6. CONTROL STATEMENTS

6.1. GENERAL

The compiler translates successive statements in the order in which they appear in the program. The statements are also executed in this same order unless the programmer interrupts this normal sequence with a "transfer of control." Once the transfer has taken place, successive statement sequencing continues from the new point of reference.

Transfer of control in ALGOL is accomplished through use of three kinds of control statements – unconditional, conditional, and iterative.

6.2. UNCONDITIONAL CONTROL STATEMENTS.

The **GO TO** statement causes an unconditional transfer of control to another part of the program.

6.2.1. The **GO TO** Statement

The **GO TO** statement may be written in any one of three ways:

- **GO TO** <designational expression>
- **GOTO** <designational expression>
- **GO** <designational expression>

There are three forms of designational expressions, the label being the simplest.

Example:

```
L:Q = SIN(SQRT(Z)) $
•
•
•
GO TO L $
```

GO TO L interrupts the normal sequence of instructions and restarts at the statement with the label L.

Alternatively the designational expression may take the form of a conditional expression.

Example:

```
GO TO IF X EQL Y THEN L1 ELSE L2 $
```

In this case if X equals Y, control is transferred to the statement labeled L1; otherwise, the transfer is to L2.

A third form of designational expression is a **SWITCH** variable explained below.

6.2.2. The **SWITCH**

The **SWITCH** declaration names a group of alternative points in a program to which control may be transferred. It includes a means for selecting a given designational expression from the **SWITCH** list by means of a subscript expression (evaluated at execution time) with the **SWITCH** identifier. In effect, the **SWITCH** declaration defines a **SWITCH** variable which is similar to a one-dimensional array except that the elements are designational expressions.

To start with, a switch must be described by a **SWITCH** declaration prior to its use as a switch variable. The range of subscripts is from 1 to n, where n is the number of elements in the switch list. If a subscript expression on a switch variable falls outside the defined range of the switch, then the switch operation is ignored.

The general form of the declaration is

SWITCH <switch identifier> = <switch list>

or

SWITCH SWITCH 1 = e_1, e_2, e_3, – – – e_n $

where SWITCH1 is the name of the switch and e_1 – – – e_n are designational expressions.

A switch element is referenced in a **GO TO** statement by means of the switch identifier with the appropriate subscript:

```
GO TO SWITCHN1(I)
```

where I is an arithmetic expression. This expression is evaluated when the **GO TO** is executed. Control is transferred to the statement designated by element I in the switch list of the **SWITCH** declaration (counting from left to right).

To illustrate, assume that it is necessary to transfer to statements labeled L1, L2, L3, and L4 depending on whether the value of J is 1, 2, 3 or 4. This could be accomplished with the following **GO TO** statement:

```
GO TO IF J EQL 1 THEN L1 ELSE
      IF J EQL 2 THEN L2 ELSE
      IF J EQL 3 THEN L3 ELSE
      IF J EQL 4 THEN L4  $
```

However, it is much easier to set up a switch to accomplish the same thing

```
SWITCH S = L1, L2, L3, L4 $
•
•
•
•
GO TO S(J) $
```

Example:

```
SWITCH S = L1, IF X GTR Y THEN L2 ELSE L3, L4, T(I+6), L5 $
```

If the switch variable S is referenced from a **GO TO** statement

```
GO TO S(J),
```

the following transfer of control is made depending upon the value of J:

(1) If J = 1 then control transfers to L1.
(2) If J = 2 then control transfers to either L2 or L3 depending upon X and Y.
(3) If J = 3 then control transfers to L4.
(4) If J = 4 then control transfers to the label which is the value of the (I+6)th designational expression of the switch T.
(5) If J = 5 then control transfers to L5.
(6) If J <1 or J> 5 then no transfer is executed.

6.3. CONDITIONAL CONTROL STATEMENTS

Conditional control statements cause certain statements to be executed or skipped depending on values of Boolean expressions. The **IF** statement provides for executing a statement if, and only if, some relation is true and for skipping over a statement if this relation is false.

The **IF** statement may take the form:

```
IF B1 THEN S1 $ S2 $
```

where B1 is a Boolean expression, S1 is a statement not beginning with **IF**, and S2 is any statement. If B1 is true, then S1 is executed after which control passes to S2. If B1 is false, then S1 is skipped and control continues at S2.

