ST programming

infoPLC.net

Automation Systems

NJ501 : IEC61131-3 ST Programming

This section will cover

- Overview of a ST program
- ST Editor
- Syntax rules
- Assignment Statement
- Comments
- ST Operators
- Standard ST commands
  - Conditional statements
  - Branching
  - Conditional Loops
- Function Block Calls in ST
- Function Calls in ST
- Data Types handling in ST
- Comparing ST vs. LD structures

Overview of a ST program

```
148  //Program Overview
149  IF NOT  fault THEN
150      Ready:=TRUE; //Green Light in ON state
151      AlarmSignal:=FALSE;
152      (*Turn OFF the red light
153          AND enable machine FOR operation*)
154
155      IF NOT HomingDone THEN
156          ExecuteHoming:=TRUE;//Make homing FOR 1st TIM
157          OpenGrip:=TRUE; //OpenGrip grip FOR Enable g
158          MyString:='This is a string';
159
160      END_IF;
161
162  ELSE
163      Ready:=FALSE;
164      ExecuteHoming:=FALSE;
165      AlarmSignal:=TRUE;
166      (*turn red light ON*)
167  END_IF;
168
169  Data1:=45676;
```

BLUE:

Flow control program statements and operators

GREEN:

User comments

DARK RED:

Text strings variables

BLACK:

User functions, user variables, and user functions blocks

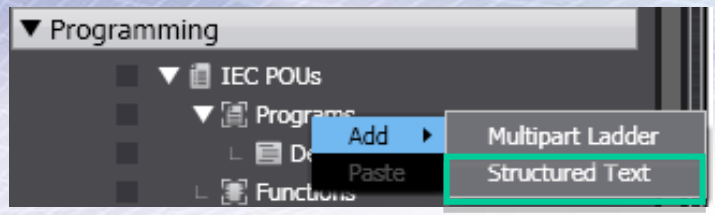
light BLUE:

Global variables

Overview of a ST program

- **Benefits in ST programming**
  - Code easily **readable** by the user
  - **Portability** (99%) between IEC61131-3 3rd party ST code
  - **Editable** by a **text editor**
  - Coding **Efficiency** better than LD or FBD
  - Excellent **handling for Structures** (Data Types)
  - **Development time** reduced (when you're used to ST)
  - You can **merge** LD and ST in same LD POU (*InLine ST*)

ST editor description



Configuration

Programming

DemoST1

Internals
Externals
Constants

Name	Data Type	Initial Value	Address	Retain	Comment
CountDown	INT	0			
Value	INT	0			
Ready	BOOL	False			
AlarmSignal	BOOL	False			
Power_Drive	MC_Power				

```
6
7       IF NOT HomingDone THEN
8           ExecuteHoming:=TRUE;//Make homing for 1st time
9           OpenGrip:=TRUE;     // Open grip for enable grasping
10      END_IF
11
12  ELSE
13      Ready:=FALSE;
14      ExecuteHoming:=FALSE;
15      AlarmSignal:=TRUE;    (*Turn red light ON*)
16
17      (*Lights must be ON in while machine is not
18      in ready state ready *)
19
20
21  END_IF
22
23  Signal := Power AND Enable AND NOT Fault ;          //Check signal avalability
24  Position:= Encodervalue * UserUnits / Increments;   //Read position in usr units
```

Variable declaration area

ST program editor area

Assignment Statement

- Assigning values, expressions or conditions to a variable is done using the "**:=**" sign combination:

```
Ready:=FALSE;
AlarmSignal:=TRUE;
OpenGrip:=FALSE;
```

- It is allowed to put spaces or tabs to improve readability

```
Signal:=     Power AND Enable AND NOT fault;          //check Signal availability
Position:=   EncoderPosition  *  UserUnits  /  Increments;   //Read Position in  UserUnits units
```

- WORD assignments can be expressed also in binary and hexadecimal.

```
ValueINT:=125;          //value expressed in integer
ValueHEX:=16#7D;        //value expressed in hexadecimal
ValueBIN:=2#1111101;    //value expressed in binary
```

