CONTROL DATA
CORPORATION

BULLETIN
PROGRAMMING SYSTEMS

ALGOL for the 1604

CONTROL DATA CORPORATION
Documentation and Evaluation Department
3145 Porter Drive
Palo Alto, California

MAY 1964

AE-16

PREFACE

This compiler was developed as a cooperative effort between the Mathematics Division of the Oak Ridge National Laboratory and Control Data Corp. Oak Ridge participants were Dr. A. A. Grau whose report "The Structure of an ALGOL Translator" formed the basis for this translator and Mr. L. L. Bumgarner who designed the compiler and directed the implementation.

CONTENTS

APPENDIX

1 PROGRAM

The format of Control Data ALGOL 60 programs is free field and can be constructed by the programmer with few restrictions. Spaces are ignored except in strings, where certain information is to be reproduced as it is originally written. The language used to write Control Data ALGOL 60 programs is based on letters of the alphabet, a through z, and the ten decimal digits, 0-9. No distinction is made between upper and lower case letters. Special additional symbols include: †

+ - * / ÷ ↑

. , : := ; $_{10}$

( ) [ ] ‘ ’

= > ≥ < ≤ ≠

≡ ⊃ ∨ ∧ ¬

† See Appendix B for symbols punched on 48 character keypunch

2 ELEMENTS OF THE LANGUAGE

The reference language is built from the following basic elements:

- delimiters
- identifiers
- numbers
- boolean values
- strings

DELIMITERS

Delimiters have fixed meanings as defined in this manual. Word delimiters used by ALGOL are underlined in this manual and are treated as individual symbols.† The following is a list of ALGOL delimiters.

Arithmetic operators:	+ - * / ÷ ↑
Relational operators:	< ≤ = ≥ > ≠
Logical operators:	≡ ⊃ ∨ ∧ ¬
Separators:	, : ; Comment
Brackets:	( ) [ ] ‘ ’ begin end
Other delimiters:	:= own boolean integer real array switch procedure format list string label value go to if then else for do step until while

IDENTIFIERS

Identifiers begin with a letter and are followed by any number of letters and/or digits. Any identifier may be used as a variable (simple or subscripted), label, switch, format, list, or procedure name.

Examples:

value	alpha	Ce3S56	sum 3
bernice	BE 156	rela	A2B3

† Delimiters are punched with apostrophes (called escape symbols) around them. (see Appendix B)

NUMBERS

Numbers in Control Data ALGOL 60 programs are written according to the following formats:

$\pm m$	$\pm m.n$	$\pm .n$
$\pm_{10}\pm E$	$\pm m_{10}\pm E$	$\pm .n_{10}\pm E$
$\pm m.n_{10}\pm E$		

The integer part m, fraction n, and exponent E consist of any combination of digits. Omitted signs are assumed positive.

A diagnostic will occur if one of the following conditions exist.

1. The total number of digits and/or symbols exceeds 64.

2. A number of the type ± m falls outside the range $\pm(2^{47}-1)$.

3. Any other number type falls outside the range $\pm 10^{\pm 308}$.

Examples:

1	-10245	+64.0025	-1.00000
.5	.999999	$-_{10}+20$	$_{10}15$
$+123_{10}12$	$13_{10}-5$	$-.5_{10}-10$	$3.549_{10}-$

BOOLEAN VALUES

Boolean values may be either <u>true</u> or <u>false.</u>

STRINGS

Strings are arbitrary sequences of basic symbols enclosed within, string quotes,‘and ’ . Strings are used as actual parameters of procedures. In particular, strings may be used in input-output operations.

Examples:

‘ delta X = ’ ‘ This is a string ’

‘ computation of the diagonal begins here ’

3 VARIABLE IDENTIFIERS

Variables are devised by the programmer to represent the values within the program. They may be either simple or subscripted; subscripted variables are covered in more detail in section 8.

Simple variables:

```
cheryl        salt
bob           sa 135 b
```

Subscripted variables:

```
form [a + 1, b]        var [i]
ab   [1,n]             box [a + i - b, c]
```

4 TYPES

Variables may be one of three types. Type integer is a variable (or constant) that does not contain a fractional part and is represented as a fixed point value in the range $|n| < 2^{-47}-1$ and zero on the 1604. Type real variables (or constants) may contain a fractional part and are represented as floating point values in the range $10^{-308} < |x| < 10^{+308}$ and zero on the 1604. Type boolean variables can only have the value true or false.

5 SIMPLE EXPRESSIONS

ARITHMETIC EXPRESSIONS

An arithmetic expression may take one of the following forms:

operand

±operand

±(arithmetic expression)

arithmetic expression arithmetic operator arithmetic expression

There are six arithmetic operators:

+(add)

-(subtract)

*(multiply)

/(real divide)

÷(integer divide)

↑(exponential)

The operators +, -, * yield type real values unless both operands are type integer. The operator / always yield type real values. The operator ÷ is defined when both operands are type integer; the result of this division is truncated and is type integer. If one or both operands involved in integer divide is type real, the real operand (operands) is replaced by entier (R + .5) before division occurs.†

The operation a ↑ b denotes exponentiation with a as the base and b as the exponent. The expression

2 ↑ n ↑ k   means $(2^n)^k$

but

2 ↑ (n ↑ m) means $2^{(n^m)}$

† .5 is added to the operand and the greatest integer not exceeding the result is used.

With i as a type integer operand, r as a type real operand, and a as an operand which may be type real or type integer, the result of exponentiation is defined by the following table:

a ↑ i	If i > 0,	a*a*..*a (i times)
	If i = 0,	if a ≠ 0, 1.0
		if a = 0, undefined
	If i < 0,	if a ≠ 0, 1/(a*a*...*a) (the denominator has i
a ↑ r	If a > 0,	exp (r*ln(a))
	If a = 0,	if r > 0, 0.0
		if r ≤ 0, undefined
	If a < 0,	undefined

The result of a ↑ b is integer only if a is an integer value and be is a positive integer constant. If a is integer a ↑ 3 results in an integer value.

Examples of expression:

ab

3

sam + table - chair

-(X*Y)/2

x ↑ 2

x ↑ y

-3 ↑ 4

a ↑ 3.5

2.1 ↑ -1.7

(3+2) ÷ (8-6)

(X+Y) - 3 * 4 - Z

(A*B-C+D/E) * F

Delta /3.1416

PRECEDENCE OF ARITHMETIC OPERATORS

Expressions are evaluated from left to right with the operators given in the following order of precedence:

first	↑
second	* / ÷
third	+ -

Expressions enclosed within parentheses are evaluated as groups within a larger expression. The value of parenthetical expressions is used in the evaluation of the larger expressions, allowing the programmer to govern the order of operations by the placement of parentheses.

Left to right evaluation is suspended temporarily when the operator in question is of lower precedence than the next operator encountered.

Examples:

The expression A+B+C*D+E is evaluated as follows:

1. A+B → R1
2. C*D → R2
3. R1 + R2 → R3
4. R3 + E

An expression containing parentheses X*(A+B)/(C+D) is evaluated as follows:

1. A+B → R1
2. X*R1 → R2
3. C+D → R3
4. R2/R3

BOOLEAN EXPRESSIONS

A boolean expression may take one of the following forms:

operand

¬ operand

¬ (boolean expression)

boolean expression <u>boolean</u> <u>operator</u> boolean expression

The five boolean operators are:

¬ (not)

∧ (and)

∨ (or)

⊃ (implies)

≡ (equivalent)

The results of the boolean operators are defined as follows:

b1	false	false	true	true
b2	false	true	false	true
¬ b1	true	true	false	false
b1 ∧ b2	false	false	false	true
b1 ∨ b2	false	true	true	true
b1 ⊃ b2	true	true	false	true
b1 ≡ b2	true	false	false	true

Examples:

¬ B

a

E ⊃ F

A ≡ B

B ∧ c ∨ d ∧ (¬E)

PRECEDENCE OF BOOLEAN OPERATORS

Boolean expressions are evaluated from left to right similarly to the arithmetic expression, operators have the following order of precedence:

first	¬
second	∧
third	∨
fourth	⊃
fifth	≡

RELATIONAL EXPRESSIONS

A relational expression takes the following form:

Arithmetic expression relational operator arithmetic expression

There are six relational operators:

< (less than)

≤ (less than or equal to)

= (equals)

≠ (does not equal)

> (greater than)

≥ (greater than or equal to)

The evaluation of a relational expression always results in a boolean value, true or false.

Examples:

$A \leq B$ (if A is less than or equal to B the result is true)

$C = D$ (if C equal D the result is true)

$A \neq \text{hand}$ (if A is not equal to hand the result is true)

Relational expressions may be connected to a boolean expression by a boolean operator'.

Examples:

$X = B \wedge C$ (if X equals B and the Boolean variable C is true, the result is true)

†$0 \leq X \wedge X \leq 1$ (if X is greater than or equal to 0 and is less than or equal to 1 the result is true)

† The mathematical notation $0 \leq x \leq 1$ is illegal in an ALGOL program and must be written as $0 \leq x \wedge x \leq 1$.

6 CONDITIONAL EXPRESSIONS

The value of an expression may be subject to a condition:

```
if B then E1 else E2
```

B is an expression which results in a boolean value and E1 and E2 are expressions.