In diagram form:

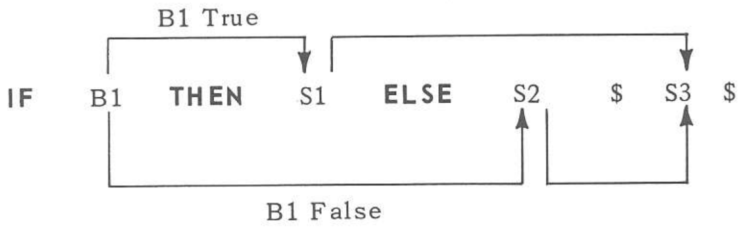
$$\begin{tikzcd}
\text{IF} & \text{B1} \arrow[rr, bend left, "\text{B1 True}"] \arrow[rrrr, bend right, "\text{B1 False}"'] & \text{THEN} & \text{S1} \arrow[rr, bend left] & \$ & \text{S2} & \$
\end{tikzcd}$$

The general form of the **IF** statement is:

```
IF B1 THEN S1 ELSE S2 $ S3 $
```

If B1 is true, statement S1 is executed and statement S2 is skipped; if B1 is false, statement S1 is skipped and S2 is executed. In either case, control continues with statement S3 (except when either S1 or S2 contains a **GO TO** statement).

In diagram form

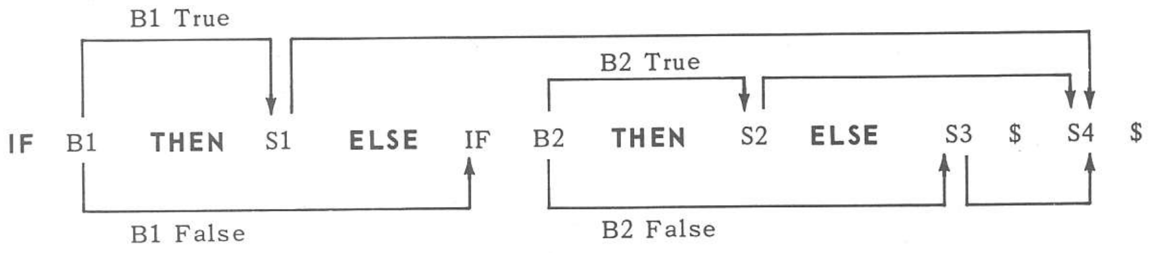


In conditional statements, the statement following **THEN** can not start with **IF**. It may be conditional only if it is enclosed by a **BEGIN – END** pair. There is no restriction on the type of statement following **ELSE**.

Example:

```
IF BOOL THEN BEGIN IF C GEQ -5 THEN GO TO CHECK
  END ELSE V = V+1 $
```

The following example illustrates "nested" conditional statements:



Example:

```
IF DISC LSS 0 THEN GO TO IMAGROOTS
  ELSE IF DISC EQL 0 THEN X1 = X2 = -B/(2*A)
  ELSE BEGIN
         SDISC = SQRT(DISC)  $
           X1  =  (-B+SDISC)/(2*A)  $
           X2  =  (-B - SDISC)/(2*A)
       END $
```

6.4. ITERATIVE CONTROL STATEMENTS – THE **FOR** STATEMENT

The **FOR** statement facilitates programming iterative operations. A part of the program is iterative if it is to be executed repeatedly a specified number of times, if it is to be executed for each one of a designated set of values assigned to a variable, or if it is to be executed repeatedly until some condition is fulfilled. The **FOR** statement handles any of these three conditions.

The general ALGOL **FOR** statement consists of a **<FOR** clause**>** followed by a statement S (simple or compound) where a **< FOR** caluse**>** is:

FOR <variable> = **<FOR** list**> DO**

The **<FOR** list**>** is a sequence of **<FOR** list elements**>** separated by commas. The value of each **<FOR** list element**>** is assigned to the controlled or iteration variable in turn from left to right and the statement S is executed once for each value.

All **FOR** list elements must be of a type compatible with the controlled variable which may be any type of simple or subscripted variable.

There are three possible kinds of **FOR** list elements:

- <arithmetic expression>
- <arithmetic expression> **STEP** <arithmetic expression> **UNTIL** <arithmetic expression>
- <arithmetic expression> **WHILE** <Boolean expression>

6.4.1. Simple List Element

FOR V = <arithmetic expression> **DO** S $

or

FOR V = e_1, e_2, e_3, e_4, – – – e_N **DO** S $

The controlled variable V is successively given the values of the arithmetic expressions, e_1, e_2, e_3, – – – e_N. The statement S is executed once for each value of V.

