two statements permit the programmer to solve the problem for many values of x during program execution.

Statement 009 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 BEGIN and END parentheses, the compiler treats statements 010 and 011 as one statement. The compiler after evaluating FX, statement 010, proceeds to statement 011 which is a GO TO statement by-passing statements 012 through 015 and going directly to statement 016.

When $x < d$ is false, the object program skips one statement and goes to statement 012. Note the one skipped statement is a complex statement consisting of statements 010 and 011. Statement 012, the false condition of statement 009, 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 012, the object program proceeds to the next successive statement which is a complex statement consisting of statements 013 and 014. Statement 014 directs Algo to statement 016. Statement 015, the false condition, is by-passed because of the GO TO statement, 014.

When $x \neq d$ is false, the object program skips one statement and proceeds to statement 015. The skipped statement is a complex statement consisting of statements 013 and 014. Statement 015 is a statement which evaluates FX for x greater than d. There is no IF statement involved, as $x < d$ and $x = d$ have been eliminated; x can only be greater than d.

After executing statement 011, 014 or 015, the object program proceeds to type out x and FX in fixed-point notation. The object program executes two carriage returns, statement 018, and proceeds to statement 019. Statement 019 is a GO TO statement which directs Algo to return to statement 008 and accept another value of x.

FOR STATEMENTS

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 by-passes the iterated statement. Through the use of the BEGIN and END 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:

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

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

Example 12

Problem

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

for $x_i = -2$ to $x_i = 10$

where $\triangle x = 1.5$

Algo Program

```
FOR x = -2 (1.5) 10  BEGIN
u = EXP x + x ↑ 2
PRINT (FL) = u  END
```

Initially, x equals −2 and Algo finds the value of u and prints the value. The value of x is then incremented by 1.5 and Algo repeats the complex state-

ment. After each iteration, Algo increments the value for x by 1.5 until the value for the limit 10 is exceeded. The total number of iterations is 9.

There are two methods of writing a FOR statement. In one method, illustrated in Example 12, the FOR statement causes a repetitive execution based on a change in a program variable and in the other method, illustrated in Example 13, based on a change in a program subscript.

In mathematics a subscript is a tool employed to differentiate one element of a series from another element. Subscripts perform the same function in Algo. Subscripts require a declaration and are discussed in detail in Chapter 5. In Algo, subscripts are enclosed in brackets [ ] in algebraic statements.

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.

FOR subscript =

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 13

Problem

Find:

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

for $x_1 = 0$

and $\triangle x = .01$

to $x_n = 1.00$

A total of 101 values for u.

Algo Program

```
x = 0
FOR i = 0 (1) 100   BEGIN
u [i] = EXP x + x ↑ 2
x = x + .01   END
STOP
```

For each value of the subscript i, Algo evaluates u. Algo performs the iteration 101 times.

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, or constant for the three values base, difference, and limit.

FOR variable =

Base	*Difference*	*Limit*
variable	variable	variable
subscript	subscript	subscript
constant	constant	constant

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 14

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
L = KEYBD
FOR x = B (.01) L   BEGIN
u = EXP x + x ↑ 2
PRINT (FL) = u   END
```

Algo evaluates the quantity u for each value of x. The values for B and L will determine the number of iterations.

Further examples of the use of the FOR statement are given in the Chapter 5 on 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_{11}	B_{12}	B_{13}	C_1
B_{21}	B_{22}	B_{23}	C_2
B_{31}	B_{32}	B_{33}	C_3
B_{41}	B_{42}	B_{43}	C_4
			C_5

Two Dimensional Array **One Dimensional Array**

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.

SUBSCript

The SUBSCript declaration announces that each identifier following SUBSCript is for use with a program variable or for use as a counter with FOR statements. As a counter, the subscript may be used to control the number of iterations through a segment of the program.

The numerical values of subscripts are always non-negative integers. A program may have a total of 20 subscripts.

However, a SUBSCript declaration may not identify more than 16 subscripts.

In the declaration, commas must separate single subscripts from one another. Subscripts may also be paired for use with two dimensional arrays. Parentheses must enclose paired subscripts. Both single and paired subscripts may be present in the declaration. The form is:

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

When an identifier, declared a subscript, is used with a program variable in subsequent statements, the declared subscripts are enclosed in brackets, [ ]. 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]. A programmer may use one subscript of a pair as a single subscript. The programmer may perform arithmetic and trigonometric operations on subscripts.

Example 15

Method

For each value of i, the program evaluates the quantity u and prints x and u. The value of x is then incremented and the evaluation repeated.

Algo Program

