Data Science and Automation

Lesson 21

PLC – Structured Text Language

Introduction

Structured Text is the highest level language described by the IEC 61131 norm.

Its syntax is similar to that of Fortran and Pascal.

Each PLC producer personalizes the language, creating a custom «dialect» (that is not compliant with other environment or other PLC produced by different brands).

We will see the «dialect» used in Automation Studio.

Basic Structure

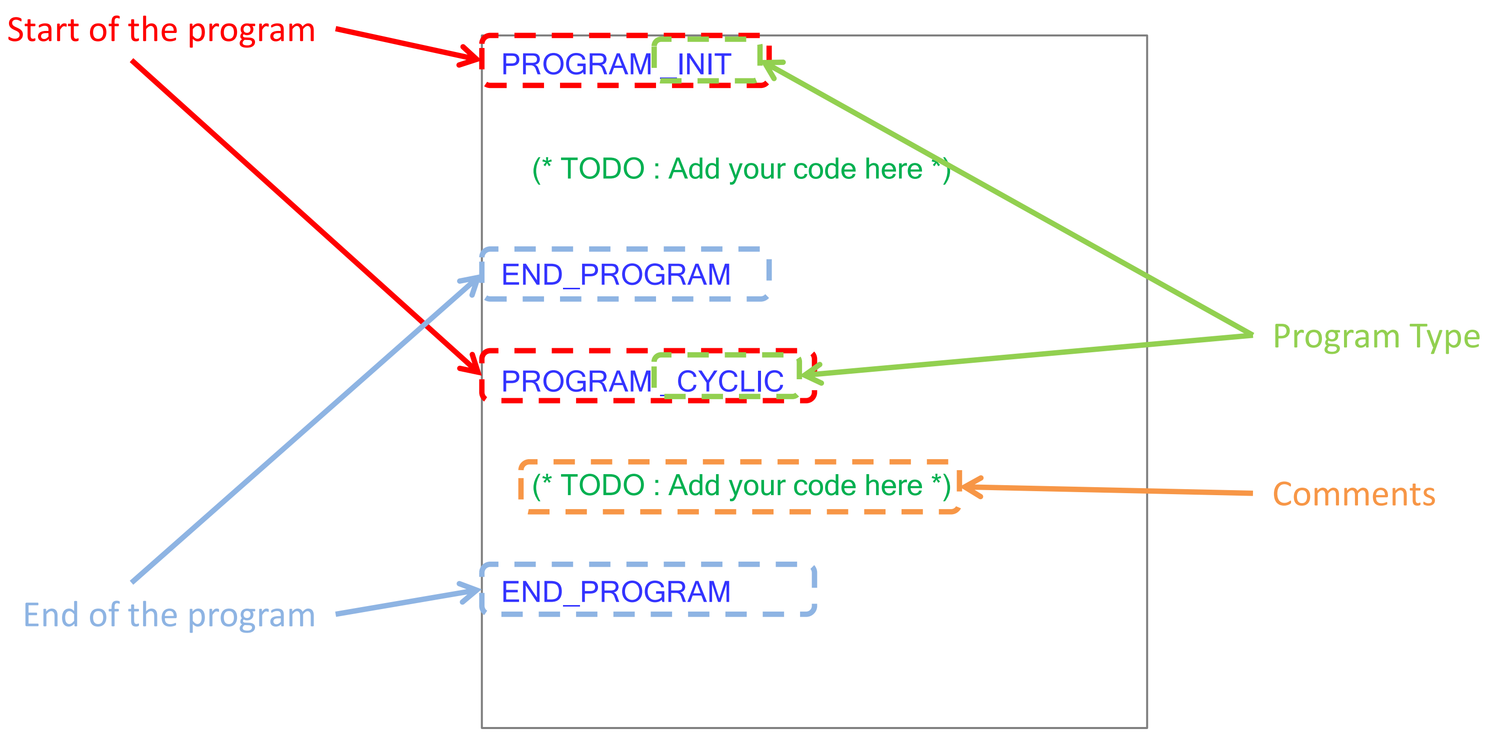
Each structured text file consists of three different programs:

- Init program
- Cyclic program
- Exit program

N.B.1: Init and Exit program are not mandatory.

N.B.2: Automation Studio allows to use these three kind of programs also with the other languages of the IEC 61131 norm.

Basic Structure



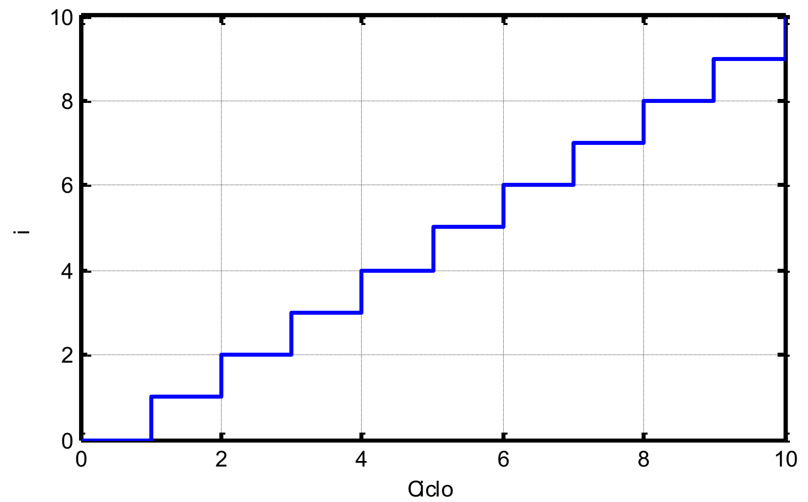
Basic Structure

N.B.: the PROGRAM_INIT is executed when the PLC is turned on, the PROGRAM_CYCLIC is executed cyclically depending on the task schedule of the PLC. This means that all the instructions in the PROGRAM_CYCLIC are executed for every cycle!