Example:

```
FOR X = 1.0,1.5,2.5,3.5,7.5 DO S $
```

6.4.2. **STEP – UNTIL** List Element

FOR V = < arithmetic expression > **STEP** < arithmetic expression > **UNTIL** < arithmetic expression > **DO** S $

or

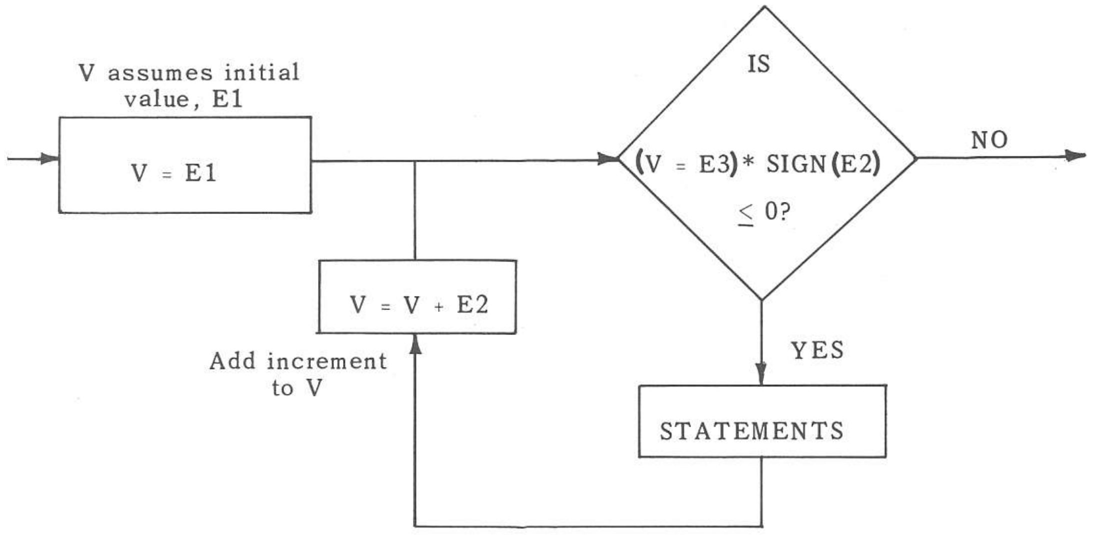
```
FOR V = E1 STEP E2 UNTIL E3 DO S $
```

where E1 is the starting or initial value of V
E2 is the increment by which V is increased algebraically
E3 is the limiting or terminal value of V

The effect of the **FOR** statement is probably best described by the equivalent ALGOL statements:

```
V = E1 $
L1: IF (V-E3)*SIGN(E2) LEQ 0 THEN
BEGIN
S $
V = V + E2 $
GO TO L1
END
```

In all cases if the test fails initially, the statement S is not executed at all. SIGN (X) is a call on a standard function which will return the value 1,0, or –1 depending on whether the value of the argument X is positive, zero, or negative, respectively. This can be shown graphically as follows:



The statement S may redefine V as well as the variables appearing in E2 and E3. Changing E1 will have no effect on the execution of the **FOR** statement as the initial value is assigned to V before S is executed. Extreme care must be taken in assigning values to V within S as this may prevent V from reaching the terminal value.

The more compact form of the **FOR** statement

```
FOR V = (E1,E2,E3)DO S $
```

may be used instead of

```
FOR V = E1 STEP E2 UNTIL E3 DO S $
FOR I = 1 STEP 1 UNTIL N DO S $
FOR I = (1,1,N) DO S $
FOR X = (3.2,.1,9.9) DO S $
```