```
001. TITLE  FIND u [i]
002. SUBSCript  i
003. DATA  u (101)
004. BEGIN
005. x = 0
006. FOR i = 0(1)100  BEGIN
007. u [i] = EXP x + x ↑ 2
008. PRINT (FL) = x
009. PRINT (FL) = u [i]
010. x = x + .01
011. CARR (2)  END
012. END
```

Example 16

Problem

Find:

$$S = \sum_{i=1}^{3} A_i B_i$$

Method

Initially, the S is equal to zero. For each value of the subscript i, Algo replaces the value of S by the sum of S and the product of the vector elements A_i B_i.

Algo Program

```
001. TITLE  SUMMATION
002. DATA  A (3), B (3)
003. SUBSCript  i
004. BEGIN
005. S = 0
006. FOR  i = 0(1)2
007. S = S + A [i] * B [i]
008. PRINT (FL) = S
009. CARR (2)
010. END
```

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 [X, Y]

where

X = i − 1 and i is the row number of the particular element.

Y = 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 × 3 matrix A:

Row 1	A_{11}	A_{12}	A_{13}
Row 2	A_{21}	A_{22}	A_{23}
Row 3	A_{31}	A_{32}	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 2 illustrates the placement of the elements of the matrix on the memory drum of the computer. Essentially, the elements of a two dimensional array enter the computer memory one column after the other.

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.

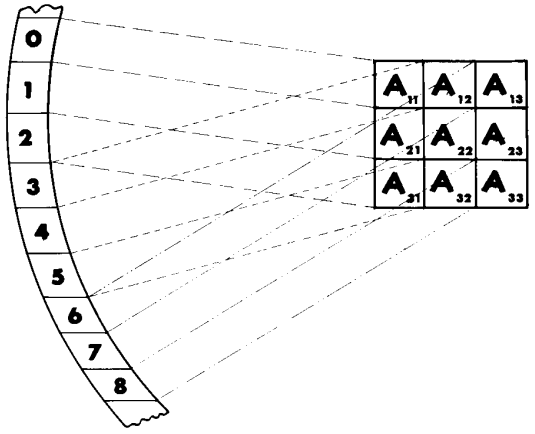


Figure 2

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 [X + Y]

where,

X = i − 1

Y = m (j − 1)

For element A_{23} of Matrix A from Figure 2, the programmer may write: A [7]. The memory location of element A_{23} is 7 in Figure 2. The programmer now has the option of referring to a particular element in one of two ways.

Example 17

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:

$$C_i = \sum_{j=1}^{3} A_{ij} \times B_j$$

Method

Each element in one row of the matrix A is multiplied by an element in the vector B. The results of the multiplication are added together to give one element of the vector C. The method involves multiplying the first element in the row by the first element of the vector; the second element in the row by the second element of the vector; and the third element in the row by the third element in the vector. When the results of each iteration are added together, the sum replaces one element of the vector C. Prior to each iteration, the sum is made equal to zero.

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 17

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 2. 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). 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 J 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,

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

When 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 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 17 the statement to read array A from tape would be:

```
READ (P) A
```

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 17, 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.

Data arrays prepared in Intercom ~~1000~~ 500X program may be used as input to an Algo program. Data arrays prepared in an Algo program may be used in an Intercom ~~1000~~ 500X program.

Floating point
single precision.

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.

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.

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:

$$z = \frac{a^2 + cy + b^2}{y}$$

where, $y = ax + b$

The programmer will write a subordinate process to evaluate:

$$y = ax + b$$

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 ↑ 2 + c*y + b ↑ 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 declaration precedes the statements of a program and may precede the subscript declaration. 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 two forms below:

PROCEdure or FUNCTion Identifier (Input — 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 subordinate 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.

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 (a, b = c)

PROCEdure RHO (a, b, c, d = z, y, x)

If an array is used in the statements of the subordinate process, the array declaration must either precede the process declaration or be part of the process elements. If an array is used in the master process, the process declaration and elements must precede the array declaration.

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 the TITLE declaration is one of the elements 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 declarations and statements of a subordinate process follow the process declaration and must be complete before any other program declarations or statements are written. The Sample Program (page 18) illustrates the placement of the elements of both the master and subordinate processes.

Example 18

Problem

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

$y = \text{LOG } x$

and for values of $x > 2.5$

$y = 0$

The statements of the subordinate process could be:

```
PROCEdure   BETA (x = y)
BEGIN
IF x > 2.5   BEGIN
y = 0
RETURn   END
y = LOG x
RETURn
END
```

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 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. A space must always separate the identifier and the opening parenthesis in a process call statement. The form of the statement is:

Process Identifier (Program Input = Output)

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 a, 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 18) the quantities a, x, and b replace and occur in the same sequence as the formal parameters f, g, and h, respectively. If the programmer wrote:

LINEAr (a, b, x = y)

Algo would assign:

a to f

b to g, and

x to h.

The evaluation would become:

$y = ab + x$

Example 19

Problem

An Algo program evaluates the equations:

$A = e^{2u} + e^{-u}$

$B = v^2 + \log v$

and has a Procedure SIGMA which evaluates:

$z = x^2 + y$

The procedure is useful since both the equations A and B may be put in its form.

Algo Program

```
      .....
007.  PROCEdure    SIGMA (x, y = z)
008.  BEGIN
009.  z = x ↑ 2 + y
010.  RETURn
011.  END
012.  BEGIN
013.  U = KEYBD
014.  V = KEYBD
015.  C = EXP U
016.  D = EXP (—U)
017.  E = LOG V
018.  SIGMA (C, D = A)
019.  SIGMA (V, E = B)
      .....
      END
```

The process call statement 018 provides the quantities C and D to the subordinate process SIGMA. Algo then uses the quantities C and D for the quantities x and y of Procedure SIGMA.

After executing Procedure SIGMA using the quantities C and D, Algo assigns the results z to the program variable A. The process call statement 019 performs the same operation using the quantities V and E for input and the quantity B for output.

In a process call statement, the quantities in parentheses may be program variables, numbers, or an element of an array. However, the programmer may not specify the entire array. For example, if the Sample Program (page 18) had an array ALPHA of 10 elements, the programmer could specify any element of the array as the input to the process. If the third element of the array were specified for the input variable g, the process call statement would be written:

```
013.  LINEAr (a, ALPHA [2], b = y)
```

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

DO A DO statement directs Algo to perform a subordinate process which has no declared inputs or outputs. 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.

In Example 19, after the first process call, Algo exits to statement 019 which is another process call. After executing Statement 019, Algo exits to Statement 020 which is not shown.

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 (page 18), the programmer 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 ↑ 2 + c * LINEAr (a, x, b = y) + b ↑ 2)/y
```

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.

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.

In both types of subordinate processes, the master process must supply quantities for process parameters identified in the declaration. Algo at the time of execution replaces the process parameters with the quantities from the master process.

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.