Assignment Statement

- Statements **always** must end with a single semicolon " **;** "

```
Ready:=FALSE;
AlarmSignal:=TRUE;
OpenGrip:=FALSE;
```

- For a complex or longer expressions, it is allowed to split the expression in multiple lines, using line feed (↵). Do not miss to end with a semicolon.

```
SafetyChain:=   (Sensor_1 AND Sensor_2 AND NOT Sensor3)
                AND (PowerEnable AND NOT StandStill OR MotorDisabled OR Compressor)
                OR(MainsSupply AND PowerSupply OR SecondarySupply)
                AND NOT (SafetyRelays OR SecurityLatch OR SafetyStop) ;
```

Using comments

- Comments can be added in any place for a ST program using parentheses and asterisks for opening and closing a comment “**(*.....*)**”, in a single or multi line comment

```
AlarmSignal:=TRUE;     (*Turn red light ON*)

(*Lights must be ON in while machine is not
in ready state ready *)
```

- A simplest way to add a single comment is using double slash “ *//* ” before your comment

```
IF NOT HomingDone THEN
    ExecuteHoming:=TRUE;//Make homing for 1st time
    OpenGrip:=TRUE;      // Open grip for enable grasping
END_IF ;
```

Syntax rules

- **Variable declaration**:
  - The editor doesn't differentiate between variables declared in uppercase (or containing them) and lowercase. But for readability and consistency , is a **good practice** to maintain the format for these variables during the program writing:

```
Signal := Power AND Enable AND NOT Fault ;
Position:= Encodervalue * UserUnits  / Increments;

IF SIGNAL AND position>1000 THEN
    OutBounds:=TRUE;
END_IF;
```

Not recommended

```
Signal := Power AND Enable AND NOT Fault ;
Position:= Encodervalue * UserUnits  / Increments;

IF Signal AND Position>1000 THEN
    OutBounds:=TRUE;
END_IF;
```

More consistent

Syntax rules

- **Keywords (reserved)**
  - ALL keywords are forbidden to use because they are ST commands:

AND, BY, CASE, DO, ELSE, ELSIF, EXIT, FALSE, FOR, IF, NOT, OF, OR, REPEAT, RETURN, THEN, TO, TRUE, UNTIL,WHILE, XOR, END_IF, END_WHILE,END_CASE, END_REPEAT…

  - Special characters cannot be used to declare variable name. However, underscore sign ("_") can be used (not in the 1st /end position).

<=, >=, <>, :=, .., &, (*,*), %,$,@...

  - Definition Types and User types cannot be used as a variable name

USINT, SINT, BYTE, UINT, INT,WORD, REAL, DINT, UDINT, DWORD, LREAL, LINT, ULINT, LWORD…

Exercise #1

- Find the **7** mistakes in this code section

```
IF NOT Fault THEN
    Ready:= TRUE; // Green light in ON state
    AlarmSignal:=FALSE;
    (*Turn OFF the red light
        and enable machine for operation*)
    Light3:= FALSE;

    IF Position=3456 THEN
        ExecuteHoming:=TRUE;//Make homing for 1st time
        OpenGrip:=TRUE;      // Open grip for enable grasping
        MotorEnable:=FALSE;
        Exit:=TRUE;

    END_IF;

ELSE
    Ready:=FALSE;
    AlarmSignal:=TRUE;     (*Turn red light ON*)
    opengrip:=FALSE;
    (*Lights must be ON  while machine is not
    in ready state ready *)
END_IF;
```