If the boolean or relational expression B is true, the value of the expression will be E1. If the boolean or relational expression B is false, the value of the expression will be E2. E1 and E2 must be of the same type; both must result in real value, integer value, or boolean value. E2 may be a conditional expression, but E1 may be conditional only if it is enclosed within parentheses.

Example:

```
if   A < B then 1.0 else 2.1
if   ¬   Z then Bool 1 else Bool 2 ∧ Bool 1
if   C ⊃ D then (if A≠B then 2 else 3) else 4
if   C = DVE then 7.14 else if Q≡R then 3.15 else 2.8
if   A then (if B then Bool 1 else Bool 2)
     else if c then Bool 3 else Bool 4
```

7 STATEMENTS

The operations to be performed within an ALGOL program are specified by statements; each of which must terminate with a semicolon. Statements not terminated by a semi-colon are continued automatically on subsequent cards until a semicolon is encountered. Four types of statements are available.

assignment statements

goto statements

conditional statements

for statements

Statements may be included between begin and end forming compound statements. Every begin must have an end. A compound statement has the form:

begin S_1; S_2; . . . ; S_n end;

S_1; S_2; . . . S_n are statements.

ASSIGNMENT STATEMENTS

Assignment statements assign the value of the expression E to the variable V. If the expression results in a boolean value, the variable must be type boolean.

V: = E;

V is a simple or subscripted variable and the symbol : = means is replaced by:

If V is type integer and E type real the following conversion is implied.

V: = entier (E+. 5)

Examples:

A: = 3; (A is assigned the value 3)

B: = R*S/5;

C: = if I < J then 5.37 else delta; (delta must be type real in this expression)

```
C:= if Bool then 1 else if A > B then 2 else 3;
```

(If the Boolean variable, Bool, is true, C will be assigned the value of 1. If Bool is false, C will be assigned 2 if A is greater than B; otherwise, the value assigned to C will be 3.)

MULTIPLE ASSIGNMENT STATEMENTS

Multiple assignment statements assign the value of an expression E to several variables, V_1, V_2, ..., V_n. The variables must be the same type.

$V_1 : = V_2 : = \ldots : = V_n : = E;$

Examples:

```
R: = S: = T: = 5;

A: = B: = C: = D: = A*(X-Y)/Z;
```

In the first example R, S, and T must be the same type; example 1 is equivalent to:

```
T: = 5;

S: = 5:

R: =5;
```

In the second example A, B, C, and D must be the same type; example 2 is equivalent to:

```
D:= A*(X-Y)/Z;

C:= A*(X-Y)/Z;

B:= A*(X-Y)/Z;

A:= A*(X-Y)/Z;
```

GOTO STATEMENTS

Algol statements are normally executed in sequence. A goto statement may be used to interrupt this sequence. A goto statement uses labels and designational expressions.

LABELS

Labels are identifiers which name statements. In the alternate label form consisting of digits only, leading zeros are ignored. Labels must be followed by a colon and precede the statement they are labeling. Multiple labels are allowed.

Examples:

```
 24: S;      L: M: S;
004: S;     L2: 3: S;
```

DESIGNATIONAL EXPRESSIONS

Designational expressions must take one of the two forms:

```
label
if B then label else designational expression
```

The first form is a simple designational expression; the second a conditional designational expression.

A conditional designational expression may follow then if it is enclosed within parentheses.

The form of the goto statement is:

goto designational expression;

A goto statement interrupts the normal sequential execution of statements by explicitly defining the next statement to be executed. The next instruction executed will be the statement with the label contained in the designational expression.

Examples:

```
goto 8;
goto L1;
goto if ab < c then 17 else help;
```

CONDITIONAL STATEMENTS

A conditional statement may take the following form:

If B1 then S1;

B1 is an expression which results in a type boolean value. S1 is any statement. If the value of B1 is true S1 is executed, otherwise it is omitted. In either case, control passes to the next statement (unless S1 is a goto statement).

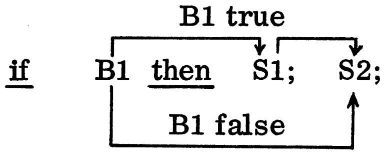
An alternate form of the conditional statement is:

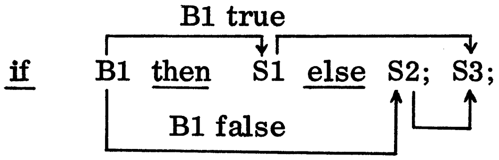
If B1 then S1 else S2;

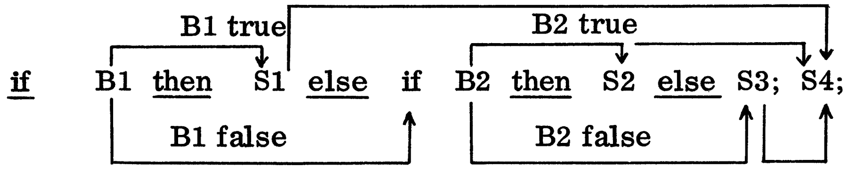
In this form, if the value of B1 is false, S2 is executed. Control then passes to the next statement.

The statement which follows then in both cases may be conditional only if it is enclosed within BEGIN and END.

The following diagrams illustrate the effects of conditional statements.







Examples:

```
(1)   if intfr < 5  then
        intfr  := intfr +.5;

(2)   if c < 1 then A:= A+1 else B:=B+1;

(3)   if intfr < 1 then begin
         if intfr > 0.5 then
            intfr  := intfr +0.5 end;

(4)   if Q ∧ 5 then A:= B-1 else if Q ∨ S
                then B:= 2 else B:= 3;
```

FOR STATEMENTS

For statements control repetition of specified statements. The general form is:

```
for V := H do S;
```

V is the loop variable and S is the statement, simple or compound, to be executed with the values of V specified by H.

H is the for list element; it can be one of two forms:

(1) E

E is any expression resulting in a value of type real or integer. S is executed with the current value of E assigned to V

(2) E_1 step E_2 until E_3

E_1 is the initial value for V; E_2 is the value by which V is incremented algebraically each time through the loop; and E3 is the limiting value. E1, E2, and E3 may be any expression resulting in values of type real or integer.

The step expression produces the same results as the following set of statements:

```
    V:= E1;

Loop:  if (V-E3)* sign (E2) > 0 then goto terminate;
       statement;
       V1=V+E2;
       goto loop;
terminate:
```

Several for list elements may appear in a single for statement.

```
for     V:= $H_1$, $H_2$, ..., $H_n$ do S;
```

Another form of the for statement allows the number of repetitions to be dete during the execution of the loop:

```
for V:= E while B do S;
```

The statement S is repeated as long as the conditional expression B is true.

The following two examples produce the same results:

```
(1)     L:      A:=B;
                if D < $10^{-5}$ then
                  begin
                    B:= A+B;
                    D:= abs(B-A);
                    goto L
                  end;
(2)     For     A:=B while D < $10^{-5}$ do
                  begin
                    B:= A+B;
                    D:= abs(B-A);
                  end;
```

The three forms of the for list element can be combined, as in:

```
for I:= 1,2,3 step 2 until 15, J while B do S;
```

The simple or compound statement S is executed for:
I = 1,2,3,5,7,9,11,13,15,J while B is true.

Examples:

```
for X:= 4 do
    A:= A+X;

for B:=1, 2, 3, do
    A:= A*B;

for W:= 1 step 2 until 400 do
    If Bool then W:= 400 else X:= Y/Z;

for W:= sin(1) step -.01 until -.6 do
    A:= A-W;

for I:= 1 step 1 until 100, 99 step - 1until 1 do
    B:= B+A[I];

for A:= if sin(.3) > cos(x) then 2 else 3
    step if B then 1.33 else 2.15
    until 456, 16 while delta > 10^5
    do
    B:= A;
```

8 DECLARATIONS

Declarations define the use of variables in Algol programs. All variables in an Algol program must be declared as to type and/or usage. Declarations may occur only after a begin or another declaration. The six kinds of declaration allowed are:

1. simple variables
2. subscripted variables
3. switches
4. procedures
5. lists
6. formats

SIMPLE VARIABLES

Simple variables represent values in Algol programs, and may be real, integer, or boolean. The form of the declaration is:

real	$V_1, V_2, \ldots, V_n;$
integer	$V_1, V_2, \ldots, V_n;$
boolean	$V_1, V_2, \ldots, V_n;$

$V_1, V_2 \ldots, V_n$ are identifiers.

The identifiers may be of arbitrary length, but only the first 64 characters are interpreted. A diagnostic occurs if an identifier is declared more than once within a block.

Examples:

```
integer N, I, GAMMA, ALF;
real ROGER, BAKER, STU, FREE;
boolean TOUR, LYON, SS, MYER;
```

SUBSCRIPTED VARIABLES

An ordered set of variables constitute an array. Array declarations must specify the maximum range of array dimensions. Such declarations permit reference by subscripting. If the type is omitted, it is assumed real. Values are assigned to array elements by input operations or assignment statements. The possible array declarations are:

array or real array P $\left[a_1:b_1,\ a_2:b_2,\ \ldots,\ a_m:b_m\right]$;

integer array AR $\left[a_1:b_1,\ a_2:b_2,\ \ldots,\ a_m:b_m\right]$;

boolean array X $\left[a_1:b_1,\ a_2:b_2,\ \ldots,\ a_m:b_m\right]$;

The number of array dimensions is m (any positive integer). The lower bounds of each dimension are specified by $a_1, a_2, \ldots, a_m$; the upper bounds by $b_1, b_2, \ldots, b_m$. Bounds values may be specified by any expression which results in a type real or integer value. Real values are rounded for subscripts by adding .5 and forming integers according to the algorithm:

I: = entier(R+.5)

Array identifiers of the same type, separated by commas, can follow the associated type declarator. Arrays of the same type, with the same dimensions, may be listed sequentially with the dimension specification after the last array identifier in a group.

