## 3. Textual languages

The textual languages defined in this standard are IL (Instruction List) and ST (Structured Text). The sequential function chart (SFC) elements can be used in conjunction with either of these languages.

### 3.1 *Common elements*

See ‘Common Elements’.

### 3.2 *Language IL (Instruction List)*

see part on IL

### 3.3 *Language ST (Structured Text*

This subclause defines the semantics of the ST (Structured Text) language whose syntax is defined in B.3. In this language, the end of a textual line shall be treated the same as a space (SP) character.

#### *3.3.1 Expressions*

An *expression* is a construct which, when evaluated, yields a value corresponding to one of the data types. Expressions are composed of operators and operands. An *operand* shall be a, a variable, a function invocation, or another expression.
The *operators* of the ST language are summarized in table 55. The evaluation of an expression consists of applying the operators to the operands in a sequence defined by the operator precedence shown in table 55. The operator with highest precedence in an expression shall be applied first, followed by the operator of next lower precedence, etc., until evaluation is complete. Operators of equal precedence shall be applied as written in the expression from left to right. For example, if A, B, C, and D are of type INT with values 1, 2, 3, and 4, respectively, then

A+B-C*ABS(D)

shall evaluate to -9, and

(A+B-C)*ABS(D)

shall evaluate to 0 .
When an operator has two operands, the leftmost operand shall be evaluated first. For example, in the expression

SIN(A)*COS(B)

the expression SIN(A) shall be evaluated first, followed by COS(B), followed by evaluation of the product.
Boolean expressions may be evaluated only to the extent necessary to determine the resultant value. For instance, if A<=B, then only the expression (A>B) would be evaluated to determine that the value of the expression

(A>B) & (C<D)

is Boolean zero.
Functions shall be invoked as elements of expressions consisting of the function name followed by a parenthesized list of arguments.
When an operator in an expression can be represented as one of the overloaded functions, conversion of operands and results shall follow the rule and examples given in the standard at 2.5.1.4.

**TABLE 55 - Operators of the ST language**

| No. | Operation | Symbol | Precedence |
|---|---|---|---|
| 1 | Parenthesization | (expression) | HIGHEST |
| 2 | Function evaluation<br>**Examples**: | identifier (argument list)<br>LN(A), MAX(X,Y), etc. | |
| 3 | Exponentiation (Note 2) | ** | |
| 4 | Negation | - | |
| 5 | Complement | NOT | |
| 6 | Multiply | * | |
| 7 | Divide | / | |
| 8 | Modulo | MOD | |
| 9 | Add | + | |
| 10 | Subtract | - | |
| 11 | Comparison | < , > , <= , >= | |
| 12 | Equality | = | |
| 13 | Inequality | <> | |
| 14 | Boolean AND | & | |
| 15 | Boolean AND | AND | |
| 16 | Boolean Exclusive OR | XOR | |
| 17 | Boolean OR | OR | LOWEST |

NOTES

1 - The same restrictions apply to the operands of these operators as to the inputs of the corresponding functions defined in 2.5.1.5.

2 - The result of evaluating the expression A**B shall be the same as the result of evaluating the function EXPT(A,B) as defined in table 24.

### 3.3.2 *Statements*

The statements of the ST language are summarized in table 56. Statements shall be terminated by semicolons as specified in the syntax of B.3.

**Table 56 - ST language statements**

| No. | Statement type/Reference | Examples |
|---|---|---|
| 1 | Assignment (3.3.2.1) | A := B; CV := CV+1; C := SIN(X); |
| 2 | Function block Invocation and FB output usage (3.3.2.2) | CMD_TMR(IN:=%IX5, PT:=T#300ms) ;<br>A := CMD_TMR.Q ; |
| 3 | RETURN (3.3.2.2) | RETURN ; |
| 4 | IF (3.3.2.3) | D := B*B - 4*A*C ;<br>IF D < 0.0 THEN NROOTS := 0 ;<br>ELSIF D = 0.0 THEN<br>NROOTS := 1 ;<br>X1 := - B/(2.0*A) ;<br>ELSE<br>NROOTS := 2 ;<br>X1 := (- B + SQRT(D))/(2.0*A) ;<br>X2 := (- B - SQRT(D))/(2.0*A) ;<br>END_IF ; |
| 5 | CASE (3.3.2.3) | TW :=<br>BCD_TO_INT(THUMBWHEEL);<br>TW_ERROR := 0;<br>CASE TW OF<br>1,5: DISPLAY := OVEN_TEMP;<br>2: DISPLAY := MOTOR_SPEED;<br>3: DISPLAY := GROSS - TARE;<br>4,6..10: DISPLAY := STATUS(TW - 4);<br>ELSE DISPLAY := 0 ;<br>TW_ERROR := 1;<br>END_CASE;<br>QW100 := INT_TO_BCD(DISPLAY); |
| 6 | FOR (3.3.2.4) | J := 101 ;<br>FOR I := 1 TO 100 BY 2 DO<br>IF WORDS[I] = 'KEY' THEN<br>J := I ;<br>EXIT ;<br>END_IF ;<br>END_FOR ; |
| 7 | WHILE (3.3.2.4) | J := 1;<br>WHILE J <= 100 & WORDS[J] <> 'KEY' DO<br>J := J+2 ;<br>END_WHILE ; |

| | | |
|---|---|---|
| 8 | REPEAT (3.3.2.4) | J := -1 ;<br>REPEAT<br>J := J+2 ;<br>UNTIL J = 101 OR WORDS[J] = 'KEY'<br>END_REPEAT ; |
| 9 | EXIT (3.3.2.4) | EXIT ; |
| 10 | Empty Statement | ; |
| NOTE - If the EXIT statement (9) is supported, then it shall be supported for all of the iteration statements (FOR, WHILE, REPEAT) which are supported in the implementation. | | |