ST Operators

( )	Operation Execution Order	Value:=(1+2) *(3+4) // Value is 21 Precedence order : **( ), * / , + -**
******	Exponent	Value:= 2**8 ; // Value is 256
NOT	Negation of a logical condition	Value:=NOT TRUE; //Value is FALSE
*****	Multiplication	Value:=8 * 100; // Value is 800
/	Division	Value:=200 / 25; // Value is 8
+	Addition	Value:=200 + 25; // Value is 225
-	Subtraction	Value:=200 - 25; // Value is 175
MOD	Remainder	Value:=10 MOD 6; // Value is 4
<,>,<=,>=	Comparison	Value:= 60 > 10; // Value is TRUE
=	Equal condition	Value:= 8=7; // Value is FALSE
<>	Not equal condition	Value:= 8<>7; // Value is TRUE
&, AND	Logical AND	Value:=2#1001 AND 2#1100; //Value is 2#1000
XOR	Logical Exclusive OR	Value:=2#1001 XOR 2#1100; //Value is 2#0101
OR	Logical OR	Value:=2#1001 XOR 2#1100; //Value is 2#1101

ST Standard Commands

- Conditional Statements
  - **IF..THEN....END_IF**
  - **IF..THEN….ELSE….END_IF**
  - **IF..THEN….ELSIF..THEN…END_IF**
- Branching
  - **CASE..OF….END_CASE**
- Conditional loops
  - **FOR.. (BY) .. DO..END_FOR**
  - **WHILE..DO….END_WHILE**
  - **REPEAT...UNTIL…END_REPEAT**
  - **EXIT**

ST Standard commands

- Single condition command: **IF .. THEN .. END_IF**

```
IF <condition_expression> THEN
    <statement_1>;
END_IF;
```

- The <*condition_expresion*> is evaluated and executes the statements between IF..THEN and END_IF when the condition is TRUE.
- If not, the command misses these statements and jumps after END_IF to continue the code execution

ST Standard commands

IF..THEN..END_IF example:

```
State:=0;
IF Enable AND NOT PowerON THEN
    State:=10;
END_IF;

Value:=State;(*if previous condition would be TRUE, Value will be 10.
              If not,  Value will be 0 *)
```

- Here, *Value* variable will be 10 if *Enable* is TRUE and *PowerON* is FALSE.
- If not, *Value* is 0.

ST Standard commands

– Therefore, a ELSE command can be added to execute statements when the condition is FALSE

```
IF <condition_expression> THEN
    <statement_1>;      <-- When TRUE
ELSE
    <statement_2>;      <-- When FALSE
END_IF;
```

– When the <*condition_expresion*> is TRUE the statements between IF..THEN and ELSE are executed. If the condition evaluation is FALSE, the statements between ELSE and END_IF are executed.

ST Standard commands

- Extended Condition command: **IF .. THEN .. ELSIF .. END_IF**

```
IF <condition_expression_1> THEN <statement_1>;
 ELSIF <condition_expression_2> THEN <statement_2>;
 ELSIF <condition_expression_3> THEN <statement_3>;

      ...
  ELSIF <condition_expression_n> THEN <statement_n>;
ELSE <statement_m>;
END_IF;
```

  - The <*condition_expresion_1*> is evaluated and executes the <statement_1> if it is TRUE. In this case, it ends the evaluation.
  - But if it is not met (FALSE) , it checks the next ELSIF conditions and executes his statements if it is TRUE. If not, it checks the next ELSIF condition, and so on. So, it allows simultaneous condition checking.
  - If none of these conditions are fulfilled , the sentences under ELSE command are executed instead.

ST Standard commands

IF_THEN..ELSIF..END_IF example:

```
IF Enable AND NOT PowerON THEN  //Checks the first condition
    State:=10;
ELSIF NOT Enable AND PowerON THEN //Also checks the 2nd
    State:=20;
ELSIF Enable AND PowerON THEN //.. and the 3rd
    State:=30;
ELSE
    State:=0;       //If no one of these conditions are true
END_IF;
```

ST Standard commands

–It is possible to nest IF…THEN..ELSIF..END_IF into a general IF..END_IF selection command.

```
IF Enable AND NOT PowerON THEN
    State:=10;
    IF Temperature >100 then
        Frezzer:=TRUE
    ELSE
        Frezzer:=FALSE;
        IF Setpoint>0 THEN
            EnableConveyor:=TRUE;
            State:=20;
        ELSE
            EnableConveyor:=FALSE;
            State:=40;
        END_IF;
    END_IF;

ELSIF Enable AND PowerON and Setpoint>20 THEN
    Feeder:=TRUE;
    Speed:=200;
ELSE
    Feeder:=FALSE;
    Speed:=0;
    EnableConveyor:=FALSE;
END_IF;
```

In a nested code section, take care to control the END_IF for each nested section.