Examples:

array ALMA [1:20, 1:10.5, 1:20, 3.1:8] , GEORGE [1:8000] ;

integer array BEAT, SOURCE, FILE [1:10] , RAVV [1:100] ;

integer array SOON [-2000: -1000] , LATE [-1: -10,.08:9.4] ;

array SING, FLAK [r:ww] ;

array SIS [0:10, 0:10] , GRIT [sin(x) :arctan(x)] ;

boolean array SORT [-k:p] , ZIRT [1:m] ;

Array elements may be referenced in an expression (or may appear on the left-hand side of an assignment symbol) by writing the array name with a list of subscripts.

Examples:

```
    A [i,j] :=4;
    B:= W [i] + X [k,l,m,n]/Y[i,i] ;
    C:= X [i] -4.5;
```

SWITCHES

A switch consists of an ordered set of designational expressions. The switch declaration takes the following form:

Switch V:= DE_1, DE_2, . . . , DE_n;

V is the switch identifier and DE_1, DE_2, . . . , DE_n are designational expressions. A switch element must be referenced by using the switch identifier with the appropriate subscript. Only subscript values 1 through n are meaningful; values outside this range produce undefined results. A subscripted switch identifier is a designational expression.

Example:

```
    switch SWD:=ENT, RENT, BENT, EXIT;
         .
         .
         .
    goto SWD [2] ;
    goto SW [if B then 1+sin(x) else 3.0] ;
```

The following two sets of statements perform the same function:

```
    1. Integer X;
            .
            .
            .
       if X ≤0 ∨ X>4 then goto undefined else
       if X = 1 then goto L1 else
       if X = 2 then goto L2 else
       if X = 3 then goto L3 else
       goto L4;
```

```
2.  switch S:= <1, <2, <3, <4;
          .
          .
          .
       goto S [X] ;
```

Switch elements may also be used in a switch list.

Example:

```
    switch VECTOR : = STA, TEM, ENT;
    switch TRSFR: = RTHV, VECTOR [3] , VECTOR [1] , VECTOR [2] ;
          .
          .
          .
    i : = 3;
    goto  TRSFR [ i ]  ;
          .
          .
          .
```

The current value of i is 3, so that reference is made to the third designational expression in the switch declaration TRSFR. The third designational expression in TRSFR transfers control to VECTOR [1] , which transfers control to statement STA.

Switch subscripts resulting in type real values are converted to type integer according to the formula:

```
    I : = entier (R+0.5)
```

9 COMMENTS

Comments may be inserted anywhere in the program if they are enclosed within the separators comment and a semicolon.

Examples:

```
Begin comment this is a comment;
    S1, S2, . . . Sn;
end;
S:= A; comment assign A to S;
```

Any information following an end and preceding another end, else, or a semicolon is treated as a comment.

Examples:

```
end this ends the first statement end;
end this ends the then part else
end the whole program is finished;
```

10 BLOCKS

A block starts with begin, includes declarations and statements and terminates with end. It takes the following form:

begin $D_1, D_2, \ldots, D_n$;
        $S_1, S_2, \ldots, S_m$;

end;

where D_1 through D_n are declarations and S_1 through S_m are simple or compound statements or blocks.

A block contained within a block is called a subordinate block. A block containing subordinate blocks is called a dominant block. Subordinate blocks appear in the statement portion of related dominant blocks, and subordinate blocks may in turn contain nested subordinate blocks. Blocks may be nested to any level. Complete blocks which appear in sequence form parallel blocks.

Examples of block nesting:

```
BLOCK A: begin
    declarations
    statements
        BLOCK B: begin
            declarations
            statements
            end BLOCK B;
        BLOCK C: begin
            declarations
            statements
            end BLOCK C;
    statements
    end BLOCK A;
```

Block A is dominant; Block B is parallel to Block C and both are subordinate to Block A.

Identifiers declared within a block are local to that block and may not be referenced by a dominant block. Identifiers declared within a dominant block may be referenced within a subordinate block and are non-local to the subordinate block. If the same identifier is declared within a dominant and a subordinate block:

1) Its use within the subordinate block is defined by the local declaration.
2) The dominant block identifier may not be referenced within the subordinate block.
3) Upon leaving a subordinate block, the identifier reassumes the dominant block usage.

A label is declared by its occurrence and is valid for the entire block in which it occurs.

Examples:

```
Q:  begin integer i,k; real w; array A [1:10, 1:10] ;
        for i:= 1 step 1 until 10 do
        for k:= i+1 step 1 until 10 do
        begin s:=A [i, k] ;
                  A [i, k]:= A [k, i];
                  A [k, i]:= w
        end loop i and k
    end block Q;
    begin
        integer A, B;
        real    C;
            A:= 5 * 6 - 4;
            Begin
                integer A;
                boolean D;
                D:= true;
                A:= if D then 7 else 5;
                B:= 6;
                print (A, B);
            end;
                C:= A ** 2;
                print (A, C);
            end;
```

The results obtained from executing this program would be:

```
 7          6
26        676
```

11 OWN

Storage for variables is assigned at the time of block entry. When exiting from a block, the variable storage is released for subsequent blocks. As a result, variable values are always undefined upon block entry. Storage may be permanently reserved for variables by the addition of the own declarator to the declaration.

Examples:

```
own real A;
own boolean B, C;
own array P, Q [1:5, -3: 1] ;
own integer array A [0:3] ;
```

Array storage is computed dynamically upon block entry for non-own arrays. Array storage for own arrays is allocated only at the time the block is first entered. The first occurrence of bounds declarations for own arrays should indicate the maximum bounds that the array will assume.

12 PROCEDURE DECLARATIONS

Statements which occur several times within a program may be written once as a procedure and the procedure may be referenced each time the series of statements is to be executed. A list of parameters makes it possible for a procedure to be used with varying values and/or variables. A procedure is divided into two parts: the heading and the body.

HEADING

The heading consists of a procedure identifier, a formal parameter list (if any), and specifications (if any).

a) A procedure identifier is the name given to the procedure in the program.

b) Formal parameters are those parameters within the body which will be replaced by actual parameters each time the procedure is executed.

A procedure name and list of formal parameters is defined as:

procedure identifier $(P_1, P_2, \ldots, P_n)$;

comments may be included in the form:

procedure identifier $(p_1, p_2, \ldots, p_m)$comment: $(p_{m+1}, \ldots, p_n)$;

Procedures to be compiled with an ALGOL program must appear in the declaration part of the block or a dominant block in which the procedure is referenced. Procedures which have been compiled independently or with another program may be declared external and used as binary input to avoid recompilation. (See external declaration, section 16.)

The following is a list of specifiers:

value, real, integer, boolean, array, integer array, boolean array, procedure, real procedure, integer procedure, boolean procedure, switch, string, list, format.

The use of a specification takes a form similar to a declaration. However, the purpose of a specification is to indicate the type or usage of an actual parameter and does not constitute a declaration.

The form of specification is:

specifier $FP_1, FP_2, \ldots, FP_n$;

FP1 through FPn are a list of formal parameters. A parameter specified value is a call-by-value parameter. Parameters not specified value are call-by-name parameters. The expression corresponding to a formal parameter specified value is computed upon procedure entry. The resulting value is assigned to the formal parameter before execution of the procedure body. This value is local to the procedure. A value parameter should not appear on the left of an assignment symbol. Parameters specified as value should also be specified as to type. Arrays may not be specified value. If the type of a formal parameter representing an arithmetic quantity is not declared in a specification, it is assumed real. A specification is required only for integer arithmetic parameters, and non-ALGOL delimiters such as list and format; however, full use of specifications is desirable for descriptive purposes and for optimization.

Examples of procedure heads:

```
procedure P(A, B, C, D) results: (E, F);
value C;
integer A, C;
real E, F: Procedure D;
integer array B;
```

P is the procedure identifier. A, B, C, D, E, and F are the formal parameters. The word, results, is a comment. The actual parameter corresponding to A is an integer expression, B corresponds to an integer array name, C is an integer expression which is evaluated once at the time of procedure entry (this value is used throughout the procedure whenever C appears in an expression), D corresponds to a procedure name, and E and F correspond to real expressions.

```
procedure triangle (X, Y, Z, S, TO, W);
    value S, TO;
    integer X, S;
    switch Z;
    boolean array W;
    boolean TO;
    real array Y;
```

BODY

The procedure body consists of a simple or compound statement or block.

A procedure is similar to a block in that a list of formal parameters defines identifiers which are local to the procedure. Identifiers within the procedure may not be referenced outside the procedure. A formal parameter identifier may be redefined within a procedure body by appearing as a label or in a declaration statement. The formal parameter cannot be referenced within the scope of the new definition.

Global variables are declared outside the procedure declaration in the same or a dominant block, and may be referenced directly in the procedure body. Formal parameters with the same name as the global variables supersede the global variables when referenced within the procedure body.

