19. STRUCTURED TEXT PROGRAMMING

Topics:
- Basic language structure and syntax
- Variables, functions, values
- Program flow commands and structures
- Function names
- Program Example

Objectives:
- To be able to write functions in Structured Text programs
- To understand the parallels between Ladder Logic and Structured Text
- To understand differences between Allen Bradley and the standard

19.1 INTRODUCTION

If you know how to program in any high level language, such as Basic or C, you will be comfortable with Structured Text (ST) programming. ST programming is part of the IEC 61131 standard. An example program is shown in Figure 261. The program is called main and is defined between the statements PROGRAM and END_PROGRAM. Every program begins with statements the define the variables. In this case the variable i is defined to be an integer. The program follows the variable declarations. This program counts from 0 to 10 with a loop. When the example program starts the value of integer memory i will be set to zero. The REPEAT and END_REPEAT statements define the loop. The UNTIL statement defines when the loop must end. A line is present to increment the value of i for each loop.

PROGRAM main
VAR
  i : INT;
END_VAR
i := 0;
REPEAT
  i := i + 1;
UNTIL i >= 10;
END_REPEAT;
END_PROGRAM

Note: Allen Bradley does not implement the standard so that the programs can be written with text only. When programming in RSLogix, only the section indicated to the left would be entered. The variable 'i' would be defined as a tag, and the program would be defined as a task.

Figure 261  A Structured Text Example Program

One important difference between ST and traditional programming languages is the nature of
program flow control. A ST program will be run from beginning to end many times each second. A traditional program should not reach the end until it is completely finished. In the previous example the loop could lead to a program that (with some modification) might go into an infinite loop. If this were to happen during a control application the controller would stop responding, the process might become dangerous, and the controller watchdog timer would force a fault.

ST has been designed to work with the other PLC programming languages. For example, a ladder logic program can call a structured text subroutine.

19.2 THE LANGUAGE

The language is composed of written statements separated by semicolons. The statements use predefined statements and program subroutines to change variables. The variables can be explicitly defined values, internally stored variables, or inputs and outputs. Spaces can be used to separate statements and variables, although they are not often necessary. Structured text is not case sensitive, but it can be useful to make variables lower case, and make statements upper case. Indenting and comments should also be used to increase readability and documents the program. Consider the example shown in Figure 262.

<table>
<thead>
<tr>
<th>GOOD</th>
<th>BAD</th>
</tr>
</thead>
<tbody>
<tr>
<td>FUNCTION sample</td>
<td></td>
</tr>
<tr>
<td>INPUT_VAR</td>
<td></td>
</tr>
<tr>
<td>start : BOOL; (* a NO start input *)</td>
<td></td>
</tr>
<tr>
<td>stop : BOOL; (* a NC stop input *)</td>
<td></td>
</tr>
<tr>
<td>END_VAR</td>
<td></td>
</tr>
<tr>
<td>OUTPUT_VAR</td>
<td></td>
</tr>
<tr>
<td>motor : BOOL;(* a motor control relay *)</td>
<td></td>
</tr>
<tr>
<td>END_VAR</td>
<td></td>
</tr>
<tr>
<td>motor := (motor + start) * stop;(* get the motor output *)</td>
<td></td>
</tr>
<tr>
<td>END_FUNCTION</td>
<td></td>
</tr>
</tbody>
</table>

```
FUNCTION sample
  INPUT_VAR
    start : BOOL; (* a NO start input *)
    stop : BOOL; (* a NC stop input *)
  END_VAR
  OUTPUT_VAR
    motor : BOOL;(* a motor control relay *)
  END_VAR
  motor := (motor + start) * stop;(* get the motor output *)
END_FUNCTION
```

`Figure 262` A Syntax and Structured Programming Example
19.2.1 Elements of the Language

ST programs allow named variables to be defined. This is similar to the use of symbols when programming in ladder logic. When selecting variable names they must begin with a letter, but after that they can include combinations of letters, numbers, and some symbols such as ‘_’. Variable names are not case sensitive and can include any combination of upper and lower case letters. Variable names must also be the same as other key words in the system as shown in Figure 263. In addition, these variable must not have the same name as predefined functions, or user defined functions.