For this reason, is strongly recommended use the **indenting** for the inlay nested commands ; also improves the readability and is a **good practice**

ST Standard commands

- Branching commands:

  CASE..OF…ELSE..END_CASE

```
CASE <integer_expression> OF
   <integer_expression_value_1>:<statement_1>;
   <integer_expression_value_2>:<statement_2>;
            ...
   <integer_expression_value_n>:<statement_n>;
ELSE<statement_m>;
END_CASE;
```

  - The <*integer_expression*> is evaluated and depending its value, executes the code for each matching value (<*integer_expression_value_n*>)
  - If none of these conditions are fulfilled , the statements under ELSE command are executed
  - After complete the evaluation, it jumps after END_CASE command

CASE..OF.. END_CASE example:

```
CASE State OF

   10: SetPoint:=45;  //Executes  IF State is 10
      Cooler:=TRUE;

   20: SetPoint:=65;  //Executes  IF State is 20
      Cooler:=FALSE;
      IF Speed>200 THEN
         State:= 30;
      END_IF;
                    ;
   30: SetPoint:=95;  //Executes  IF State is 30
      Cooler:=TRUE;
      Speed:=100;

ELSE
      Speed:=0;

END_CASE;
```

Note that **if the condition changes during the evaluation**, it is evaluated at the next PLC cycle

ST Standard commands

For a CASE..OF..END_CASE if the condition changes during the evaluation, it is evaluated at the next PLC cycle

value=10

```
CASE value OF
    10:
        value:=20;
    20:
        value:=30;
    30:
        value:=40;
    40:
        value:=50;
ELSE
    value:=0;
END_CASE;
```

Next scan cycle

value=20

value=20

```
CASE value OF
    10:
        value:=20;
    20:
        value:=30;
    30:
        value:=40;
    40:
        value:=50;
ELSE
    value:=0;
END_CASE;
```

Next scan cycle

value=30

value=30

```
CASE value OF
    10:
        value:=20;
    20:
        value:=30;
    30:
        value:=40;
    40:
        value:=50;
ELSE
    value:=0;
END_CASE;
```

Next scan cycle

value=40

value=40

```
CASE value OF
    10:
        value:=20;
    20:
        value:=30;
    30:
        value:=40;
    40:
        value:=50;
ELSE
    value:=0;
END_CASE;
```

Next scan cycle

value=50

ST Standard commands

CASE..OF.. END_CASE example (using Enums):

```
(* example usig ENUM in a  CASE..END_CASE;*)

CASE State OF

      low:
            SetPoint:=45;
            Cooler:=TRUE;
      medium:
            SetPoint:=65;
            Cooler:=FALSE;
            IF Speed>200 THEN
               Speed:= 30;
            END_IF;

       high:
            SetPoint:=95;
            Cooler:=TRUE;
            Speed:=100;

ELSE
       Speed:=0;

END  CASE;
```