Formal parameters not appearing in the value list are replaced throughout the procedure body by the corresponding actual parameters and are call-by-name parameters. This means that actual parameters that correspond to expressions are evaluated each time the formal parameter is encountered. Formal parameters that appear to the left of an assignment symbol must correspond to an actual parameter which is a simple or subscripted variable. This formal parameter must not be specified by value (it is a call-by-name parameter).

Examples of procedure declarations

```
Procedure Beta (X, Y);
    Value Y;
    Real array X;  Integer Y ;
    Begin Integer A;  Real Sum;
        Sum:=0
        For A:=1 step 1 until Y Do
            Sum:= Sum+X [A] ;
    end;
Procedure Assign (I, A) to:  (B);
    Integer I;  Real A, B;
    Begin Switch S:=L, M, N;
        goto S [I] ;
L: B: = sin(A);
    goto exit;
M: B:= cos(A);
    goto exit;
N: B:= sin(A)/cos(A);
exit; end assign;
```

13 PROCEDURE CALL

A procedure call statement is defined as follows:

$P(AP_1, AP_2, ... , AP_n);$

or

$P(AP_1, AP_2, ... , AP_m)$ comments: $(AP_{m+1}, AP_n);$

AP_1 through AP_n are the actual parameter and P is the procedure name. N must be equal to the number of formal parameters in procedure P. The actual parameters AP_1 through AP_n may be any expression or identifier.

Examples:

```
Beta (AR, if bool then 50 else 100);
assign (2, AR delta+Y, Result);
prop (Z, Z - 7 ) result: (r);
```

14 FUNCTION PROCEDURE DECLARATIONS

A function procedure declaration takes the following form:

type procedure declaration

Type may be real, integer, or boolean. Procedure declaration is defined in section 12.

The function procedure name must appear to the left of an assignment symbol at least once within the procedure body. This is the value that is obtained when activating the procedure.

Example:

```
real procedure PAT (A, B);
real A, B;
PAT:= if A >B then A * B else
    if A = B then A else B;
```

15 FUNCTION PROCEDURE CALL

A function procedure is activated by the appearance of the function name and associated actual parameters in an expression. When the program is executed, the expression is evaluated using the value computed by the function procedure.

Example:

```
X:= Y * PAT (cos(X), Z + F - sin (y))/2;
```

16 EXTERNAL PROCEDURE DECLARATIONS

An external procedure declaration takes the form:

external P_1, P_2, . . . , P_n;

P_1 through P_n are independently compiled procedures.

External function procedures must be declared as to type and take the form:

type external P_1, P_2, . . . , P_n;

P_1 through P_n is defined as above and type is real, integer, or boolean.

Example:

```
external prol, pro2, pro3;
integer external dizzy, weak, sickly;
```

17 STANDARD FUNCTION PROCEDURES

Standard function procedures are defined for 1604 Control Data ALGOL, and require no declaration. They act as if declared in a block dominant to the program. At execution time expressions containing standard functions are evaluated using the value computed by the functions. The argument E may be a variable, a number, a standard function procedure, including itself, or a combination of these. Standard functions are the only procedures which can call themselves.

The following standard function procedures accept real or integer values and supply type real results.

sqrt (E)	square root of E; not defined for x less than 0
sin (E)	sine of E radians
cos (E)	cosine of E radians
arctan (E)	arctan of E radians; principal value between - $^{\pi}/2$ and + $^{\pi}/2$
ln (E)	natural logarithm of E; not defined for E less than or equal to 0
exp (E)	exponential function of E
abs (E)	absolute value of E

The remaining two standard function procedures result in type integer values.

sign (E)

E	sign (E)
E > 0	+1
E = 0	0
E < 0	-1

entier (E) largest integer not exceeding the value of E

Examples of function procedures:

```
abs (z)

B1 + (sqrt(B ↑ 2-4*A, *C)/2A.)

sin(sin(X))

entier (2.5*sign(ABAR))

ACE + arctan(TRFL)
```

18 DATA TRANSMISSION

Six standard procedures provide for data transmission operations.

READ

PRINT

WRITE

PUNCH

INPUT

OUTPUT

READ

ALGOL numbers and boolean values are entered from the standard input unit, 50, into the computer by READ statements. The input values must take the form defined in section 2 except the letter E may be used in place of $_{10}$.

READ ($v_1, v_2, \ldots v_n$);

v_1 through v_n comprise the list of variable identifiers into which the values are to be stored. If a number read differs from the type declaration contained in the program, the number is converted and stored according to the type declaration. The following values may be used for true and false:

T, TRUE, T, TRUE

F, FALSE, F, FALSE

Input numbers may be any length up to 64 characters, but each number, including the last, must be terminated by a comma. Any number of cards may be used to contain the punched input numbers and values. Spaces are ignored and the number field may be continued on a following card. All 80 card columns are interpreted. Arrays may be read by the READ procedure and a for statement.

Example:

```
for I:=1 step 1 until 100 do READ (A[I]);
```

The variable I is incremented by 1, reading 100 numbers from cards in the standard input unit and storing them in the A array. Each number, including the last, must be followed by a comma, to indicate the field widths.

WRITE ('string');

Procedure WRITE has one or more parameters; each is a string or a conditional string expression. The WRITE procedure reproduces strings on standard output unit, 51 for headings and other identification. The strings can be subjected to a conditional clause. A WRITE procedure always begins a printed line with single spacing between WRITE procedure strings. If a string is not completed on a 120 space line, it is continued on subsequent lines. Each string printed begins a new line.

Examples:

```
WRITE ('TABLE');
WRITE (if D=0 then 'TRUE' else 'FALSE');
```

19 DATA TRANSMISSION USING LIST DECLARATIONS

The remaining input/output procedures require lists to specify the transmitted information. The actual list may be used as a parameter, or a list identifier may be used to reference a list declaration. The form of a list declaration is:

list identifier:=$v_1, v_2, \ldots, v_n$;

The elements of the list, v_1 through v_n can be variable identifiers, for clauses, or expressions.

Examples:

```
list RES:=X, A+B;
list SOLUTION:= for I:=1 step 1 until N do A [I];
list ar:= for I:=1 step 1 until N do (A[i], B[i]);
```

A list identifier may appear in a list. The following examples illustrate two methods of transmitting the same data.

Examples:

```
(1)  list M:= for I:=1 step 1 until 10 do (A[I], for J:=1 step 1
              until 20 do B [I, J]);
(2)  list L:= for J:=1, J+1 while J≤20 do B [I, J];
     list M:= for I:=1 step 1 until 10 do (A [I], L);
```

list M in either case transmits the following data:

```
A [1] ,       B [1, 1] ,       B [1, 2] ,....,  B [1, 20]
A [2] ,       B [2, 1] ,       B [2, 2] ,....,  B [2, 20]
  .              .                .                .
  .              .                .                .
  .              .                .                .
A [10],       B [10, 1],....,                   B [10, 20]
```

PRINT

The PRINT procedure produces up to 6 numbers, with 10 decimal places, per line in the following format:

n.nnnnnnnnnnE±nn

n is any integer in the range 0 through 9. The numbers are printed on the standard output unit, right justified in 20-column fields. Print lists containing more than six variables are continued six per line until the list is exhausted. Each use of PRINT begins a new line.

Example:

(1) PRINT (A, B);

(2) PRINT (for I:=1 step 1 until 10 do A[I]);

Example 1 will print two numbers on a line and example 2 will produce one printed line containing six numbers, followed by a line containing four numbers.

PUNCH

The PUNCH procedure produces up to 4 numbers with 10 decimal places pei card (in the same format as PRINT) on standard punch unit, 52, right justified in 20-column fields. The numbers are separated by commas and carc are punched until the list is exhausted. The cards may be used as input for a READ statement. At least one card is produced for every execution of a PUNCH statement.

PUNCH ($v_1, v_2, \ldots, v_n$);

INPUT/OUTPUT

The INPUT procedure enters input numbers and Hollerith information accor ing to format specifications. The forms are:

INPUT (M, F, list);

INPUT (M, F);

The OUTPUT procedure produces output on the specified logical unit numbe according to format specifications. The forms are:

OUTPUT (M, F, list);

OUTPUT (M, F);

M is an expression designating the logical unit number: programmer units 1-49, 56 and 57, 50 for standard input, 51 for standard output. If the expression is not an integral value, it is rounded upward if the fractional part is one half or more; otherwise it is truncated.

F may be one of the following format expressions:

format

format identifier

conditional format expression

variable representing the initial address of a format string

20 FORMAT

Formats must be in parentheses and are enclosed within string quotes.

‘(format)’

Examples:

‘(6E20.10)’

‘(1HO, 9X, 5HTABLE, I3)’

Formats may appear explicitly in an INPUT or OUTPUT statement or may be referenced by an identifier in the declaration part.

format identifier := ‘(format)’;

Formats control conversion by specifying the position of information on an input/output record. Printed output record lines are 120 Hollerith characters in width. The first character, which is interpreted for printer control, does not appear as printed output. Punched cards containing alphanumeric information have a record width of 80 Hollerith characters. See Control Data FORTRAN publications for a complete explanation of conversion types.

Numerical Conversion Types

Ew.d	Floating point with exponent
Fw.d	Floating point without exponent
Iw	Integer
Ow	Octal

Hollerith Conversion Types

Aw	Alphanumeric
Hw	Headings and titles
wX	Spaces

Multiple Record Control

/	Begin new record

Format strings and format identifiers cannot be mixed in a conditional format expression. READ and INPUT procedures both buffer ahead one card image and should not be used together in a program. Input always initiates a new card. READ continues with the unused portion of a card remaining i its buffer.