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

Example 20

Only the statements pertaining to PROCEdure ALPHA are illustrated.

```
.....
005.  PROCEdure  ALPHA (x, y = z)
006.  BEGIN
007.  h = x ↑ 2 + y
008.  i = y ↑ 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 (a ↑ 2 + b) + LOG c + d
.....
```

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

For instance, the quantity, a ↑ 2 + b, after the EXP operator in statement 019 is equal to the quantity h after the subordinate process has been called in statement 017. The programmer may not write EXP h when the subordinate process is a procedure.

Statement 017 causes Algo to execute the subordinate process ALPHA using the quantities a and b for x and y, respectively. Algo then assigns the evaluation for z to the variable e and exits from the subordinate process. The quantity e is equal to $a^2 + b + b^2 + ab + a$. Algo re-enters the master process at statement 018 and finds the square root of e.

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 21

If Example 20 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 ↑ 2 + y
008.  i = y ↑ 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 ↑ c + y

where c would indicate the power of x.

NESTED PROCESSES

A program may have a total of 9 subordinate processes which are nested one within the other up to 3 deep. 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.

level 2

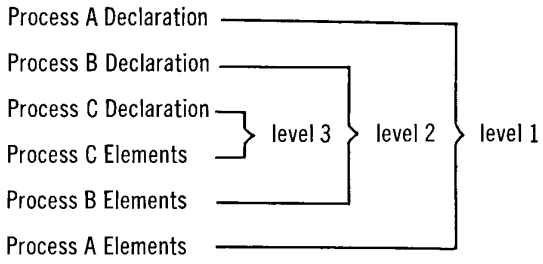
level 1

Figure 3

Figure 3 illustrates the position of nested processes in a program. The process on level 2 contains a subordinate process which is on level 3. The process on

level 2 is the master of the process on level 3. Similarly, the process on level 1 contains the process on level 2. The process on level 1 is the master; the process on level 2 is the subordinate.

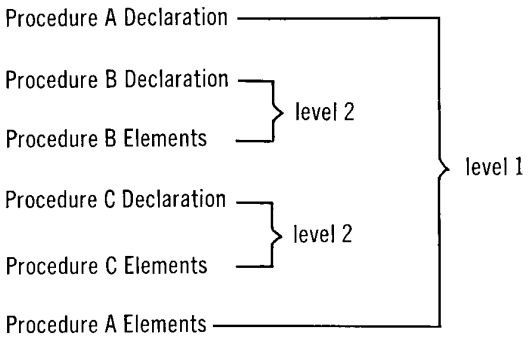
The declarations for nested processes occur in a program in the sequence which corresponds to the order of the levels. The declaration for a process on level 1 is written first; level 2 is written second; and level 3 is written last. However, as all the elements of a subordinate process are written prior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



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 22

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:



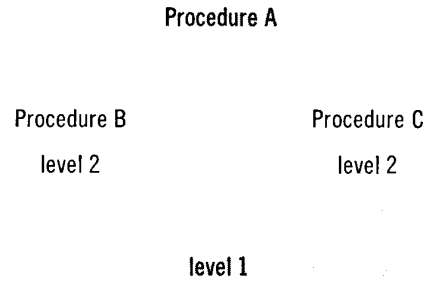
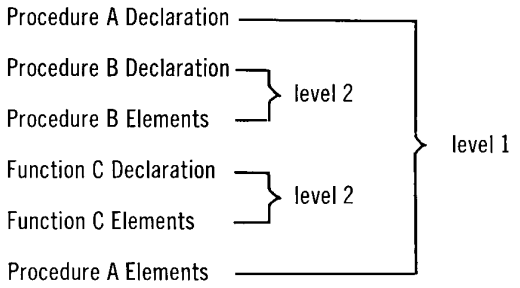


Figure 4

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



Example 23

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.