ENUM declaration

	Name	Enum Value
▼	States	
	low	10
	medium	20
	high	30

Variable declaration

program_0

Internals / Externals

Name	Data Type	Initial Value
State	States	
Fault	BOOL	False

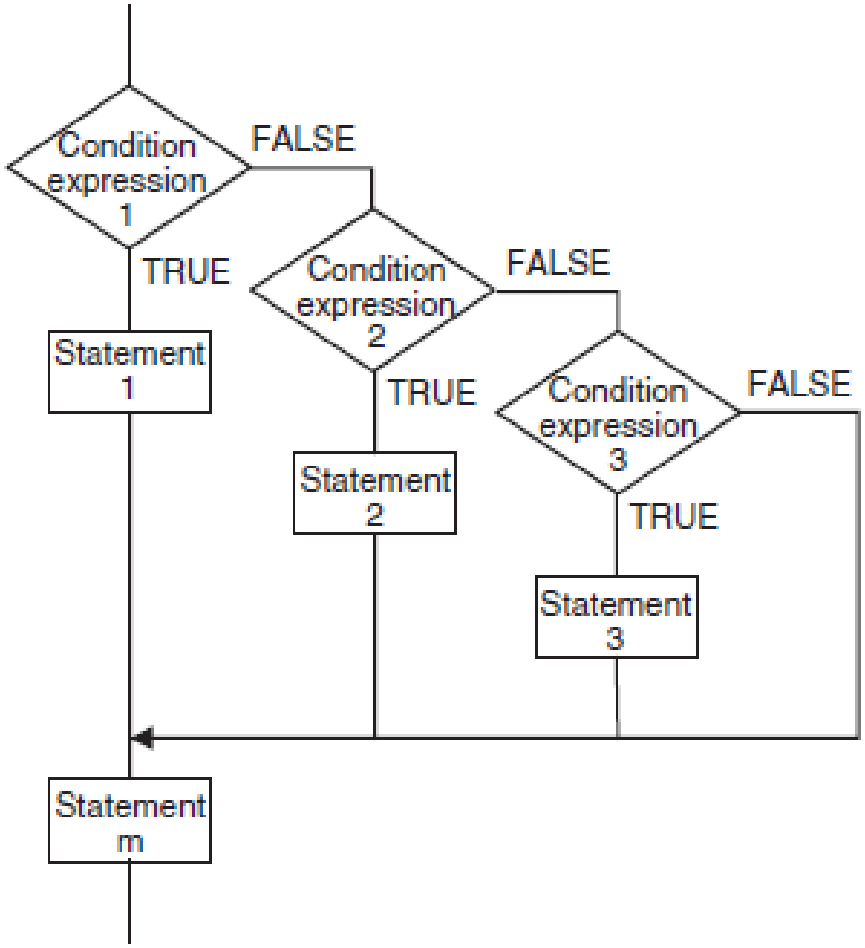
ST Standard commands

– In a CASE..OF..END_CASE structure, **multiple matching values** declaration are supported , in this way:

```
CASE A OF

   1:
      X:=1;

   2,5:    //if X value is 2 or 5

      X:=2;

   6..10:  //if X value is within 6 to 10 range

      X:=3;

   11,12,15..20:   //if X value is 11 0r 12, or within 15 and 20

      X:=4;
ELSE

   X:=0;

END_CASE;
```