Examples:

```
INPUT      (4, if B then '(E20.7)' else '(E20.6)', BETA);

INPUT      (6, if B then F1 else F2, X, Y, Z);

INPUT      (11, if B then F1 else '(E20.6)' A);
                This is mixed and not allowed since F1 is a format
                identifier and '(E20.6)' is a format.

OUTPUT     (41, '(6H TABLE)';

OUTPUT     (51, '(1HO, 9X, 10E 10.2)', for I:=1 step 1 until
           100 do A [I] );

OUTPUT     (51, '(6E20.6)', A,B,C,D,E,F);
```

21 INTERMEDIATE TAPE PROCEDURES

Five intermediate tape procedures use magnetic tape for auxiliary storage, programmer units 1-49.

```
BINREAD (M, list);              ENDFILE (M);
BINWRITE (M, list);             REWIND (M);
                   BACKUP (M)
```

BINWRITE (M, list)

Writes one logical record of binary information on the logical tape unit M. The logical record consists of the data defined as list. If the list is omitted, a logical record containing no information is written.

BINREAD (M, list)

Reads one logical record of binary information from logical unit M. If the list is omitted, the tape moves over the record without transmitting information. If a BINREAD list does not require all the values in the record, the additional information is lost and the tape moves to the beginning of the next logical record. If the list is longer than the values on the record, an error diagnostic occurs and the program is terminated.

Example:

```
BINREAD (6, for I:=1 step 1 until 1000 do A[I]);
BINWRITE (X+Y, a, b, c +d/e, for I:=n,n+1 while n≤m↑2 do AR[I]);
```

ENDFILE (M)

Writes an end-of-file mark on the specified logical tape unit M.

REWIND (M)

Rewinds logical tape unit M to the load point.

BACKUP (M)

Backspaces the logical tape unit M one logical record of binary information or one physical record of BCD information.

22 TAPE CHECKING PROCEDURES

The following boolean procedures check the magnetic tape condition.

```
EOF (M)
READERR (M)
WRITERR (M)
```

EOF (M)

Yields the value true if logical unit M encountered an end-of-file condition on the previous read operation, or the previous write operation encountered an end-of-tape condition. Otherwise, the value is false.

Example:

```
if EOF (6) then go to ALARM;
```

READERR (M)

Yields the value true if the previous read operation on logical unit M produced a parity error; otherwise, the value is false. READERR should not be used to test a READ procedure operation, since READ automatically rereads several times in case of errors terminating the program if necessary.

WRITERR (M)

Yields the value true if the previous write operation on logical unit M produced a parity error; otherwise, the value is false.

23 PAGE

Causes the standard output unit, 51, to begin a new page; it uses no parameters.

24 CONTROL CARDS

COOP Monitor control cards listed below govern compilation and execution. They begin with a 7-9 punch in column 1 and terminate with a period.

${}^{7}_{9}$COOP, acc, ld, $s/u_1/u_2$, time, lines, n.

${}^{7}_{9}$ALGO.

${}^{7}_{9}$ALDAP, P_1, P_2, ..., P_n.

${}^{7}_{9}$EXECUTE,t.

${}^{7}_{9}$BINARY,u.

${}^{7}_{9}$FTN,m,n,u or FORTRAN,m,n,a.

Control cards read within a subsystem are not punched in column 1. The subsystem control cards are:

EOP

END

FINIS

LIB name

MCS CARD

The master control system card, COOP, precedes every program run under control of the COOP Monitor system. A 7-9 punch in column one is followed in column two by:

COOP, acc, id, $S/u_1/u_2$, time, lines, r.

acc	account number
id	identification
u_1	load-and-go; 56 is standard
u_2	57, auxiliary memory tape
time	time limit in minutes for job
lines	maximum number of lines printed output

r recovery key for dumps on standard output unit

0 or blank	octal console dump
1	octal console and numbered common
2	octal console, labeled common, and program
3	octal console, numbered and labeled common
4	octal console and all memory
5	octal console and all memory except monitor region

Examples:

```
COOP, 52462, MOOS, S/56/57,4,1000,5.
COOP, A-3021, DSD, S/12/57,5,500,3.
COOP, 6B100,DEC,S/34/57/4,1500,4.
```

Control Data ALGOL 60 programs can be compiled in two different modes using the COOP Monitor system. ALGO which is the standard mode for compilation is simpler, faster, and more suitable for the major number of ALGOL programs. For programs not suited to the ALGO mode, the more general ALDAP mode can be used.

ALGO

This mode is selected by a card with a 7-9 punch in column 1, followed by ALGO. in columns 2-6. The ALGO mode always executes the source program.

ALDAP

This mode is selected by a card with a 7-9 punch in column 1, followed by ALDAP, P_1, ..., P_n, beginning in column 2.

P_1 through P_n are arbitrary length parameters each beginning with one of the following letters:

L, A, P, E, B, S, O, I, R, N

The meaning of each parameter may be modified by adding =n (n being an integer)

Parameter beginning with	action	using = n
L	List source program on unit 51	list source program on unit n $(1< n \leq 49)$
A	List assembly on unit 51	list assembly on unit n $(1< n \leq 49)$
P	Punch binary cards on unit 52	punch binary cards on unit n $(1< n \leq 49)$
E	Prepare load and go tape on unit 56	Prepare load and go tape on unit n $(1 < n \leq 49$, or $n=56)$
B	Punch BCD CODAP cards on unit 52	punch BCD CODAP cards on unit n $(1< n \leq 49)$
S	Change size of assembler symbol table to 2048 words	Change size of symbol table to n if n is greater 1024
O	Assign unit 47 as translator overflow tape (required only for very large programs)	assign unit n as overflow tape $(1 < n \leq 49)$
R	suppress assembler reference table	
N	suppress assembler null listing	
I[†]	use logical unit 50 as input	use logical unit n as input

EXECUTE,m,n,k.

After an ALDAP compilation or after loading an execution only binary deck, the EXECUTE statement transfers control to the object program. A 7-9 punch is contained in column 1, followed by EXECUTE,m,n,k. beginning in column 2.

m time limit in minutes

n logical input unit; the standard scratch unit 56 is assumed if n is zero or blank

k memory map key; a map will not be produced if k is not zero or blank

[†] If this parameter is missing logical 50 is assumed.

BINARY,n,k.

When a BINARY statement is encountered, the relocatable binary deck following it is transmitted from standard input medium, 50, to logical unit n. A memory map will not be produced if the memory map key, k, is not zero or blank. Standard scratch unit 56 is assumed if n is zero or blank.

FTN,m,n,u.

FORTRAN 62, Control Data publication number 506a or FORTRAN 63 number 529.

EOP

EOP is punched in columns 10-14. In the ALGO mode, an EOP card terminates the program. In the ALDAP mode, an EOP card terminates each ALGOL program or procedure compiled separately.

END

Each FORTRAN subprogram and each CODAP subprogram is terminated by END, punched beginning in column 10.

FINIS

FINIS terminates compiling and assembly operations. FINIS begins in column 10.

LIB

A card with LIB in columns 10-12, as well as, an external declaration within the program is required for each non-standard library procedure called in a program compiled in the ALGO mode. The name of a library entry point begins in column 20 of each LIB card. This library routine must be on the master tape.

PROGRAM

A PROGRAM card is optional and may be used for identification. In the ALDAP mode, it names the program entry point. The format is free field; any combination of alphanumeric characters can follow PROGRAM.

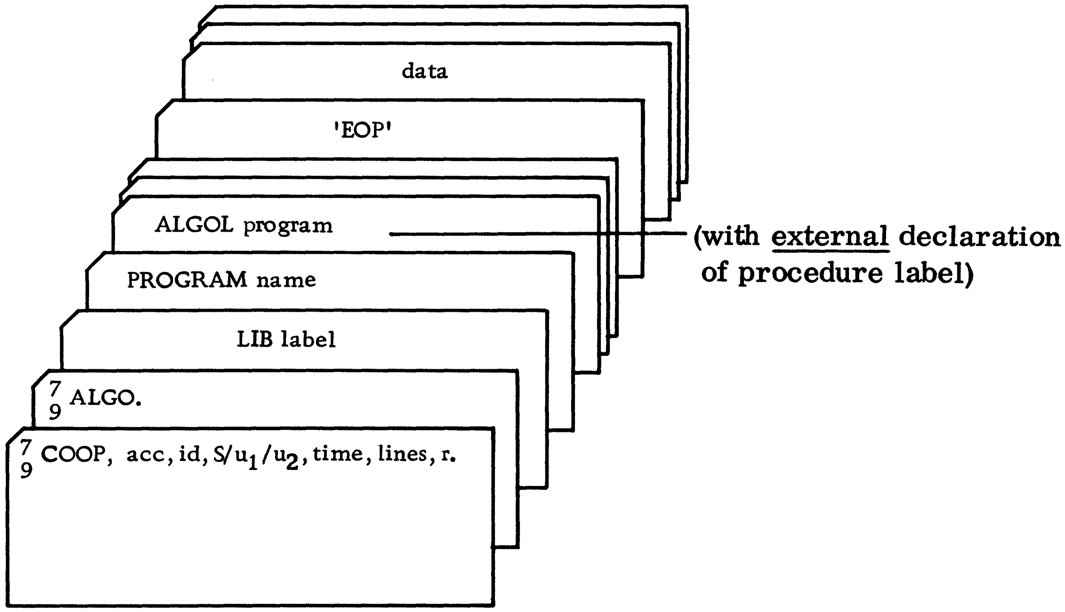
SOURCE DECK

Control Data ALGOL 60 source programs are punched on cards with the hardware representation described in Appendix B. The format is free field from column 1 to 72, except within strings quotes. Columns 73-80 are not interpreted by the compiler, these may be used for sequence numbering and identification.