Invalid variable names: START, DATA, PROJECT, SFC, SFC2, LADDER, I/O, ASCII, CAR, FORCE, PLC2, CONFIG, INC, ALL, YES, NO, STRUCTURED TEXT

Valid memory/variable name examples: TESTER, I, I:000, I:000/00, T4:0, T4:0/DN, T4:0.ACC

Figure 263 Acceptable Variable Names

When defining variables one of the declarations in Figure 264 can be used. These define the scope of the variables. The VAR_INPUT, VAR_OUTPUT and VAR_IN_OUT declarations are used for variables that are passed as arguments to the program or function. The RETAIN declaration is used to retain a variable value, even when the PLC power has been cycled. This is similar to a latch application. As mentioned before these are not used when writing Allen Bradley programs, but they are used when defining tags to be used by the structured programs.

<table>
<thead>
<tr>
<th>Declaration</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAR</td>
<td>the general variable declaration</td>
</tr>
<tr>
<td>VAR_INPUT</td>
<td>defines a variable list for a function</td>
</tr>
<tr>
<td>VAR_OUTPUT</td>
<td>defines output variables from a function</td>
</tr>
<tr>
<td>VAR_IN_OUT</td>
<td>defines variable that are both inputs and outputs from a function</td>
</tr>
<tr>
<td>VAR_EXTERNAL</td>
<td>a global variable</td>
</tr>
<tr>
<td>VAR_GLOBAL</td>
<td>a value will be retained when the power is cycled</td>
</tr>
<tr>
<td>VAR_ACCESS</td>
<td>a value that cannot be changed</td>
</tr>
<tr>
<td>AT</td>
<td>can tie a variable to a specific location in memory (without this variable locations are chosen by the compiler</td>
</tr>
<tr>
<td>END_VAR</td>
<td>marks the end of a variable declaration</td>
</tr>
</tbody>
</table>

Figure 264 Variable Declarations

Examples of variable declarations are given in Figure 265.
Figure 265  Variable Declaration Examples

Basic numbers are shown in Figure 266. Note the underline ‘_’ can be ignored, it can be used to break up long numbers, ie. 10_000 = 10000. These are the literal values discussed for Ladder Logic.

<table>
<thead>
<tr>
<th>number type</th>
<th>examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>integers</td>
<td>-100, 0, 100, 10_000</td>
</tr>
<tr>
<td>real numbers</td>
<td>-100.0, 0.0, 100.0, 10_000.0</td>
</tr>
<tr>
<td>real with exponents</td>
<td>-1.0E-2, -1.0e-2, 0.0e0, 1.0E2</td>
</tr>
<tr>
<td>binary numbers</td>
<td>2#111111111, 2#1111_1111, 2#1111_1101_0110_0101</td>
</tr>
<tr>
<td>octal numbers</td>
<td>8#123, 8#777, 8#14</td>
</tr>
<tr>
<td>hexadecimal numbers</td>
<td>16#FF, 16#ff, 16#9a, 16#01</td>
</tr>
<tr>
<td>boolean</td>
<td>0, FALSE, 1, TRUE</td>
</tr>
</tbody>
</table>

Figure 266  Literal Number Examples

Character strings defined as shown in Figure 267.
Basic time and date values are described in Figure 268 and Figure 269. Although it should be noted that for ControlLogix the GSV function is used to get the values.