Example

```
PROGRAM_INIT
   i := 0;
END_PROGRAM

PROGRAM _CYCLIC
   i := i + 1;
   IF i>=100 THEN
      i := 0;
   END_IF;
END_PROGRAM
```



Keywords

Keyword	Meaning
ACCESS	Access to a dynamic variable (pointer)
BIT_CLR	A = BIT_CLR(IN, POS) A contains the value of IN with the in position POS set to 0
BIT_SET	A = BIT_SET(IN, POS) A contains the value of IN with the in position POS set to 1
BIT_TST	Returns the value of a single bit: A := BIT_TST(IN, POS) A contains the bit at position POS of IN
EDGE	Identify all the edges of the input
EDGENEG	Identify all the negative edges of the input
EDGEPOS	Identify all the positive edges of the input

Operators

Operator	Meaning
ABS	Absolute value
ACOS	Inverse cosine
ADR	Address of the variable
AND	AND bit per bit
ASIN	Inverse sine
ASR	Shift to the right of N bit: A := ASR (IN, N); A contains IN shifted of N bit to the right. The left part is filled with the sign bit
ATAN	Inverse tangent
COS	Cosine
EXP	Exponential

Operators

Operator	Meaning
EXPT	Exponentiation: A := EXPT (IN1, IN2); $A=IN1^{IN2}$
LIMIT	Limit the value of a variable: A = LIMIT (MIN, IN, MAX); MIN is the lower limit, MAX is the upper limit. If IN is less than MIN, the operator returns MIN. If IN is greater than MAX, the operator returns MAX. Otherwise, IN is returned.
LN	Natural logarithm
LOG	Base-10 logarithm
MAX	Maximum between two numbers
MIN	Minimum between two numbers
MOD	Remainder after division between USINT, SINT, INT, UINT, UDINT, DINT variables
MOVE	Assignment; "A := B;" is equals to "A := MOVE (B);"

Operators

Operator	Meaning
MUX	Selection: A = MUX (CHOICE, IN1, IN2, ... INX) CHOICE defines which of the variables IN1, IN2, ... INX has to be assigned to A
NOT	Not bit per bit
OR	Or bit per bit
ROL	Rotation bit per bit to the left: A := ROL (IN, N); IN is shifted N times to the left bit per bit, the leftmost bit is moved to the right
ROR	Rotation bit per bit to the right: A := ROR (IN, N); IN is shifted N times to the right bit per bit, the rightmost bit is moved to the left
SEL	Binary selection: A := SEL (CHOICE, IN1, IN2); CHOICE has to be a BOOL variable. If CHOICE is FALSE, IN1 is returned. Otherwise, IN2 is returned.

Operators

Operator	**Meaning**
SHL	Shift bit per bit to the left: A := SHL (IN, N); IN is shifted of N bit to the left, the right part is filled with zeros
SHR	Shift bit per bit to the right: A := SHR (IN, N); IN is shifted of N bit to the right, the left part is filled with zeros
SIN	Sine
SIZEOF	Returns the number of bytes of the variable (or of the type)
SQRT	Square root
TAN	Tangent
TRUNC	Returns only the integer part of a number
XOR	XOR bit per bit

Control Flow Instructions

IF – THEN – ELSE

ELSIF,
not ELSE IF!!!

```
IF <expression1> THEN
    <instruction_list1>
ELSIF <expression2> THEN
    < instruction_list2>
    .
    .
ELSE
    < instruction_listN>
END_IF;
```

Control Flow Instructions

IF – THEN – ELSE

```
PROGRAM _INIT
   t := 0;
   Out := 8;
END_PROGRAM

PROGRAM _CYCLIC
   IF t < 4 THEN
      t := t +1;
   END_IF;
   IF t < 2 THEN
      Out := 0;
   ELSIF t < 2 THEN
      Out := 1;
   ELSIF t > 3 THEN
      Out := 2;
   ELSE
      Out := 3;
END_IF;
END_PROGRAM
```

Results:

Execution cycle	**Out**
0	8
1	0
2	3
3	3
4	2

Control Flow Instructions

CASE

```
CASE <expression> OF
    <value1> : <instruction_list1>
    <value2>, <value3> : <instruction_list2>
    <value4>..<value5> : <instruction_list3>
    ELSE <instruction_list4>
END_CASE;
```

More values

Range of values

Control Flow Instructions

CASE

```
PROGRAM _INIT
    t := 0;
    Out := 8;
END_PROGRAM

PROGRAM _CYCLIC
    IF t < 4 THEN
        t := t +1;
    END_IF;
    CASE t OF
        1 : Out := 6;
        0, 2 : Out := 1;
        3..4 : Out := 2;
    ELSE Out := 4;
    END_CASE;
END_PROGRAM
```

Results:

Execution cycle	**Out**
0	8
1	6
2	1
3	2
4	2

Control Flow Instructions

FOR

Initial value

Increment per cycle

```
FOR <variable> := <expression1> TO <expression2> BY <expression3> DO
    <lista_istruzioni>
END_FOR;
```

Final value (included)

N.B.: If the increment sets <expression3> to a value greater than <expression2> the cycle is halted.