27 EXAMPLES OF EXECUTION

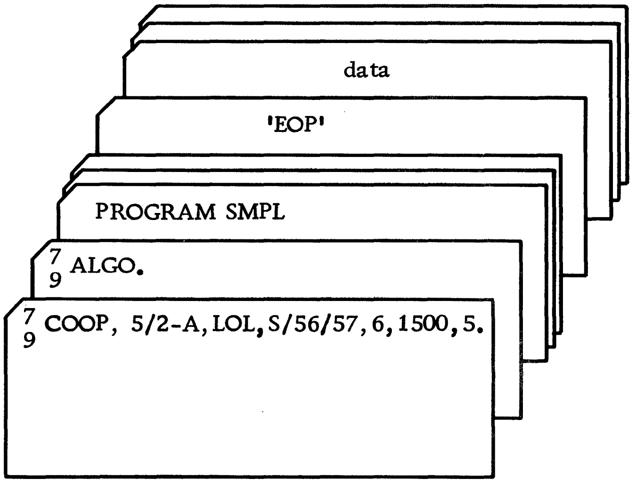
ALGO COMPILATION AND EXECUTION

ALGO provides a simple and fast mode of compiling and executing an ALGOL program. Input cards for the ALGO mode are arranged in the following order. The use of non-standard library procedures and LIB cards is optional.

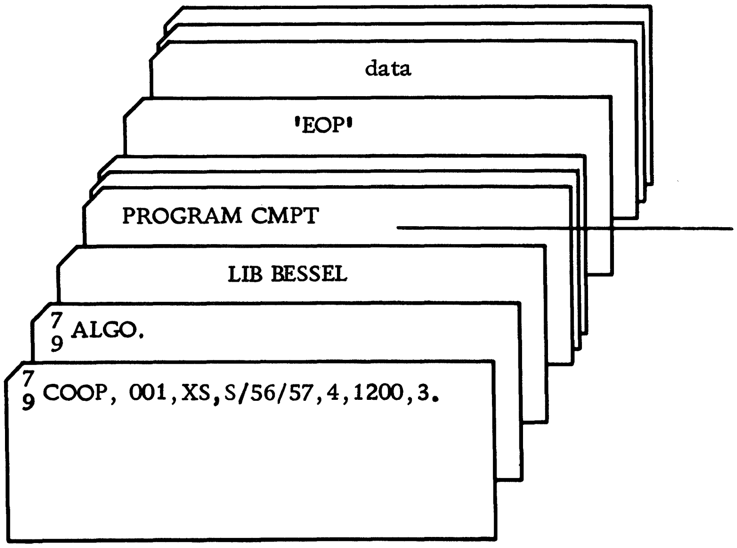

data
'EOP'
ALGOL program
(with external declaration of procedure label)
PROGRAM name
LIB label
7 9 ALGO.
7 9 COOP, acc, id, S/u1/u2, time, lines, r.

Examples:

LOAD and GO



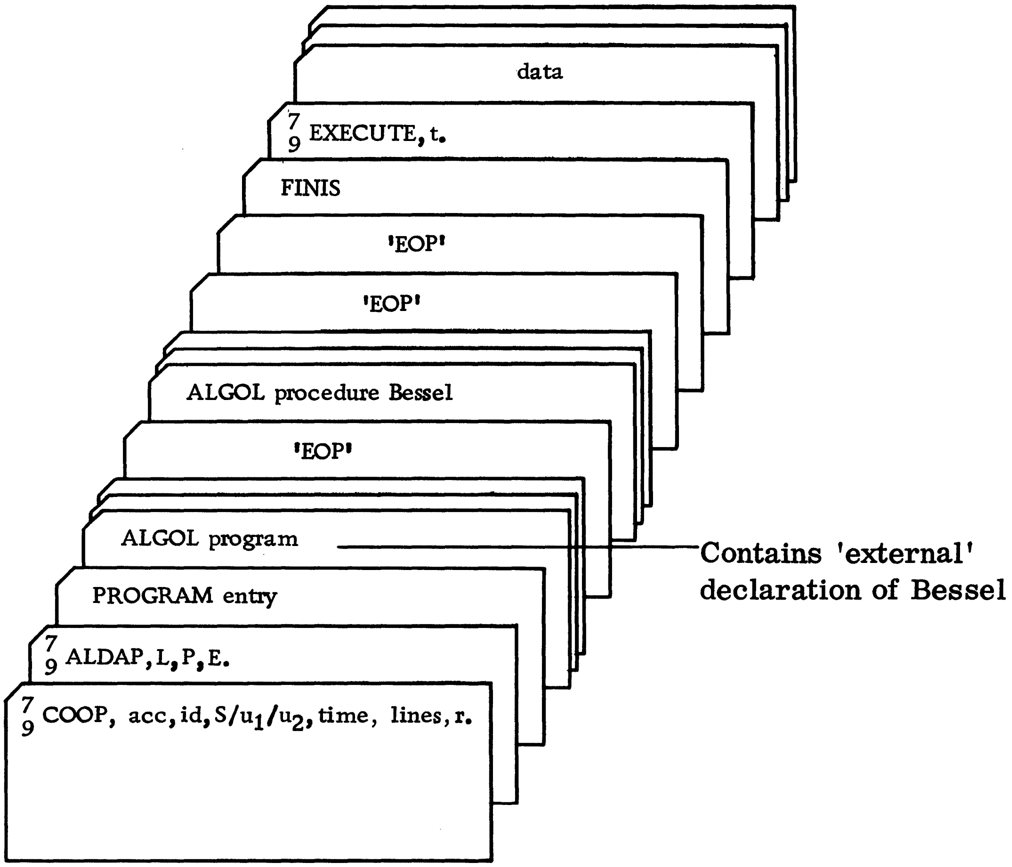
LOAD and GO with Non-Standard Library Procedure Bessel in source deck



contains external declaration of BESSEL

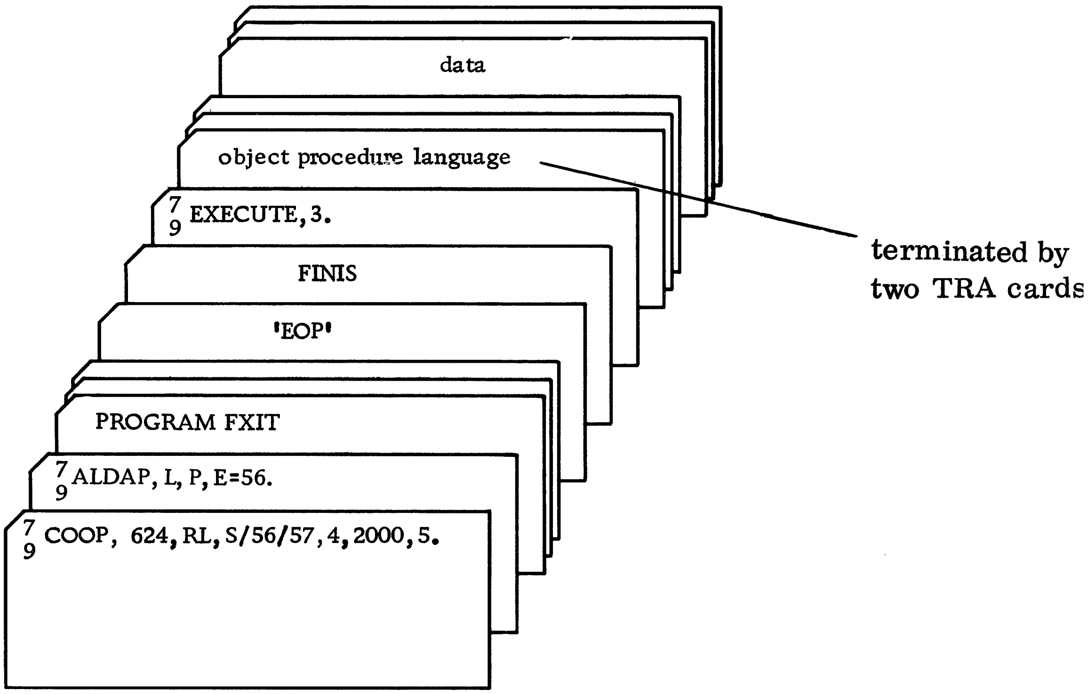
ALDAP COMPILATION AND EXECUTION

ALDAP provides a more varied, slower mode which allows compilation only, compilation and execution, and execution with independently compiled or assembled procedures. Input cards for the ALDAP mode programs are arranged in the following order:



Examples

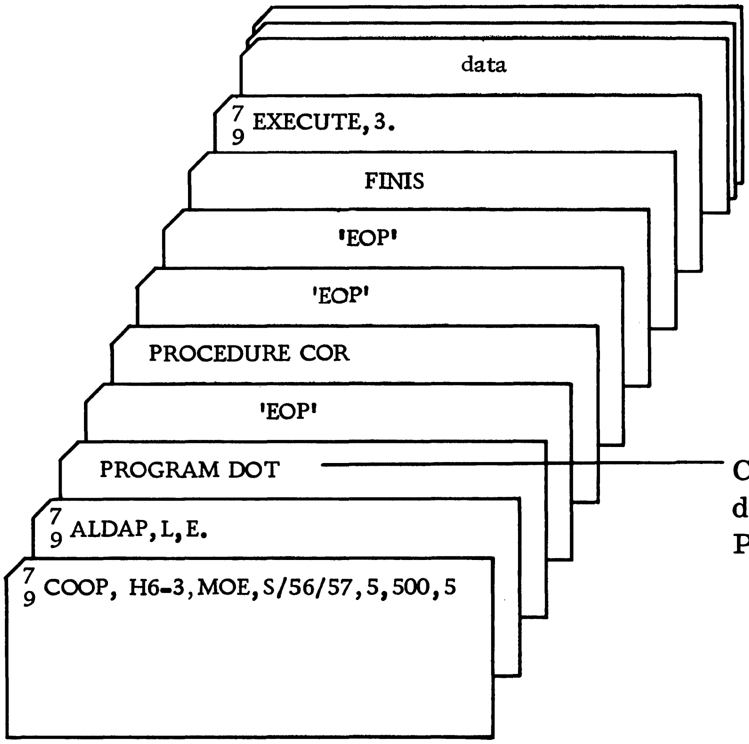
Load and Go with object language procedure



Two TRA cards can be produced during compilation of the procedure by terminating the source language procedure deck with two 'EOP' cards. If a second TRA card is not produced during the procedure compilation, it can be inserted by the programmer for this run.

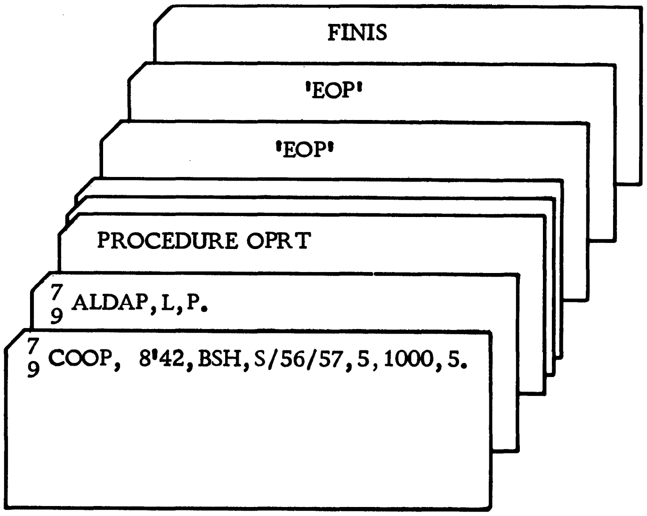
The ALGOL source program FXIT must contain an external declaration of the object language procedure.

Load and Go with PROCEDURE COR



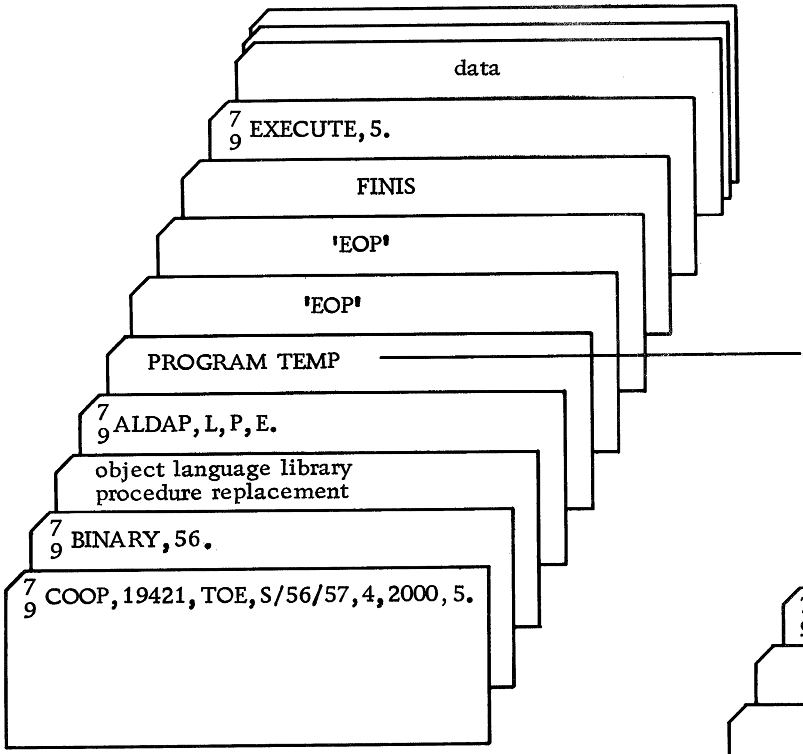
Contains external declaration of PROCEDURE COR

Independent compilation of PROCEDURE OPRT



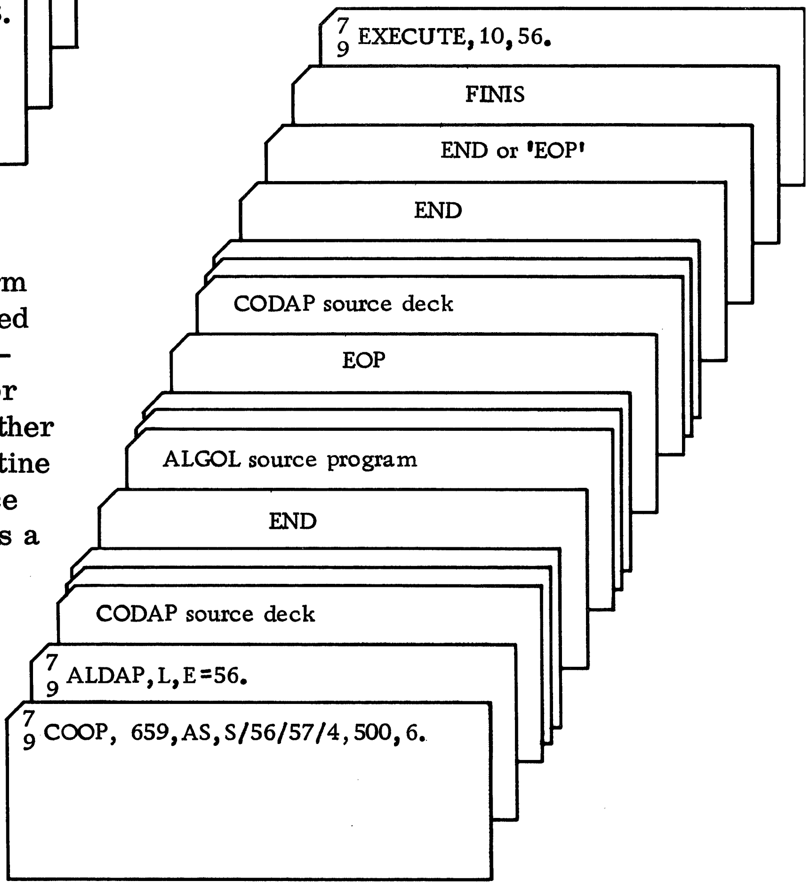
ALDAP

Load and Go, replacing a standard library procedure



Contains external declaration of replaced procedure

CODAP subprograms in the form of subroutines may be assembled with an ALDAP completion containing external declarations for the CODAP subprograms. Neither a CODAP or FORTRAN subroutine may have a transfer card, since the ALGOL program always has a transfer card.



28 FORTRAN SUBPROGRAMS

The procedures FORTRAN, FORTRANF, FTN and FTNF designate FORTRAN compiled subroutines and functions to be executed with an ALGOL program.

```
FORTRAN (name(list) );          FORTRANF (name(list) );

FTN (name(list) );              FTNF (name(list) );
```

Name refers to a FORTRAN subprogram name, and list is the list of actual parameters required by the FORTRAN subprogram (function). The subprogram name must be declared external.