<table>
<thead>
<tr>
<th>Time Value</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>25ms</td>
<td>T#25ms, T#25.0ms, TIME#25.0ms, T-25ms, T#25ms</td>
</tr>
<tr>
<td>5.5 hours</td>
<td>TIME#5.3h, T#5.3h, T#5h_30m, T#5h30m</td>
</tr>
<tr>
<td>3 days, 5 hours, 6 min, 36 sec</td>
<td>TIME#3d5h6m36s, T#3d_5h_6m_36s</td>
</tr>
</tbody>
</table>

The math functions available for structured text programs are listed in Figure 270. It is worth noting that these functions match the structure of those available for ladder logic. Other, more advanced, functions are also available - a general rule of thumb is if a function is available in one language, it is often available for others.
Figure 270 Math Functions

Functions for logical comparison are given in Figure 271. These will be used in expressions such as IF-THEN statements.

>               greater than
>=              greater than or equal
=               equal
<=              less than or equal
<               less than
<>              not equal

Figure 271 Comparisons

Boolean algebra functions are available, as shown in Figure 272. The can be applied to bits or integers.

:= assigns a value to a variable
+ addition
- subtraction
/ division
* multiplication
MOD(A,B) modulo - this provides the remainder for an integer divide A/B
SQR(A) square root of A
FRD(A) from BCD to decimal
TOD(A) to BCD from decimal
NEG(A) reverse sign +/-
LN(A) natural logarithm
LOG(A) base 10 logarithm
DEG(A) from radians to degrees
RAD(A) to radians from degrees
SIN(A) sine
COS(A) cosine
TAN(A) tangent
ASN(A) arcsine, inverse sine
ACS(A) arccosine - inverse cosine
ATN(A) arctan - inverse tangent
XPY(A,B) A to the power of B
A**B A to the power of B

>               greater than
>=              greater than or equal
=               equal
<=              less than or equal
<               less than
<>              not equal
AND(A,B) logical and
OR(A,B) logical or
XOR(A,B) exclusive or
NOT(A) logical not
!

logical not (note: not implemented on AB controllers)

*Figure 272 Boolean Functions*

The precedence of operations are listed in Figure 273 from highest to lowest. As normal expressions that are the most deeply nested between brackets will be solved first. (Note: when in doubt use brackets to ensure you get the sequence you expect.)

! - (Note: not available on AB controllers)

() functions
XPY, **
- negation
SQR, TOD, FRD, NOT, NEG, LN, LOG, DEG, RAD, SIN, COS, TAN, ASN, ACS, ATN
*, /, MOD
+, -
>, >=, =, <=, <, <>
AND (for word)
XOR (for word)
OR (for word)
AND (bit)
XOR (bit)
OR (bit)
ladder instructions

*Figure 273 Operator Precedence*

Common language structures include those listed in Figure 274.

IF-THEN-ELSIF-ELSE-END_IF; normal if-then structure
CASE-value:-ELSE-END_CASE; a case switching function
FOR-TO-BY-DO-END_FOR; for-next loop
WHILE-DO-END_WHILE;

*Figure 274 Flow Control Functions*

Special instructions include those shown in Figure 275.
19.2.2 Putting Things Together in a Program

Consider the program in Figure 276 to find the average of five values in a real array ‘f[]’. The FOR loop in the example will loop five times adding the array values. After that the sum is divided to get the average.

```plaintext
avg := 0;
FOR (i := 0 TO 4) DO
    avg := avg + f[i];
END_FOR;
avg := avg / 5;
```

The example in Figure 278 shows the use of an IF statement. The example begins with a timer. These are handled slightly differently in ST programs. In this case if ‘b’ is true the timer will be active, if it is false the timer will reset. The second instruction calls ‘TONR’ to update the timer. (Note: ST programs use the FBD_TIMER type, instead of the TIMER type.) The IF statement works as normal, only one of the three cases will occur with the ELSE defining the default if the other two fail.

```plaintext
avg := 0;
retain();
immediate input update
EXIT;
EMPTY
causes a bit to be retentive
```