Control Flow Instructions

FOR

```
PROGRAM _INIT
   A := 0;
   U := 0;
END_PROGRAM

PROGRAM _CYCLIC
   U := A;
   FOR B := A-1 TO 1 BY -1 DO
      U := U*B;
   END_FOR;
END_PROGRAM
```

What does this code compute?

The software computes the factorial of the number A and assigns the result to the variable U

Control Flow Instructions

REPEAT (cycle with the check condition at the end)

```
REPEAT
   <instruction_list>
UNTIL
   <expression>
END_REPEAT;
```

Attention!!!
If <expression> is false the repeat is re-executed, otherwise the execution of the cycle is stopped.

Control Flow Instructions

REPEAT (cycle with the check condition at the end)

```
PROGRAM _INIT
   A := 0;
   B := 0;
   C := 0;
   FOR i:=0 TO 7 BY 1 DO
      U[i] := 0;
   END_FOR;
END_PROGRAM

PROGRAM _CYCLIC
   B := A;
   C := 8;
   FOR i:=0 TO 7 BY 1 DO
      U[i] := 0;
   END_FOR;
   REPEAT
      C := C - 1;
      U[C] := (B/REAL_TO_USINT(EXPT(2,C)))>0;
      B := B - U[C]*REAL_TO_USINT(EXPT(2,C));
   UNTIL C<=0
   END_REPEAT;
END_PROGRAM
```

What does this code compute?

The software computes binary value of the variable A and assigns the result to the vector U

Control Flow Instructions

WHILE (loop with the check condition at the begin)

```
WHILE <expression> DO
   <instruction_list>
END_WHILE;
```

Attention!!!
If <expression> is true,
the loop is executed,
otherwise it stops.

Additional instructions

EXIT: it is like the «break» in the C language. It halts the execution of the loop in which it is inserted.

RETURN: it is like the “return” in the C language. It halts the execution of the function, of the function block or of the program in which it is inserted.

Additional instructions

EXIT

```
PROGRAM _CYCLIC
   U := 0;
   FOR A:=0 TO 1 BY 1 DO
      FOR B:=0 TO 10 BY 1 DO
         IF B>=5 THEN
            EXIT;
         ELSE
            U := U + 1;
         END_IF;
      END_FOR
      U := U + 1;
   END_FOR
END_PROGRAM
```

What is the value of U at the end of the execution?

Let's count the loops:

A=0, B=0 U = 1

…

A=0, B=5 U = 6

A=0, B=5 U = 7 ← Instruction outside the first for loop

A=1, B=0 U = 8

…

A=1, B=5 U = 13

A=1, B=5 U = 14 ← Instruction outside the first for loop

U is equal to 14

Function Block

A Function Block is a piece of control code, that defines some outwards interface variables (input, internal and output).

A Function Block consists of a .st file that contains the software written by structured text.

N.B.: Each program contains also a file .fun which gather all the interfaces of the blocks, of the functions, etc.

Function Block

Example

Write a function block that computes $U = A \cap \bar{B}$

```
(* File ExampleFB.st *)
FUNCTION_BLOCK ExampleFBD
    U := A AND NOT B;
END_FUNCTION_BLOCK;
```

```
(* File Example.st *)
PROGRAM _INIT
END_PROGRAM

PROGRAM _CYCLIC
    ExampleFB(A := In1, B := In2);
    Out := ExampleFB.U;
END_PROGRAM
```

N.B.: Remind to declare, into the variables of the program Example.st, the variabile ExampleFB with type ExampleFBD too!!

Exercices

Exercise 1

We want to create a system that allows the transportation of stones with a cart. The operator starts the system by pressing the button START. The cart follows the rail from left to right and stops itself, waiting the loading of the stones.

When the stones are accumulated into a tank, they are transferred into the cart.

After that, the cart has to move automatically down the rail from right to left.