#### 3.3.2.1 *Assignment statements*

The assignment statement replaces the current value of a single or multi-element variable by the result of evaluating an expression. An assignment statement shall consist of a variable reference on the left-hand side, followed by the *assignment operator* ":=", followed by the expression to be evaluated. For instance, the statement

A := B ;

would be used to replace the single data value of variable A by the current value of variable B if both were of type INT. However, if both A and B were of type ANALOG_CHANNEL_CONFIGURATION as described in table 12, then the values of all the elements of the structured variable A would be replaced by the current values of the corresponding elements of variable B.
As illustrated in figure 6, the assignment statement shall also be used to assign the value to be returned by a function, by placing the function name to the left of an assignment operator in the body of the function declaration. The value returned by the function shall be the result of the most recent evaluation of such an assignment. It is an error to return from the evaluation of a function with the " ENO" output non-zero unless at least one such assignment has been made.

#### 3.3.2.2 *Function and function block control statements.*

Function and function block control statements consist of the mechanisms for invoking function blocks and for returning control to the invoking entity before the physical end of a function or function block.
Function evaluation shall be invoked as part of expression evaluation, as specified in 3.3.1.
Function blocks shall be invoked by a statement consisting of the name of the function block followed by a parenthesized list of named input parameter value assignments, as illustrated in table56. The order in which input parameters are listed in a function block invocation shall not be significant. It is not required that all input parameters be assigned values in every invocation of a function block. If a particular parameter is not assigned a value in a function block invocation, the previously assigned value (or the initial value, if no previous assignment has been made) shall apply.
The RETURN statement shall provide early exit from a function , function block or program (e.g., as the result of the evaluation of an IF statement).

#### 3.3.2.3 *Selection statements*

Selection statements include the IF and CASE statements. A selection statement selects one (or a group) of its component statements for execution, based on a specified condition. Examples of selection statements are given in table 56.
The IF statement specifies that a group of statements is to be executed only if the associated Boolean expression evaluates to the value 1 (true). If the condition is false, then either no statement is to be

executed, or the statement group following the ELSE keyword (or the ELSIF keyword if its associated Boolean condition is true) is to be executed.
The CASE statement consists of an expression which shall evaluate to a variable of type INT (the "selector"), and a list of statement groups, each group being labelled by one or more integers or ranges of integer values. It specifies that the first group of statements, one of whose ranges contains the computed value of the selector, shall be executed . If the value of the selector does not occur in a range of any case, the statement sequence following the keyword ELSE (if it occurs in the CASE statement) shall be executed. Otherwise, none of the statement sequences shall be executed.

#### 3.3.2.4 *Iteration statements*

Iteration statements specify that the group of associated statements shall be executed repeatedly. The FOR statement is used if the number of iterations can be determined in advance; otherwise, the WHILE or REPEAT constructs are used.
The EXIT statement shall be used to terminate iterations before the termination condition is satisfied. When the EXIT statement is located within nested iterative constructs, exit shall be from the innermost loop in which the EXIT is located, that is, control shall pass to the next statement after the first loop terminator (END_FOR, END_WHILE, or END_REPEAT) following the EXIT statement. For instance, after executing the statements shown in figure 22, the value of the variable SUM shall be 15 if the value of the Boolean variable FLAG is 0, and 6 if FLAG=1.

```
SUM := 0 ;
FOR I := 1 TO 3 DO
  FOR J := 1 TO 2 DO
    IF FLAG THEN EXIT ; END_IF
    SUM := SUM + J ;
  END_FOR ;
  SUM := SUM + I ;
END_FOR ;
```

**Figure 22 - EXIT statement example**

The FOR statement indicates that a statement sequence shall be repeatedly executed, up to the END_FOR keyword, while a progression of values is assigned to the FOR loop control variable. The control variable, initial value, and final value shall be expressions of the same integer type (SINT, INT, or DINT) and shall not be altered by any of the repeated statements. The FOR statement increments the control variable up or down from an initial value to a final value in increments determined by the value of an expression; this value defaults to 1. The test for the termination condition is made at the beginning of each iteration, so that the statement sequence is not executed if the initial value exceeds the final value. The value of the control variable after completion of the FOR loop is implementation-dependent.
An example of the usage of the FOR statement is given in feature 6 of table 56. In this example, the FOR loop is used to determine the index J of the first occurrence (if any) of the string 'KEY' in the odd-numbered elements of an array of strings WORDS with a subscript range of (1..100). If no occurrence is found, J will have the value 101.
The WHILE statement causes the sequence of statements up to the END_WHILE keyword to be executed repeatedly until the associated Boolean expression is false. If the expression is initially false, then the group of statements is not executed at all. For instance, the FOR...END_FOR example given in table 56 can be rewritten using the WHILE...END_WHILE construction shown in table 56.
The REPEAT statement causes the sequence of statements up to the UNTIL keyword to be executed repeatedly (and at least once) until the associated Boolean condition is true. For instance, the WHILE...END_WHILE example given in table 56 can be rewritten using the REPEAT...END_REPEAT construction shown in table 56.
The WHILE and REPEAT statements shall not be used to achieve interprocess synchronization, for example as a "wait loop with an externally determined termination condition. The SFC elements defined in 2.6 shall be used for this purpose.

It shall be an *error* in the sense of 1.5.1 if a WHILE or REPEAT statement is used in an algorithm for which satisfaction of the loop termination condition or execution of an EXIT statement cannot be guaranteed.