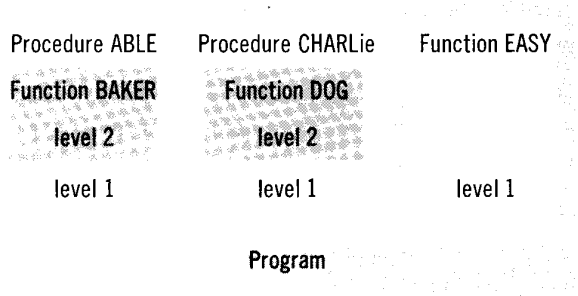
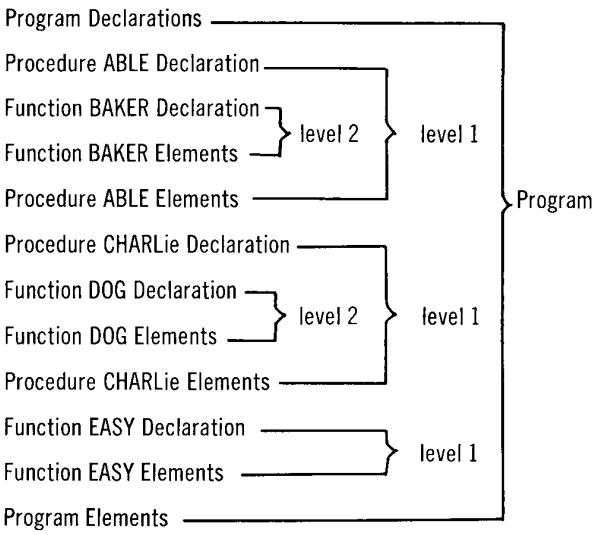


Figure 5

The program skeleton would be:



The declarations necessary to the program precede the process declarations. The programmer then declares Procedure ABLE which has a subordinate process BAKER. Process BAKER is a function and may not have a subordinate process. Therefore, the elements of BAKER immediately follow its declaration. The elements of Procedure ABLE then follow. Note that ABLE completely contains BAKER.

Processes CHARLie and DOG follow the same form. Function EASY is the last subordinate process declared and is on the same level as Procedures ABLE and CHARLie. Function EASY may not be declared before Procedures ABLE and CHARLIE. After the elements of EASY, the programmer writes the program elements.

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 24

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

Algo Program

```
001. TITLE    Nest of 3 processes
002. LIBRAry  SIN (0101000)
003. FORMAt   AA (S2DP3DT)
004. PROCEdure  ALPHA (M = X)
005. PROCEdure  BETA (N = Y)
006. PROCEdure  GAMMA (Q = Z)
007. BEGIN
008. Z = Q ↑ 2 + Q
009. RETURn
010. END
011. BEGIN
012. GAMMA (N = B)
013. Y = B + B ↑ 3
014. RETURn
015. END
016. BEGIN
017. BETA (M = C)
018. X = M * LOG C
019. RETURn
020. END
021. BEGIN
022. F = KEYBD
023. U = EXP F + ALPHA (F = G)
024. H = KEYBD
025. GAMMA (H = V)
026. W = U + V
027. PRINT (AA) = U
028. PRINT (AA) = V
029. PRINT (AA) = W
030. CARR (2)
031. END
```

DISCUSSION OF EXAMPLE 24

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. The output would be:

$$H^2 + H$$

TYPICAL PROGRAM

Problem

Find: The solution of the differential equation

$dy/dx = y + \sin x +$ polynomial (x)

where,

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

$x_{min} = 0$

$\Delta x = .001$

$x_{max} = 1$

Print values of x and y for increments of x = .01.

Method

The program establishes the initial values of x, dx and y. As the values of x and y are to be printed for increments of x equal to .01, the program establishes a variable, XPRINT, equal to .01. The program then evaluates the differential equation and increments x. The new value of x is compared to the value of XPRINT and, if less, the program again evaluates the differential equation using the new value of x. If x is equal to XPRINT, the program prints the values of x and y. After the variable XPRINT is incremented, the value of x is compared to 1. If less, the program returns to the differential equation; if greater, the computer rings the bell and halts.