Figure 275 Special Instructions
t.TimerEnable := b;
TONR(t);
IF (a = 1) THEN
  x := 1;
ELSIF (b = 1 AND t.DN = 1) THEN
  y := 1;
  IF (I:000/02 = 0) THEN
    z := 1;
  END_IF;
ELSE
  x := 0;
  y := 0;
  z := 0;
END_IF;

CASE test OF
  0:
    a.0 := 1;
  1:
    a.1 := 1;
  2:
    a.2 := 1;
  3:
    a.3 := 1;
ELSE
  a := 0;
END_CASE;

Figure 278  Example With An If Statement

Figure 279 shows the use of a CASE statement to set bits 0 to 3 of ’a’ based upon the value of ’test’. In the event none of the values are matched, ’a’ will be set to zero, turning off all bits.

The example in Figure 280 accepts a BCD input from ’bcd_input’ and uses it to change the delay time for TON delay time. When the input ’test_input’ is true the time will count. When the timer is done ’set’ will become true.
Most of the IEC61131-3 defined functions with arguments are given in Figure 281. Some of the functions can be overloaded, for example ADD could have more than two values to add, and others have optional arguments. In most cases the optional arguments are things like preset values for timers. When arguments are left out they default to values, typically 0. ControlLogix uses many of the standard function names and arguments but does not support the overloading part of the standard.
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABS(A);</td>
<td>absolute value of A</td>
</tr>
<tr>
<td>ACOS(A);</td>
<td>the inverse cosine of A</td>
</tr>
<tr>
<td>ADD(A,B,...);</td>
<td>add A+B+...</td>
</tr>
<tr>
<td>AND(A,B,...);</td>
<td>logical and of inputs A,B,...</td>
</tr>
<tr>
<td>ASIN(A);</td>
<td>the inverse sine of A</td>
</tr>
<tr>
<td>ATAN(A);</td>
<td>the inverse tangent of A</td>
</tr>
<tr>
<td>BCD_TO_INT(A);</td>
<td>converts a BCD to an integer</td>
</tr>
<tr>
<td>CONCAT(A,B,...);</td>
<td>will return strings A,B,... joined together</td>
</tr>
<tr>
<td>COS(A);</td>
<td>finds the cosine of A</td>
</tr>
<tr>
<td>CTD(CD:=A,LD:=B, PV:=C);</td>
<td>down counter active &lt;=0, A decreases, B loads preset</td>
</tr>
<tr>
<td>CTU(CU:=A,R:=B, PV:=C);</td>
<td>up counter active &gt;=C, A decreases, B resets</td>
</tr>
<tr>
<td>CTUD(CU:=A,CD:=B,R:=C,LD:=D,PV:=E);</td>
<td>up/down counter combined functions of the up and down counters</td>
</tr>
<tr>
<td>DELETE(IN:=A, L:=B, P:=C);</td>
<td>will delete B characters at position C in string A</td>
</tr>
<tr>
<td>DIV(A,B);</td>
<td>A/B</td>
</tr>
<tr>
<td>EQ(A,B,C,...);</td>
<td>will compare A=B=C=...</td>
</tr>
<tr>
<td>EXP(A);</td>
<td>finds e**A where e is the natural number</td>
</tr>
<tr>
<td>EXPT(A,B);</td>
<td>A**B</td>
</tr>
<tr>
<td>FIND(IN1:=A, IN2:=B);</td>
<td>will find the start of string B in string A</td>
</tr>
<tr>
<td>F_TRIG(A);</td>
<td>a falling edge trigger</td>
</tr>
<tr>
<td>GE(A,B,C,...);</td>
<td>will compare A&gt;=B, B&gt;=C, C&gt;=...</td>
</tr>
<tr>
<td>GT(A,B,C,...);</td>
<td>will compare A&gt;B, B&gt;C, C&gt;...</td>
</tr>
<tr>
<td>INSERT(IN1:=A, IN2:=B, P:=C);</td>
<td>will insert string B into A at position C</td>
</tr>
<tr>
<td>INT_TO_BCD(A);</td>
<td>converts an integer to BCD</td>
</tr>
<tr>
<td>INT_TO_REAL(A);</td>
<td>converts A from integer to real</td>
</tr>
<tr>
<td>LE(A,B,C,...);</td>
<td>will compare A&lt;=B, B&lt;=C, C&lt;=...</td>
</tr>
<tr>
<td>LEFT(IN:=A, L:=B);</td>
<td>will return the left B characters of string A</td>
</tr>
<tr>
<td>LEN(A);</td>
<td>will return the length of string A</td>
</tr>
<tr>
<td>LIMIT(MN:=A, IN:=B, MX:=C);</td>
<td>checks to see if B&gt;=A and B&lt;=C</td>
</tr>
<tr>
<td>LN(A);</td>
<td>natural log of A</td>
</tr>
<tr>
<td>LOG(A);</td>
<td>base 10 log of A</td>
</tr>
<tr>
<td>LT(A,B,C,...);</td>
<td>will compare A&lt;B, B&lt;C, C&lt;...</td>
</tr>
<tr>
<td>MAX(A,B,...);</td>
<td>outputs the maximum of A,B,...</td>
</tr>
<tr>
<td>MID(IN:=A, L:=B, P:=C);</td>
<td>will return B characters starting at C of string A</td>
</tr>
<tr>
<td>MIN(A,B,...);</td>
<td>outputs the minimum of A,B,...</td>
</tr>
<tr>
<td>MOD(A,B);</td>
<td>the remainder or fractional part of A/B</td>
</tr>
<tr>
<td>MOVE(A);</td>
<td>outputs the input, the same as :=</td>
</tr>
<tr>
<td>MUL(A,B,...);</td>
<td>multiply values A<em>B</em>....</td>
</tr>
<tr>
<td>MUX(A,B,C,...);</td>
<td>the value of A will select output B,C,...</td>
</tr>
<tr>
<td>NE(A,B);</td>
<td>will compare A &lt;&gt; B</td>
</tr>
<tr>
<td>NOT(A);</td>
<td>logical not of A</td>
</tr>
<tr>
<td>OR(A,B,...);</td>
<td>logical or of inputs A,B,...</td>
</tr>
</tbody>
</table>
Control programs can become very large. When written in a single program these become confusing, and hard to write/debug. The best way to avoid the endless main program is to use subroutines to divide the main program. The IEC61131 standard allows the definition of subroutines/functions as shown in Figure 282. The function will accept up to three inputs and perform a simple calculation. It then returns one value. As mentioned before ControlLogix does not support overloading, so the function would not be able to have a variable size argument list.