Exercise 1

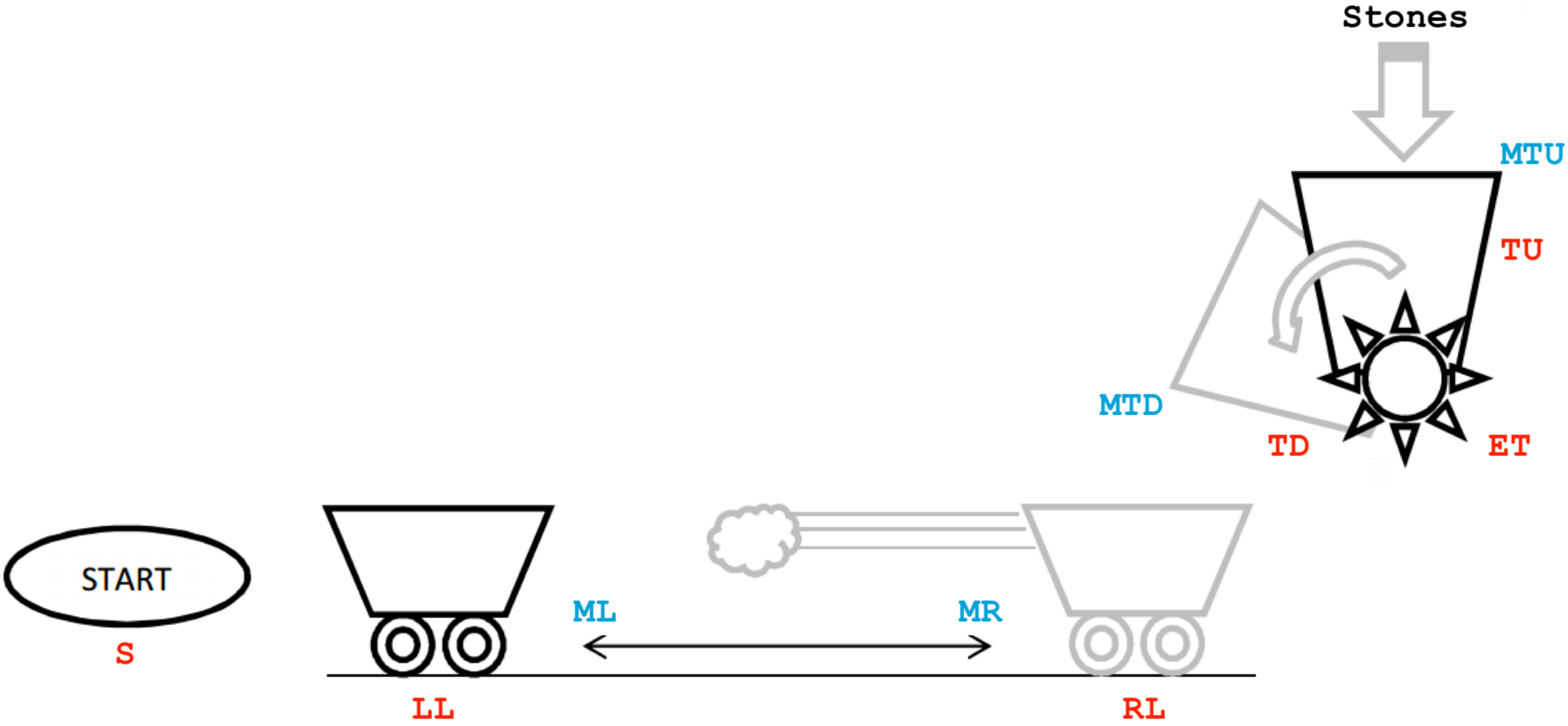
We have six sensors as INPUTS:

- S : Start
- LL : Left Limit-switch
- RL : Right Limit-switch
- ET : Empty tank
- TD : Tank down
- TU : Tank up

We have the following OUTPUTS:

- MR : Cart motor to the right
- ML : Cart motor to the left
- MTD : Tank's motor down
- MTU : Tank's motor up

Exercise 1



Exercise 1

Using structured text, the code can be written in several ways.

Since structured text is an high level language, the code can be organized as preferred by the developer, depending on what he needs.

N.B.: it is a double-edged sword: it is possible to write a not understandable code!

Exercise 1

«Ladder-based» solution

```
PROGRAM _INIT
  ML := 0;
  MR := 0;
  MTD := 0;
  MTU := 0;
END_PROGRAM

PROGRAM _CYCLIC
  IF S AND LL AND NOT ET THEN
    MR := 1;
  END_IF;
  IF RL THEN
    MR := 0;
  END_IF;
  IF RL AND NOT LL AND TU AND NOT TD AND NOT ET THEN
    MTD := 1;
  END_IF;
  IF RL AND NOT LL AND TD AND NOT TU THEN
    MTD := 0;
  END_IF;
  IF RL AND NOT LL AND TD AND NOT TU AND ET THEN
    MTU := 1;
  END_IF;
  IF RL AND NOT LL AND TU AND NOT TD THEN
    MTU := 0;
  END_IF;
  IF RL AND NOT LL AND TU AND NOT TD AND ET THEN
    ML := 1;
  END_IF;
  IF LL AND NOT RL THEN
    ML := 0;
  END_IF;
END_PROGRAM
```

Exercise 1

«State-based» solution

```
PROGRAM _INIT
  ML := 0;
  MR := 0;
  MTD := 0;
  MTU := 0;
  State := 0;
END_PROGRAM

PROGRAM _CYCLIC
  CASE State OF
    0:
      ML := 0;
      MR := 0;
      MTD := 0;
      MTU := 0;
      IF S AND NOT ET THEN
        State := 1;
      END_IF;
    1:
      MR := 1;
      IF RL THEN
        MR := 0;
        State := 2;
      END_IF;
    2:
      MTD := 1;
      IF TD THEN
        MTD := 0;
        State := 3;
      END_IF;
    3:
      IF ET THEN
        MTU := 1;
        State := 4;
      END_IF;
    4:
      IF TU THEN
        MTU := 0;
        State := 5;
      END_IF;
    5:
      ML := 1;
      IF LL THEN
        ML := 0;
        State := 0;
      END_IF;
  END_CASE
END_PROGRAM
```

Exercise 1

Which is the best solution?

It depends on the type of the machine to be controlled: as we will see later, in the case of the car-wash system, the "ladder-based" solution is the simplest.

On the contrary, in the case of the machining station, the best solution is the "state-based" one.

Exercise 1.1

Let’s add to the previous exercise a maintenance stop every 100 cycles.

We have to add:

SM (stop for maintenance) as output

RM (maintenance reset) as input

N.B.: We need also an internal counter variable (N) to count how many cycles the system have excuted!