Exercise #2

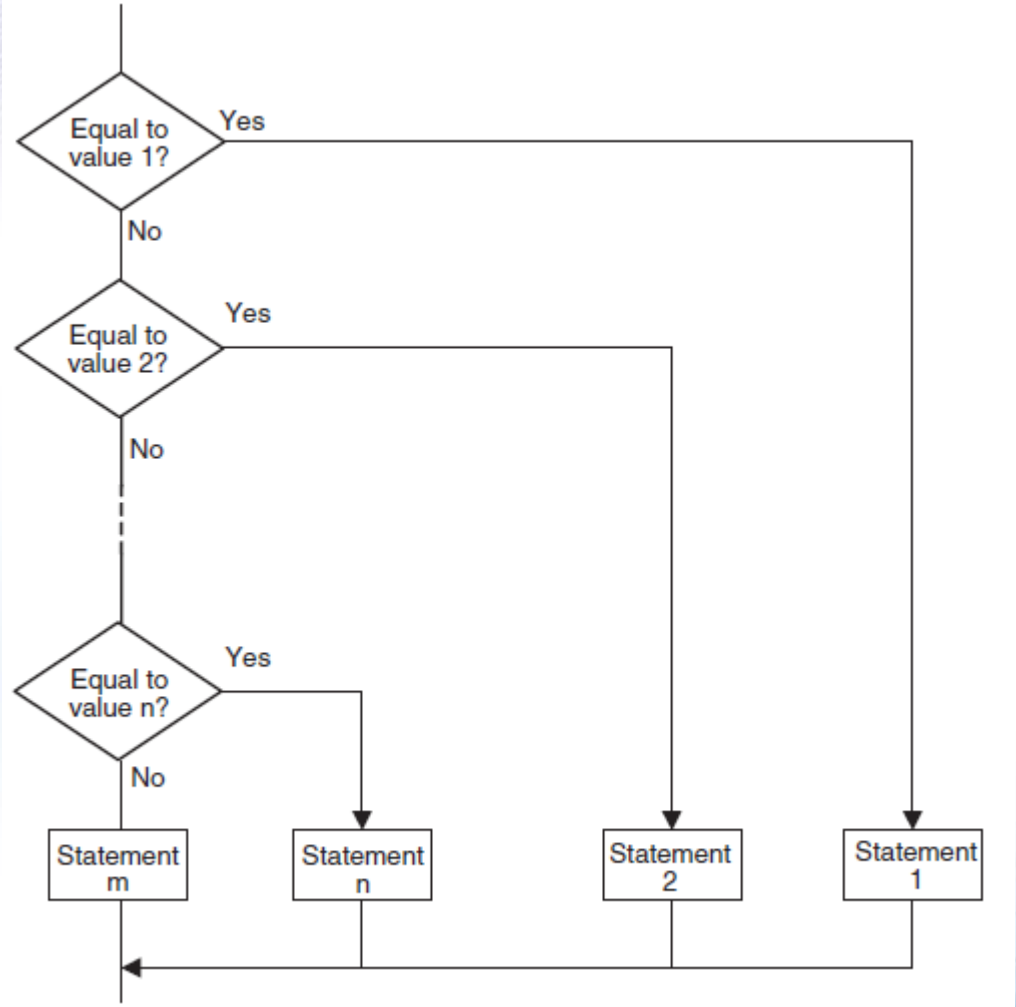
- Make a short program following the next flow code
  (we assume the condition expression is evaluating the same variable)




```
IF A =1 THEN
     Value:=10;

ELSIF A=2 THEN
     Value:=20;

ELSIF A=3 THEN
     Value:=30;

ELSIF A=4 THEN
     Value:=40;

ELSE
     Value:=50;
END_IF;
```

Exercise #3

- Make a short program following the next flow code




```
CASE A OF

    1:
      Value:=10;

    2:
      Value:=20;

    3:
      Value:=30;

ELSE
      Value:=40;

END_CASE;
```

Exercise #4

- Make a program in ST for a **pumping sequence**: when pump_1 reaches its setpoint, enables pump_2.
- After, when pump_2 reaches its setpoint , enables pump_3.

Hint:

Use CASE..OF..END_CASE for the sequence

```
IF Enable THEN

    CASE State OF
        10:   enable_Pump_1:=TRUE;
              IF Pump_1_OK THEN
                  State:=20;
              END_IF;

        20: enable_Pump_2:=TRUE;
              IF Pump_2_OK THEN
                  State:=30;
              END_IF;

        30:enable_Pump_3:=TRUE;
              IF Pump_3_OK THEN
                  State:=40;
              END_IF;

        40:   //DO something...
              ;
    END_CASE;

ELSE

    State:=10;
    enable_Pump_1:=FALSE;
    enable_Pump_2:=FALSE;
    enable_Pump_3:=FALSE;

END_IF;
```

ST Standard commands

- Conditional Loop commands :**FOR.. (BY) .. DO..END_FOR**

```
FOR <FOR_variable> := <initial_value> TO <end_value_expression> BY <increment_expression>
    DO
  <statement>;
END_FOR;
```

- Executes the statements within FOR ..END_FOR, in a loop from <*initial_value*> to <*end_value*>.
- When the counting is fulfilled it goes out the loop, after END_FOR.
- The **BY command** is optional, it allows to count in <*increment_expression*> steps .
- The execution control will remain into the loop up to reach the condition. It could origin a **Task Period Exceeded Error** for a large loops.