6.4.3. **WHILE** List

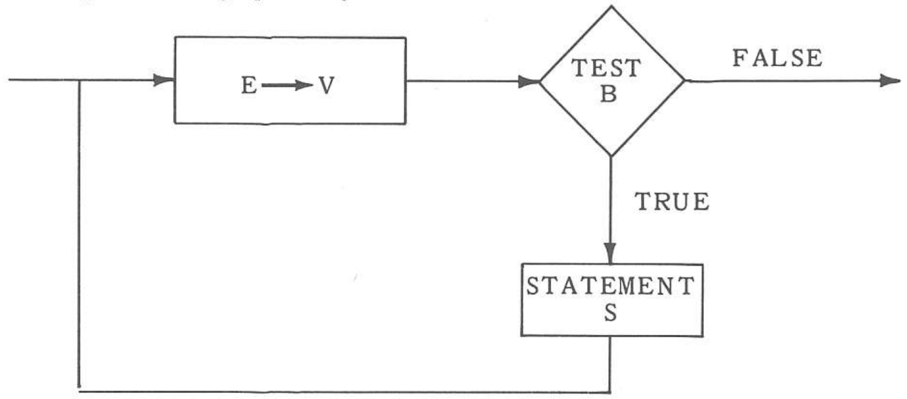
FOR V = < arithmetic expression > **WHILE** < Boolean expression > **DO** S $

or

FOR V = E **WHILE** B **DO** S $

First V is set equal to the arithmetic expression E. If B is true, statement S is executed. After the execution of S, V is replaced by E and again B is tested. If, on the other hand, B is false, then S is skipped and control resumes with the statement following the **FOR** statement.

This can be represented graphically as follows:



The statement S may redefine V or the variables in the expressions E and B.

Example (taken from Problem 2 Appendix D):

```
FOR B = 0.5 * (A/OLDB + OLDB)
          WHILE ABS(B-OLDB) GTR 10**(-6)*B
          DO OLDB = B $
```

In this example the **FOR** statement is executed until B, the square root of A, is accurate to six digits.

The three forms of the list elements may be combined:

```
FOR K = 1,3,5,10 STEP 2 UNTIL 20,50 WHILE B DO S $
```

The statement S will be executed for

K = 1,3,5,10,12,14,16,18,20 and

will then assume the value 50 as long as B is true.

6.4.4. Termination of **FOR** Statements

The following section should be carefully read because it deals with concepts that are not defined in rigorous ALGOL 60. The problem is that a program written in UNIVAC 1108 ALGOL 60 which utilized these concepts would possibly not work on a machine with a different version of ALGOL. The concern here is with the value of the iteration variable in a **FOR** statement when the **FOR** statement is terminated. The ALGOL 60 report leaves this value as undefined when the **FOR** statement is terminated by exhaustion of the **<FOR** list**>** , but in UNIVAC 1108 ALGOL it is well defined, and indeed, very useful. It is because of its usefulness that it is documented here with the warning that it may not work on another machine.

If the statement S has a **GO TO** statement leading out of the **FOR** statement, the value of the iteration variable is the same as it was before the **GO TO** statement was executed. (This is also true in ALGOL 60.) If the exit is made from the **FOR** statement because of the exhaustion of the **<FOR** list**>**, then the value of the variable is that value it held last as may be determined from the equivalent ALGOL statements. For example, to find the first nonblank charact er of a string, either one of two methods could be used.

```
STRING S(120)$
INTEGER I,R $
I=0 $
FOR I=I+1 WHILE (I LSS 121 AND S(I) EQL ' ')
DO $
IF I EQL 121 THEN GO TO STRINGALLBLANK
ELSE FOUNDIT: R=RANK(S(I)) $
```

That method depends on the exhaustion of the **<FOR** list**>** , either because the whole string has been scanned or because a nonblank character has been found. In one case, the final value of I is 121 and in the other it is the index to the non-blank character. Note that a dummy statement **DO** $ follows the **FOR** statement (see 5.7). RANK is a standard function returning the Fieldata equivalent of the first character of the string.

The second method is as follows:

```
FOR I=(1,1,120) DO IF S(I) NEQ ' ' THEN GO TO FOUNDIT $
GO TO STRINGALLBLANK $
FOUNDIT: R=RANK(S(I)) $
```

This method produces the correct value of I because an exit is made from the **FOR** statement by a **GO TO** statement.

A **GO TO** statement from outside a **FOR** statement referring to a label within the **FOR** statement may result in an undefined situation and should thus be avoided.

```
FOR I=(1,1,N) DO
    BEGIN
    .
    .
    L:
    .
    .
    END $
.
.
GO TO L $
```

The above statement, **GO TO** L, is not allowed.

However, it is easy to program the above logic by not using the **FOR** statement.

Example:

```
      I = 0
LOOP: I = I+1 $
      .
      .
   L: .
      .
      .
      . IF I LSS N GO TO LOOP $
      .
      .
      .
      .
   GO TO L $
```