Exercise 1.1

```
PROGRAM _INIT
   ML := 0;
   MR := 0;
   MTD := 0;
   MTU := 0;
   n := 0;
END_PROGRAM

PROGRAM _CYCLIC
   CASE State OF
      0:
         ML := 0;
         MR := 0;
         MTD := 0;
         MTU := 0;
         IF S AND NOT ET THEN
            State := 1;
         END_IF;
      1:
         MR := 1;
         IF RL THEN
            MR := 0;
            State := 2;
         END_IF;
      2:
         MTD := 1;
         IF TD THEN
            MTD := 0;
            State := 3;
         END_IF;
      3:
         IF ET THEN
            MTU := 1;
            State := 4;
         END_IF;
      4:
         IF TU THEN
            MTU := 0;
            State := 5;
         END_IF;
      5:
         ML := 1;
         IF LL THEN
            ML := 0;
            n := n + 1;
            IF n>=100 THEN
               SM := 1;
               State := 6;
            ELSE
               State := 0;
            END_IF;
         END_IF;
      6:
         IF RM THEN
            SM := 0;
            n := 0;
            State := 0;
         END_IF;
   END_CASE
END_PROGRAM
```

Exercise 2

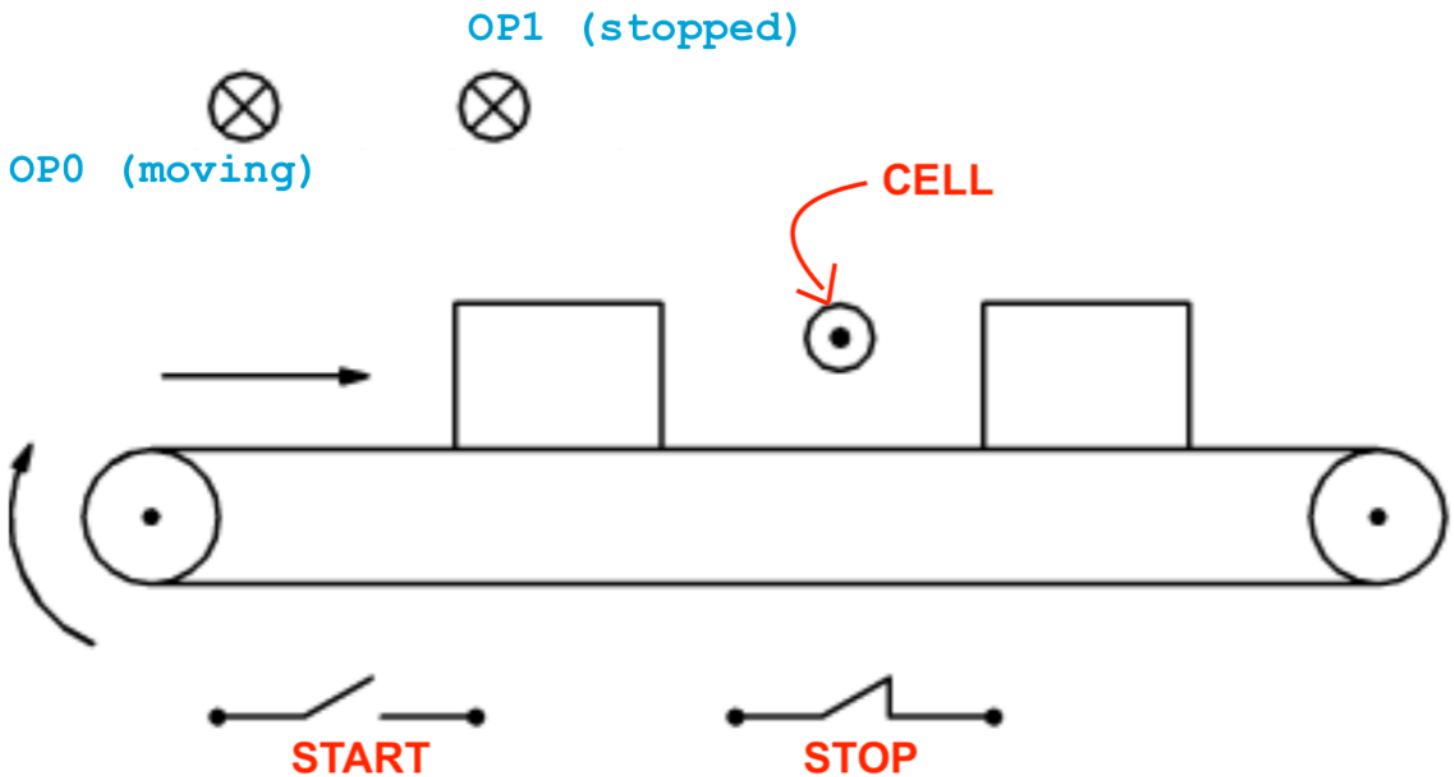
Consider a piece-counter with a conveyor belt.

The conveyor belt is moved by a motor which is controlled by two buttons: START and STOP. There are two lamps that signal the state of the conveyor belt(stopped/moving).

Each piece is placed at the beginning of the conveyor belt and, for each piece that passes in front of a photo-cell, a counter has to be incremented. The system has to automatically stop itself every 50 pieces.

The conveyor belt can be stopped at any time by pressing the STOP button. The restarting of the conveyor belt can occur by means of the pression of the START button, but, in this case, the counter have to retain its value.

