# 5. STATEMENTS

## 5.1. GENERAL

The ALGOL statement is the fundamental unit of operation within the language. The operations to be performed are specified by statements which may be divided into two classes:

- Assignment statements
- Control statements

This section discusses assignment statements, combination of statements, statement labels, and rules for punctuating statements. Section 6 is devoted exclusively to control statements.

## 5.2. COMPOUND STATEMENTS

A number of statements may be grouped to form a compound statement which is to be considered as a single statement. The general form of a compound statement is:

BEGIN S1 $ S2 $ ..... $ SN END

where $S_1$ through $S_n$ are single statements or other compound statements. The words **BEGIN** and **END** serve as opening and closing statement parentheses. Note the absence of $ between $S_n$ and **END**.

## 5.3. ASSIGNMENT STATEMENTS

An assignment statement is of the form:

v = e or
v:= e

Where v is a variable (simple or subscripted), e is an expression, and the equal sign is known as the replacement operator. This replacement operator is not equivalent to the equal sign in mathematical notation. The assignment statement specifies that the expression e is to be evaluated and this value is to replace the current value of the variable v.

Examples:

```
SUM = 0 $
X = Y + Z $
A(I) = X1 $
N = N + 1 $
```

Replace the current value of SUM with zero.

Replace the current value of X with the value of (Y + Z).

Replace the current value of the Ith element of array A with the value of X1.

Increase the value of N by 1.

Note that the last example is not a valid algebraic equation, but it is a valid assignment statement. On the other hand, a valid algebraic equation such as:

```
Z**2 = X**2 + Y**2
```

has no meaning to the compiler as Z**2 is not an identifier. Neither an expression nor a constant may appear to the left of the replacement operator.

In the statement

```
N = IF X EQL Y THEN 1 ELSE 2
```

N is assigned the value 1 or 2 depending on whether X equals Y. In the assignment statement

```
v = e
```

v must be compatible in type with e. The compiler includes transfer functions to transfer from the type of e to the type of v. The available transfer functions are summarized at the end of Appendix B. If v and e are of different types, then the compiler converts e to the type of v before the assignment is made.

If the conversion is from **REAL** to **INTEGER** then the result is rounded to the nearest integer as in the following example:

```
INTEGER I $
I = 1.57 $
```

The assignment statement assigns the value 2 to I. This is equivalent to writing I = ENTIER (1.57 + 0.5) where ENTIER is a standard function which returns the integral part of the argument.

If the expression e is **BOOLEAN**, then v must also be **BOOLEAN**. If e is a string expression then v must be type **STRING** or **INTEGER**; type **INTEGER** applies only if the expression is a numeric string (see Appendix B). If e is an arithmetic expression then the v must be arithmetic. v may be type **STRING** if e is **INTEGER**.

5.3.1. String Assignment Statements

If the variable V in V = S is a string variable, then this is known as a string assignment statement. In this case, the expression S must be either arithmetic or of type **STRING**. If S is an arithmetic expression then it is first converted to type **INTEGER** and then into its associated string.

In all cases, the replacement is made such that the leftmost character of the right-hand side replaces the leftmost character in the left-hand variable. If the string on the left-hand side is longer, extra spaces are supplied to the right as necessary to fill out the left-hand string. If the string on the right-hand side is longer, any excess of characters from the right-hand side is dropped (that is, the replacement is left justified in the left-hand string variable).

As an illustration, consider the following uses in which A is a string variable:

| A before | Statement | A after |
|---|---|---|
| ABCDEF | A = 'XYZUVW' | XYZUVW |
| ABCDEF | A = 'LOOP-DE-LOOP' | LOOP-D |
| ABCDEF | A = 'HOW' | HOW |
| ABCDEF | A(2) = 'Q' | AQCDEF |
| ABCDEF | A(2,3) = 'XYZ' | AXYZEF |
| ABCDEF | A(2,3) = 69 | A69 EF |

It is preferable to write the last statement in the form A(2, 3) = '69Δ' so as to avoid the time consuming integer-to-string conversion. One word of caution, the string replacements are performed a character at a time starting with the leftmost character; hence, a replacement of the form

```
S(2, N-1) = S(1,N-1) $
```

will result in the character in the 1, 1 position, S(1, 1), being propagated down the string (i.e. the first N characters of the string S will all be the same as the character in S(1, 1)).

Characters in a string can be shifted right or left by this type of statement. To shift the characters in a string right, first move the string into another string of the same length and then perform a replacement operation. For example, let S and T be strings of the same length; then the following statements shift S right one position and leave the first character unchanged.

```
T = S
S(2, N-1) = T(1, N-1) $
```

Similarly S can be shifted left by

```
S(1,N-1) = S(2,N-1) $
```

## 5.4. MULTIPLE ASSIGNMENT STATEMENTS

The same value can be assigned to a number of variables by means of a multiple assignment statement. If the variables to which a value is being assigned are mixed in type, then type conversions are performed. Assume X, Y, and E are **REAL**, and I is **INTEGER**. Then the statement

```
X = I = Y = E $
```

evaluates E and assigns this value to Y; the value of Y is then rounded to an integer and assigned to I; the value of I is converted to **REAL** and assigned to X.

The general form of assignment statement is of the form

$$V_1=V_2=V_3.....V_n=E$$

where the V's are variables (simple or subscripted) and E is an expression. If the V's include subscripted variables, then the order of evaluation is as follows:

(1) Any subscript expressions are evaluated in sequence from left to right.

(2) The expression E is evaluated.

(3) The value of the expression is assigned to all variables proceeding from right to left (as in the example X=I=Y=E) with the subscripts having values as determined in 1.

If the value of I is 1 before this statement is encountered,

```
A(I) = B(I+1) = I = I+1 $
```

then evaluation continues as follows (A and B real arrays)

(1) The subscript for A is determined as 1 and for B as 2.

(2) I is incremented by 1 thus it becomes 2.

(3) The integer is converted to REAL and assigned to A(1) and B(2).

Thus A(1)=B(2)=2.0

## 5.5. STATEMENT LABELS

In order to identify a statement a name may be attached to it. This name is called a statement label and permits one statement to refer to another. A label is an identifier – a string of letters and digits beginning with a letter. Numeric labels are not permitted in this implementation of ALGOL. The string may be of any length but, like any identifier they must be unique within the first 12 characters (see 2.2). The label precedes the statement and is separated from it by a ":" (or..). Multiple labels are permitted.

Examples:

```
L1: Y = A*X + B*C $
START: SUMX = 0 $
START:SUMX = 0 $
L1: L2: L3: L4 X=Y $
```

A label is defined by its actual occurrence and is therefore local to the block in which it occurs. Of course, each label must be different from all other identifiers referenced within the block. See Section 8 for further discussion of labels and blocks.

## 5.6. PUNCTUATION

Each statement as well as each declaration must be terminated by a $ or a ;. In a compound statement, a $ preceding the **END** would be redundant and may be omitted (see example in 5.4).

The end of a line has no meaning as punctuation. There is no restriction as to the number of cards that may be used for forming a statement. Spaces must not appear within numbers, labels, or in basic symbols (except for **GO TO** and **REAL 2**). However, spaces must be used to separate adjacent symbols composed of letters or digits. Spaces may be used freely for indentation or to facilitate reading.

## 5.7. DUMMY STATEMENTS

A dummy statement performs no operation. It can be used to place a label.

Example:

```
L1: END
DO $
```