ST Standard commands

FOR.. (BY) .. DO..END_FOR example:

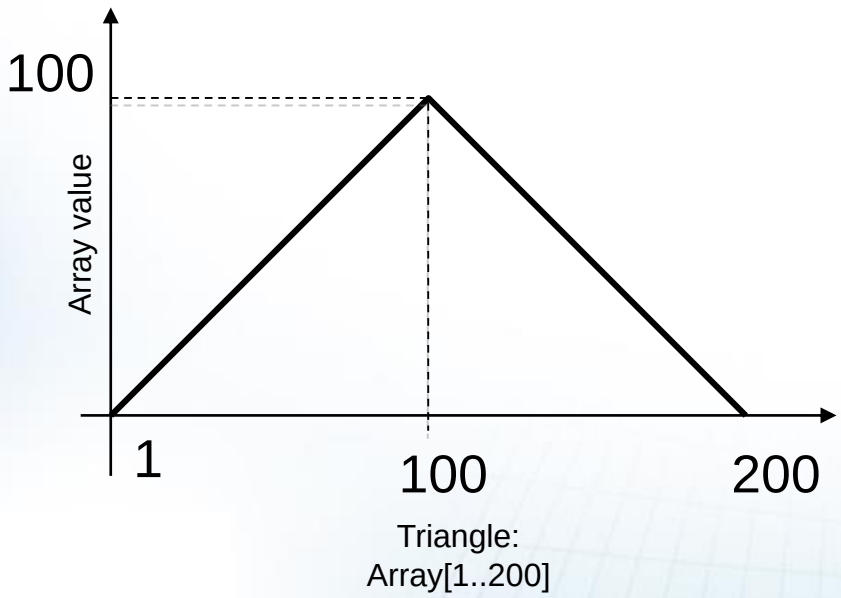
```
FOR CountDown:=100 TO 0 BY -1 DO
    Value:= Value + 5;
    Toggle:= NOT Toggle;
END_FOR;
```

– For a FOR..END_FOR without **BY command**, the ST interpreter assumes that the increment for the control variable is +1 by default. Let's see an example:

```
FOR CountDown:=3 TO 56 DO
    Value:= Value + CountDown;
END_FOR;
```

Exercise #5

- Build a ST program that writes a triangular signal period, from 1 to 200, in a array variable



Triangle:
Array[1..200]

```
FOR index:=1 TO  200 DO

    IF Index <=100 THEN
        triangle[index]:=index;
    ELSE
        triangle[index]:=INT#200-index;
    END_IF;

END_FOR;
```

We must force INT datatype otherwise system will use DINT

ST Standard commands

- Conditional Loop commands :
  - **WHILE.. DO..END_WHILE**

```
WHILE <condition_expression> DO
   <statement>;
END_WHILE;
```

  - Executes the <statement> within the WHILE ..END_WHILE loop continuously while the <*condition_expression*> is fulfilled.
  - When the condition is not fulfilled, the loop is over and it jumps to the sentences after END_WHILE
  - The execution control will remain into the loop while the condition is fulfilled. It could origin a **Task Period Exceeded Error** for a large loops.

ST Standard commands

– **WHILE..DO..END_WHILE** example:

```
WHILE (Counter<10) DO
    Counter:=Counter+1;
END_WHILE;

Value:=Counter;
```

Here, the loop is released when *Counter* reaches 10, and *Value* is 10.