Exercise 2



Exercise 2

- At startup, the motor that drives the conveyor belt must be halted, so only *OP1* has to be on.
- By pressing the *START* button, the motor starts: OP0 has to turn on and OP1 has to turn off.
- Every time the motor is running and a new piece is detected in front of the photocell *CELL*, the counter has to be increased.
- When the counter reach the value 50, the conveyor belt has to be stopped: OP1 has to turn on and OP0 has to turn off. *(in a following restart, the counter must start from the value 0).*
- When the conveyor belt is stopped before the value 50 the counter must retain its value.

Exercise 2

```
PROGRAM _INIT
  OP1 := 1;
  OP0 := 0;
  State := 0;
  N := 0;
END_PROGRAM

PROGRAM _CYCLIC
  CASE State OF
    0: (* Stop *)
      IF START AND NOT STOP THEN
        OP0 := 1;
        OP1 := 0;
        State := 1;
      END_IF;
    1: (* Moving *)
      IF EDGEPOS(CELL) THEN
        N := N+1;
      END_IF;
      IF STOP THEN
        State := 0;
        OP1 := 1;
        OP0 := 0;
      END_IF;
      IF N>=50 THEN
        N := 0;
        State := 0;
      END_IF:
  END_CASE;
END_PROGRAM
```

Exercise 3

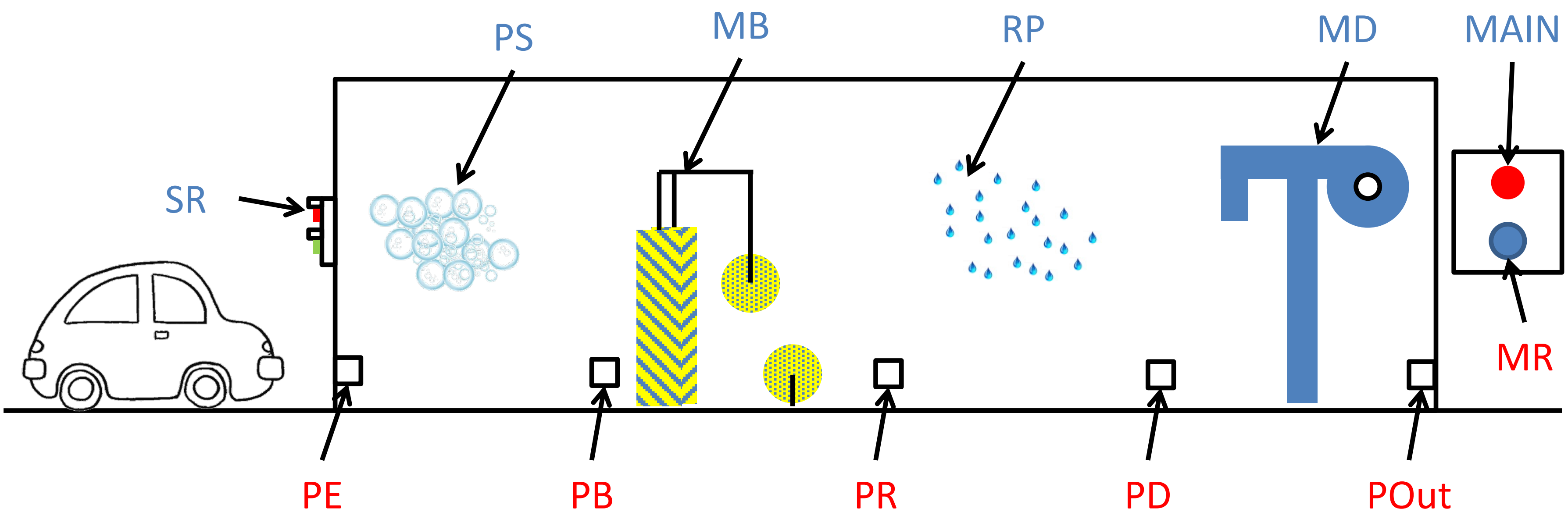
Consider an automatic carwash system.

The customer approaches to the conveyor belt when the semaphore is green. The phases of the washing are: soaping, brushing, rinsing e drying.

All the phases are preceded by a photocell that signals the arrival of the car in that new section of the carwash.

Every 1000 washing, the carwash system have to block itself waiting for the maintenance, that is executed by an operator.

Exercise 3



Input		Output	
PE	Entry photocell	SR	Stop semaphore (0=GREEN, 1=RED)
PB	Brushing photocell	PS	Soaping pump
PR	Rinsing photocell	MB	Brushing motor
PD	Drying photocell	RP	Rinsing pump
POut	Out photocell	MD	Drying motor
MR	Maintenance reset	MAIN	Halt for maintenance

Exercise 3

The system can be developed as a set of sub-systems:

- Soaping
- Brushing
- Rinsing
- Drying

Each of this sub-systems has to start when the previous photocell activates its output and has to stop when the next photocell deactivates its output.

Exercise 3

As shown in the previous lesson, this exercise requires a "distributed" solution for each part of the system.

With the SFC language we cannot manage more than a single «execution cycle» with a single code. For this reason we need more than a program: one for each section!

In the program that manages the soaping, we will manage also the semaphore and the mainteinance.

Exercise 3

```
FUNCTION_BLOCK PlantSection
  IF NOT Activation THEN
    IF EDGEPOS(Pentry) THEN
      Activation := 1;
    END_IF;
  ELSE
    IF EDGENEG(Pexit) THEN
      Activation := 0;
    END_IF;
  END_IF;
END_FUNCTION_BLOCK
```

Each section is activated when the entry photocell gives a positive edge and is deactivated when the exit photocell gives a negative edge.