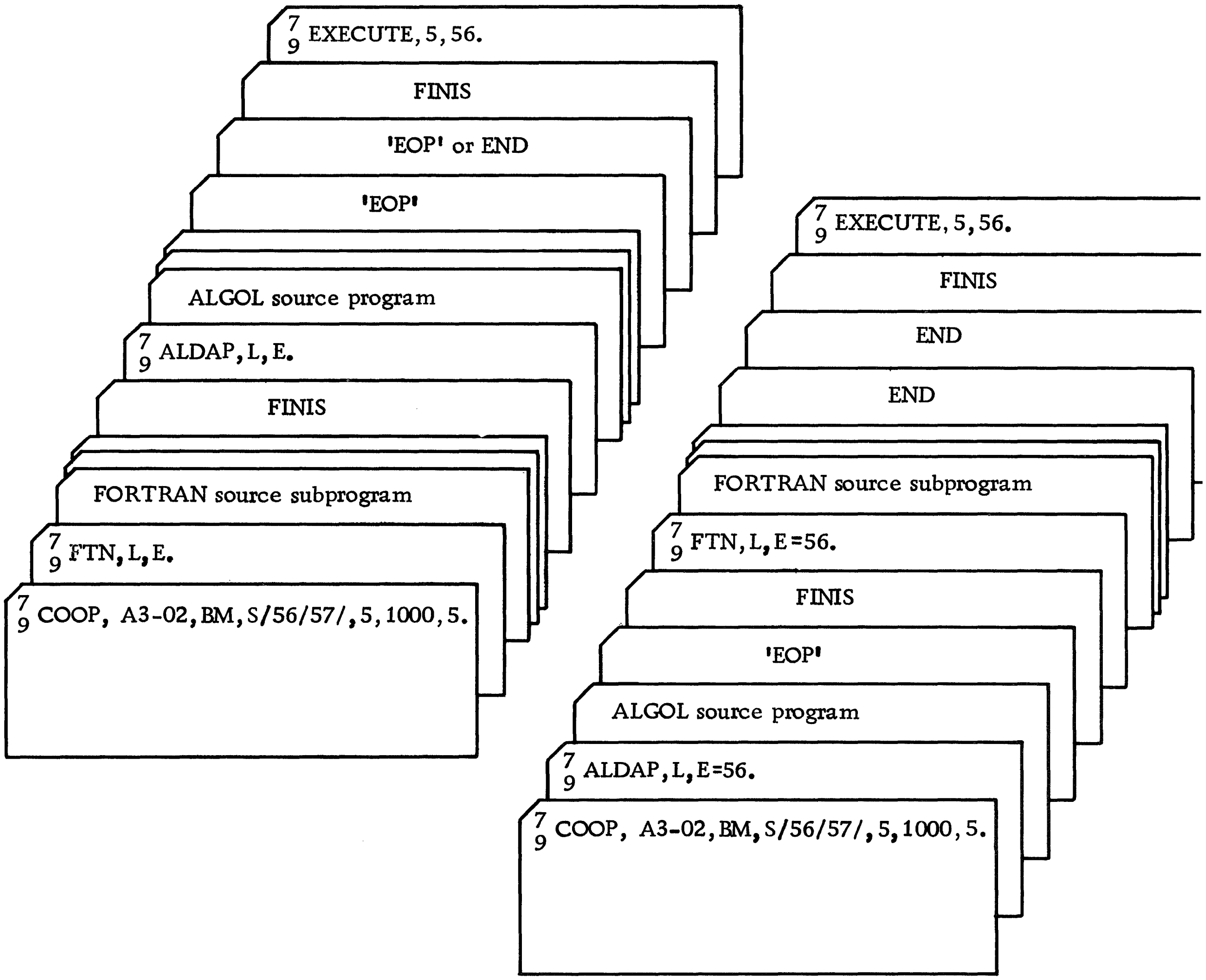
FORTRAN AND FORTRANF procedures refer to FORTRAN-62 subprograms. FTN and FTNF refer to FORTRAN - 63 subprograms.

Labels, switches, and standard functions cannot be used as actual parameters. Array names may not appear as actual parameters, but an actual parameter consisting of an array name with subscripts corresponding to the first array element will satisfy a FORTRAN subroutine which has an array name as a formal parameter. The DIMENSION statement must indicate the size of the ALGOL array.

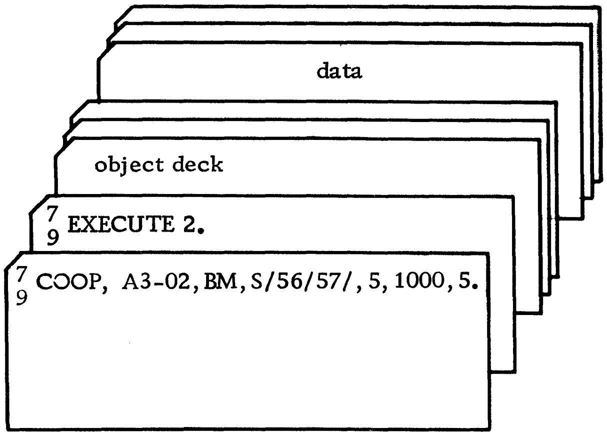
Example of an array used as a parameter to a FORTRAN subroutine:

```
ALGOL program     begin array A[1:10, 6:10];

                  external SUB;
                  .
                  .
                  .
                  FORTRAN (SUB(A[1,6]) );
                  .
                  .
                  .
                  END

FORTRAN           SUBROUTINE SUB(A)
subroutine
                  DIMENSION A(10,5)
                  .
                  .
                  .
                  END
```

ALGOL programs may be compiled before or after FORTRAN programs for the same job.



Execution Only



APPENDIX A

ERROR CHECKING AND DIAGNOSTICS

During the two pass compilation, the following types of errors are detected and listed on the standard output unit.

syntactical errors

undeclared identifiers

identifiers declared more than once in the same block

misspelled delimiters

delimiters which cannot be interpreted by the compiler

missing escape symbols; if both are missing the delimiter is treated as an identifier.

The following errors are not diagnosed:

number of array dimensions used differ from number declared

number of actual parameters differ from number of formal parameters in a procedure call

type errors (simple variable used as subscripted variable)

SYNTACTICAL ERRORS

Syntactical errors are printed with a message to indicate the error and part of the program immediately proceding the error. Example of a misplaced declaration diagnostic:

```
x:=a+b; INTEGER K;

****LAST CHARACTER INDICATES SYNTACTICAL ERROR.

x:=a+b; INTEGER
```

A single syntactical error may cause more than one diagnostic. Syntactical errors corrected by the compiler are noted in messages on the standard output unit. Identifiers are often abbreviated as $C_1, C_2, \ldots C_n$, or ident, in diagnostic messages.

APPENDIX B

The keypunch apostrophe is reserved as the escape symbol which delineates ALGOL delimiters. No distinction is made between upper and lower case letters in the hardware language. The transliteration rules for the symbol delimiters assume a 48-character keypunch and are consistent with the usage in the ALCOR group. In the case of string quotes, the tolerated symbols are required for the inner strings of a nest of strings. Delimiters which are underlined in this manual are punched with escape symbols (begin is 'begin')

ALGOL Symbol	Keypunch	Tolerated Keypunch
<	'LS'	'LESS'
≤	'LQ'	'LSEQ', 'NOT GREATER', 'NOT GREATER'
=	'EQ'	'EQUAL'
≥	'GQ'	'GREQ', 'NOTLESS' 'NOT LESS'
>	'GR'	'GREATER'
≠	'NQ'	'NTEQ', 'NOT EQUAL', 'NOT EQUAL'
¬	'NOT'	
∧	'AND'	
∨	'OR'	
⊃	'IMP'	'IMPLIES', 'IMPL'
≡	'EQV'	'EQUIV'
10	'	'E', 'T'
x	*	
↑	**	'POWER'
÷	//	'DIV'
:	..	
;	$	.,
:=	=	.=, ..=
[	(/	
]	/)	
‘	"	'('
’	"	')'

APPENDIX C

Algol procedures (with n parameters) that are written in CODAP must have the following type format:

```
P       SLJ           0
        LDA     6     0
        STA           P1
        LDA     6     1          Entry
        STA           P2
         .
         .
         .
        LDA     6     n-1
        STA           Pn

        RTJ           P1
+       LDQ           =077777
        STL           T          Picking up the first parameter
        LDA     7     T

        SLJ           p          Exit

P1      BSS           1
        SLJ           P1
P2      BSS           1
        SLJ           P2         Storage location for parameter
         .                       links
         .
         .
Pn      BSS           1
        SLJ           Pn
```

Upon entry to the procedure, index register 6 contains the address of a list of consecutive parameters. These parameters must be stored in the locations designated P1 through Pn in the illustration. All the parameter must be picked up and stored before referencing any of them. The actual parameters take one of the following forms:

```
SLJ    0    ENA    V
SLJ    0    RTJ    GL
```

The parameters are referenced by making a return jump to the appropriate location P1 through Pn in the illustration. Upon return, the accumulator will contain an address of a value, variable, procedure, string, switch or label (depending on the actual parameter). Transferring to a switch element or label outside the procedure must be accomplished by use of the 'goto' interpreter. The call of the interpreter is made with the label address in the accumulator. The call is simply:

```
CALL    AL16GOTO
```

APPENDIX D

The statement:

```
        FORTRAN (name(P1, P2, ...Pn) );
```

or expression:

```
        FORTRANF (name(P1, P2, ... Pn) );
```

Produces the following type subroutine (or function) call

```
             ENA      L(P1)     }  address of P1 to A and
             ENQ      L(P2)     }  address of P2 to Q
             RTJ      Name      }  call of subroutine (function)
        +    ZRO      P3        }
        +    ZRO      P4        }  addresses of P3 through
             .
             .
             .
             ZRO      Pn        }  Pn
        +    RTJ      AL16ERR   }  error return
        +    Normal   return
```

The statement:

```
        FTN (name(P1, P2, ...Pn) );
```

or the expression:

```
        FTNF (name(P1, P2, ...Pn) );
```

produces the following type subroutine (or function) call

```
             RTJ      Name      }  call of subroutine (function)
        +    ZRO      P1        }
             ZRO      P2        }
        +    ZRO      P3        }  addresses of parameters
             .
             .                     P1 through Pn
             .
             ZRO      Pn        }
        +    Normal   return    }
```

CONTROL DATA
CORPORATION

8100 34th AVENUE SOUTH, MINNEAPOLIS 20, MINNESOTA

Litho in