The equation to evaluate the differential equation contains a process call for the procedure POLYX which evaluates the polynomial equation. The quantities x and SHIRL replace the procedure parameters R and SUM. The procedure has a constant array declaration and a subscript declaration. Initially, the procedure establishes the value of SUM equal to the constant A_0, and then evaluates the polynomial equation by means of an iteration.

Initially

$SUM = A_0$

when i = 1,

$SUM = A_0x + A_1$

when i = 2

$SUM = A_0x^2 + A_1x + A_2$

when i = 3

$SUM = A_0x^3 + A_1x^2 + A_2x + A_3$

When i = 4

$SUM = A_0x^4 + A_1x^3 + A_2x^2 + A_3x + A_4$

when the limit for the subscript i is exceeded, the procedure returns to the algebraic statement of the differential equation in the master process.

Algo Program

The program is shown in the form in which it is typed into the G-15 computer. The operator types the (S) 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)
 7. .4 (S)
 8. .3 (S)
 9. .2 (S)
10. .1 (S)
11. SUBSCRIPT I (S)
12. BEGIN (S)
13. SUM = A[0] (S)
14. FOR I = 1(1)4 (S)
15. SUM = SUM * R + A[I] (S)
16. RETURN (S)
17. END (S)
18. BEGIN (S)
19. START: DX = .001 (S)
20. XPRINT = .01 (S)
21. X = 0 (S)
22. Y = 0 (S)
23. FY: Y = Y + DX * (Y + SIN X + POLYX (X = SHIRL)) (S)
24. X = X + DX (S)
25. IF X < XPRINT - .005 (S)
26. GO TO FY (S)
27. PRINT (ESW) = X (S)
28. PRINT (RNG) = Y (S)
29. XPRINT = XPRINT + .01 (S)
30. IF X < 1 (S)
31. GO TO FY (S)
32. BELLS (5) (S)
33. END (S)
```

INDEX

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

REVISION SHEET

The following changes should be noted in the Algo Operating Instructions Manual.

1. Page 6. The 7th paragraph under "Spaces are needed" should read:

   There must be no space between a label and the following colon in a labeled statement. There must be no more than one space after the colon.

2. Page 20. The right hand column, step 7, 2nd, 3rd and 4th paragraphs, should be changed to step 8, step 9 and step 10, respectively.

SELF-CHECKING LOADER

Each of the six magazines of the Algo system has a self-checking loader which compares the check sum of a block of information against a previously determined check sum. If the two check sums do not agree, the computer reverses the tape, rings a bell, re-reads the information, and again compares the check sums.

If the checking cycle is repeated more than three times, assume the tape is in error. If tapes were prepared by means of the Housekeeper using a correction tape, use the old Algo master tapes with the Housekeeper and the correction tape, and reproduce the Algo tapes.

UPDATER

If during the updating of a program, an error is made while typing the entry number, type: o o o. The computer executes a carriage return and rings the bell twice. Type the correct entry number.

If an error is made while entering an element of a constant array, type: o o o. The computer tabs twice. Type the correct element of the constant array.

If an error is noted before the "tab (s)" while updating a statement or declaration, type: → (the upper case of the numeral 1). The computer tabs twice. Type the correct statement or declaration.

When processing statements or declarations, the Updater detects certain typing errors and tabs twice. The operator should inspect the statement or declaration for one of the errors discussed in the section on Magazine No. 1. Note that the error indication in Magazine No. 1 is different from the error indication in the Updater.

The entry numbers of the statements or declarations being modified must occur in ascending order. If the numbers are not typed in ascending order, the computer returns the carriage, types: ERROR, executes a carriage return, and rings the bell. If the entry number was mistyped, retype the correct entry. If it was not mistyped, the operator must start over with entry numbers in ascending order.