Inputs:
Pentry, Pexit

Outputs:
Activation

Exercise 3

```
FUNCTION_BLOCK FirstSection
  IF n<1000 THEN
    IF NOT StopSemaphore THEN
      IF EDGEPOS(Pentry) THEN
        StopSemaphore := 1;
        Activation := 1;
      END_IF;
    ELSE
      IF EDGENEG(Pexit) THEN
        StopSemaphore := 0;
        Activation := 0;
        n := n + 1;
      END_IF;
    END_IF;
  ELSE
    StopSemaphore := 1;
    Activation := 0;
    MaintenanceRequired := 1;
    IF MainReset THEN
      n := 0;
      StopSemaphore := 0;
      MaintenanceRequired := 0;
    END_IF;
  END_IF;
END_FUNCTION_BLOCK
```

The first section (soaping) will have to manage the semaphore and the scheduled maintenance. For this reason we have to use a different Function Block

Inputs:
Pentry, Pexit, MainReset

Outputs:
StopSemaphore, MaintenanceRequired, Activation

Exercise 3

```
PROGRAM _INIT
END_PROGRAM

PROGRAM _CYCLIC
   Soaping(Pentry := PE, Pexit := PB, MainReset := MR);
   MAN := Soaping.MaintenanceRequired;
   SS := Soaping. StopSemaphore;
   PI := Soaping.Activation;
   Brushing(Pentry := PB, Pexit := PR);
   MS := Brushing.Activation;
   Rinsing(Pentry := PR, Pexit := PD);
   PR := Rinsing.Activation;
   Drying(Pentry := PD, Pexit := POut);
   MA := Drying.Activation;
END_PROGRAM
```

The main file of the program is very simple: a single instance of the function block is created to manage each plant section.

N.B.: Why have we used more that a single type of Function Block?

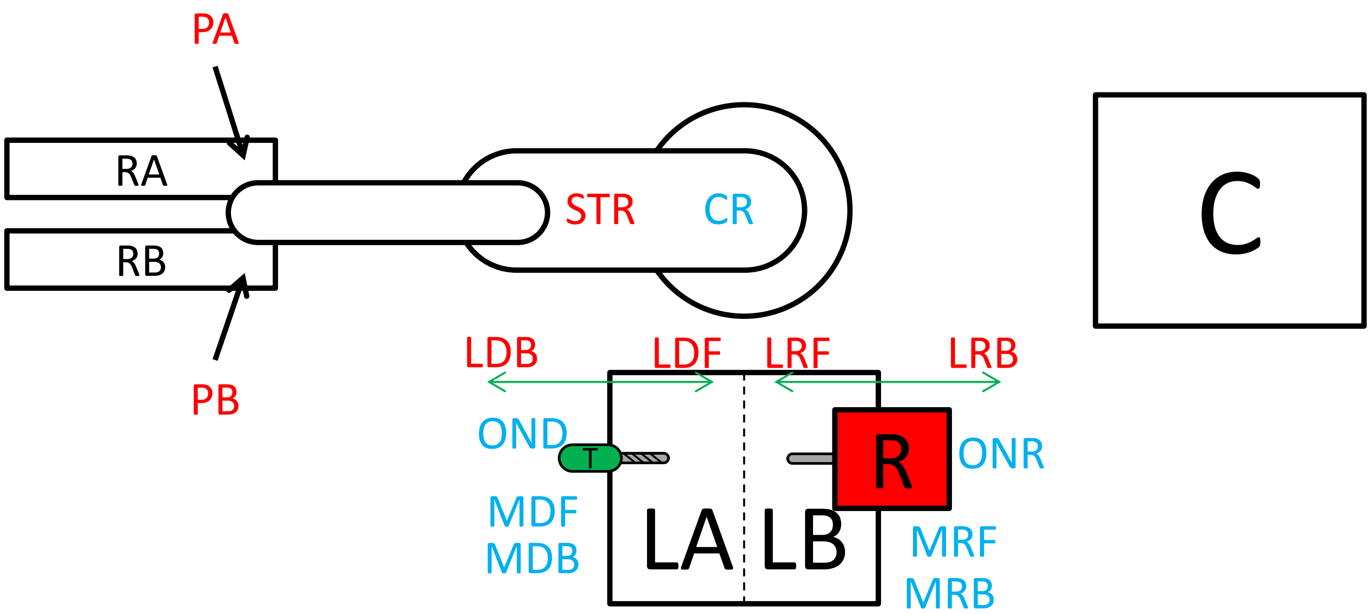
Exercise 4

Consider a system used for automatic drilling and riveting metal sheets.

When the two sheets are available, a robot takes them (one by one) and places them over an assembly mask. After the end of the placing, the pieces are perforated by an automatic drill (5 seconds) and rivetted by a riveter (10 seconds).

At the end of the process, the robot move the produced product in a container.