**Figure 281** Structured Text Functions

Control programs can become very large. When written in a single program these become confusing, and hard to write/debug. The best way to avoid the endless main program is to use subroutines to divide the main program. The IEC61131 standard allows the definition of subroutines/functions as shown in Figure 282. The function will accept up to three inputs and perform a simple calculation. It then returns one value. As mentioned before ControlLogix does not support overloading, so the function would not be able to have a variable size argument list.

```plaintext
D := TEST(1.3, 3.4); (* sample calling program, here C will default to 3.14 *)
E := TEST(1.3, 3.4, 6.28); (* here C will be given a new value *)

FUNCTION TEST : REAL
  VAR_INPUT A, B : REAL; C : REAL := 3.14159; END_VAR
  TEST := (A + B) / C;
END_FUNCTION
```

**Figure 282** Declaration of a Function
19.3 AN EXAMPLE

The example beginning in Figure 284 shows a subroutine implementing traffic lights in ST for the ControlLogix processor. The variable 'state' is used to keep track of the current state of the lights. Timer enable bits are used to determine which transition should be checked. Finally the value of 'state' is used to set the outputs. (Note: this is possible because '=' and ':=' are not the same.) This subroutine would be stored under a name such as 'TrafficLights'. It would then be called from the main program as shown in Figure 283.