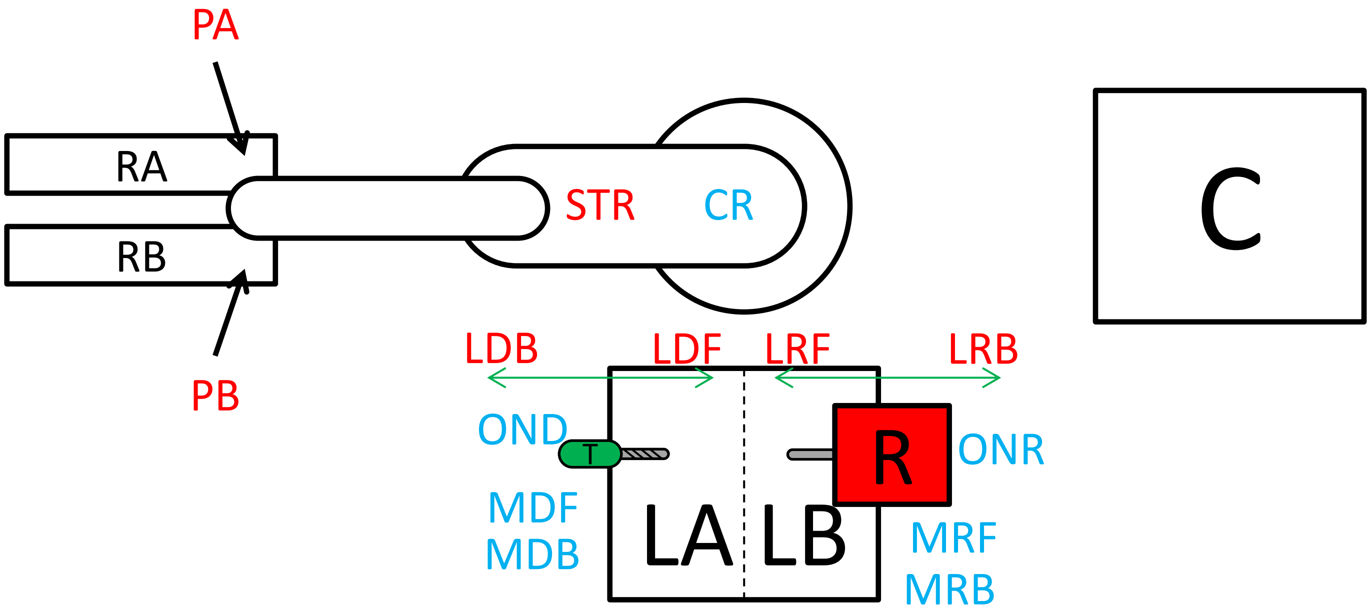
Exercise 4



Input

PA	Piece A available	LDF	Drill forward limit
PB	Piece B available	LRB	Riveter backward limit
STR	Status robot (0=END, 1=EXECUTING)	LRF	Riveter forward limit
LDB	Drill backward limit		

Exercise 4



Output

CR	Robot command (0=STOP, 1=take piece A, 2=take piece B, 3=deposit final product in C)	OND	Drill ON
		MRF	Riveter motor forward
MDF	Drill motor forward	MRB	Riveter motor backward
MDB	Drill motor backward	ONR	Riveter ON

Exercise 4

The steps to be executed to create a finite product are:

1)Wait until PA=1 and PB=1

2)Send command 1 to the robot and wait the end of its execution

3)Send command 2 to the robot and wait the end of its execution

4)Move the drill forward until the forward-limit is reached

5)Operate the drill for 5 seconds

6)Move the drill backward until the backward-limit is reached

7)Move the riveter forward until the forward-limit is reached

8)Operate the riveter for 10 seconds

9)Move the riveter backward until the backward-limit is reached

10)Send command 3 to the robot and wait the end of its execution

11)Send command 0 to the robot

Exercise 4

```
PROGRAM _INIT
  State := 0;
END_PROGRAM

PROGRAM _CYCLIC
  CASE State OF
    0: (* Stop *)
      IF PA AND PB THEN
        CR := 1;
        State := 1;
        STR := 1;
      END_IF;
    1: (* Take piece A *)
      IF NOT STR THEN
        CR := 2;
        State := 2;
        STR := 1;
      END_IF;
    2: (* Take piece B *)
      IF NOT STR THEN
        State := 3;
      END_IF;
    3: (* Drill forward *)
      MDF := 1;
      IF LDF THEN
        MDF := 0;
        State := 4;
      END_IF;
    4: (* Perforing *)
      OND := 1;
      t := t + dt;
      IF t>=T#5s THEN
        t := T#0s;
        OND := 0;
        State := 5;
      END_IF;
    5: (* Drill backward *)
      MDB := 1;
      IF LDB THEN
        MDB := 0;
        State := 6;
      END_IF;
    6: (* Riveter forward *)
      MRF := 1;
      IF LRF THEN
        MRF := 0;
        State := 7;
      END_IF;
    7: (* Riveting *)
      ONR := 1;
      t := t + dt;
      IF t>=T#10s THEN
        t := T#0s;
        ONR := 0;
        State := 8;
      END_IF;
    8: (* Riveter backward *)
      MRB := 1;
      IF FRI THEN
        LRB := 0;
        CR := 3;
        State := 9;
        STR := 1;
      END_IF;
    9: (* Deposit final product in the
           container*)
      IF NOT STR THEN
        State := 0;
      END_IF;
  END_CASE;
END_PROGRAM
```

Conclusions

<u>Final considerations on Structured Text</u>

It is the higher level language of those available in the IEC 61131 norm.

It is simple but the developer has to pay attention on several software engineering aspects (like it is also with other programming languages for computers).

Unfortunately in the industry it is not very used: in the latest years its use is increasing thanks to some tools that automatically generates the code from the model of the system.