Figure 283  The Main Traffic Light Program
SBR();

IF S:FS THEN
  state := 0;
  green_EW.TimerEnable := 1;
  yellow_EW.TimerEnable := 0;
  green_NS.TimerEnable := 0;
  yellow_NS.TimerEnable := 0;
END_IF;

TONR(green_EW); TONR(yellow_EW);
TONR(green_NS); TONR(yellow_NS);

CASE state OF
  0: IF green_EW.DN THEN
    state :=1;
    green_EW.TimerEnable := 0;
    yellow_EW.TimerEnable := 1;
  END_IF
  1: IF yellow_EW.DN THEN
    state :=2;
    yellow_EW.TimerEnable := 0;
    green_NS.TimerEnable := 1;
  END_IF
  2: IF green_NS.DN THEN
    state :=3;
    green_NS.TimerEnable := 0;
    yellow_NS.TimerEnable := 1;
  END_IF
  3: IF yellow_NS.DN THEN
    state :=0;
    yellow_NS.TimerEnable := 0;
    green_EW.TimerEnable := 1;
  END_IF

light_EW_green := (state = 0);
light_EW_yellow := (state = 1);
light_EW_red := (state = 2) OR (state = 3);
light_NS_green := (state = 2);
light_NS_yellow := (state = 3);
light_NS_red := (state = 0) OR (state = 1);

RET();

Note: This example is for the AB ControlLogix platform, so it does not show the normal function and tag definitions. These are done separately in the tag editor.

state : DINT
green_EW : FBD_TIMER
yellow_EW : FBD_TIMER
green_NS : FBD_TIMER
yellow_NS : FBD_TIMER
light_EW_green : BOOL alias = rack:1:O.Data.0
light_EW_yellow : BOOL alias = rack:1:O.Data.1
light_EW_red : BOOL alias = rack:1:O.Data.2
light_NS_green : BOOL alias = rack:1:O.Data.3
light_NS_yellow : BOOL alias = rack:1:O.Data.4
light_NS_red : BOOL alias = rack:1:O.Data.5

Figure 284   Traffic Light Subroutine
19.4 SUMMARY

- Structured text programming variables, functions, syntax were discussed.
- The differences between the standard and the Allen Bradley implementation were indicated as appropriate.
- A traffic light example was used to illustrate a ControlLogix application

19.5 PRACTICE PROBLEMS

1. Write a structured text program that will replace the following ladder logic.

2. Implement the following Boolean equations in a Structured Text program. If the program was for a state machine what changes would be required to make it work?

\[
\text{light} = (\text{light} + \text{dark} \cdot \text{switch}) \cdot \text{switch} \cdot \text{light}
\]

\[
\text{dark} = (\text{dark} + \text{light} \cdot \text{switch}) \cdot \text{switch} \cdot \text{dark}
\]
3. Convert the following state diagram to a Structured Text program.

```
FS
X
A
Y
B
D
Z

C
```

4. A temperature value is stored in F8:0. When it rises above 40 the following sequence should occur once. Write a ladder logic program that implement this function with a Structured Text program.

```
horn
2 5 11 15
```

5. Write a structured text program that will replace the following ladder logic.

```
EQU
Source A 20
Source B n

MOV
Source -2
Dest n

LEQ
Source A n
Source B 20

ADD
Source A n
Source B 2
Dest n

LOG
Source n
Dest x

SUB
Source A x
Source B 1
Dest x

MUL
Source A x
Source B 2
Dest x
```
19.6 ASSIGNMENT PROBLEMS

1. Write logic for a traffic light controller using structured text.

2. Write a structured text program to control a press that has an advance and retract with limit switches. The press is started and stopped with start and stop buttons.

3. Write a structured text program to sort a set of ten integer numbers and then find the median value.