– **Avoid WHILE loops like this:**

```
WHILE (TRUE) DO  // infinite loop. Avoid !!
    Counter:=Counter+1;
END_WHILE;
```

ST Standard commands

- Conditional Loop commands :
  - **REPEAT .. UNTIL..END_REPEAT**

```
REPEAT
   <statement>;
UNTIL <condition_expression>
END_REPEAT;
```

- First, <*statement*> is executed unconditionally. Then the <*condition_expression*> is evaluated. If <condition_expression> is FALSE, <*statement*> is executed again. If <condition_expression> is TRUE, <*statement*> is not executed and the loop is over.
- The execution control will remain into the loop while the condition is fulfilled. It could origin a **Task Period Exceeded Error** for a large loops.

ST Standard commands

REPEAT..UNTIL..END_REPEAT example:

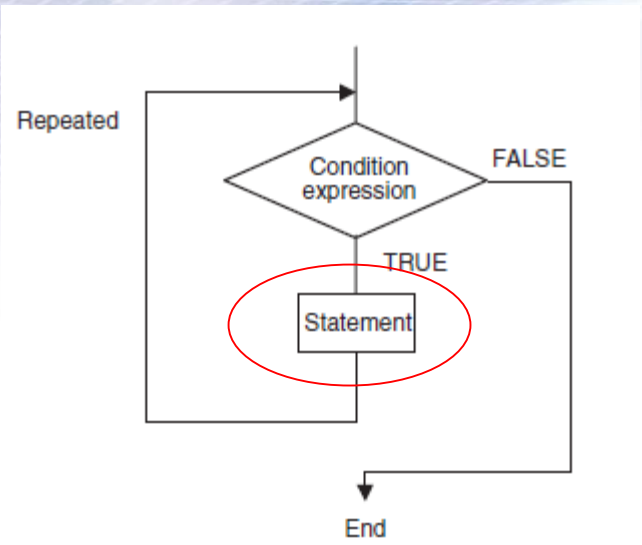
```
REPEAT
    Counter:=Counter+1;
UNTIL Counter=10
END_REPEAT;

Value:=Counter;
```

After END_REPEAT, *Value* is equal to 10

ST Standard commands

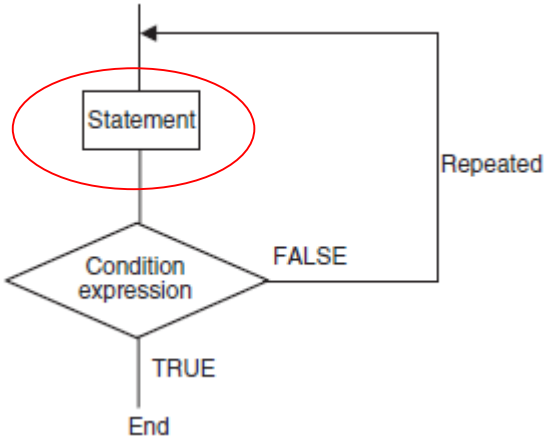
- Differences between **WHILE** loop vs. **REPEAT** loop

WHILE



- Condition Expression is evaluated **BEFORE** entering into the loop
- Statement won't execute if the condition is not satisfied in the 1st loop cycle

REPEAT



- Condition Expression is evaluated **AFTER** entering into the loop
- Statement is at least executed one time

ST Standard commands

- Conditional Loop breaking command :

  EXIT

```
FOR (WHILE, REPEAT) <statement>
        …
IF<condition_expression> THEN EXIT;
END_IF;
        …
END_FOR (WHILE, REPEAT);
```

  - EXIT command is used usually in combination with a loop
  - The purpose of EXIT is breaking a loop in a program-defined situation
  - It can be useful for avoiding infinite or large loops that could origin a Task Period Exceeded Error

ST Standard commands

EXIT example:

```
loop:=0;
WHILE(NOT Input_1) DO
     IF loop=5 THEN EXIT; END_IF;
     loop:=loop+1;
     // do something
END_WHILE;
```

– This is an example using a WHILE that checks a variable up to 5 times, then it breaks the loop

Exercise #6

- Make a Function to calculate the **n!** (factorial) for a number

$$5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$$

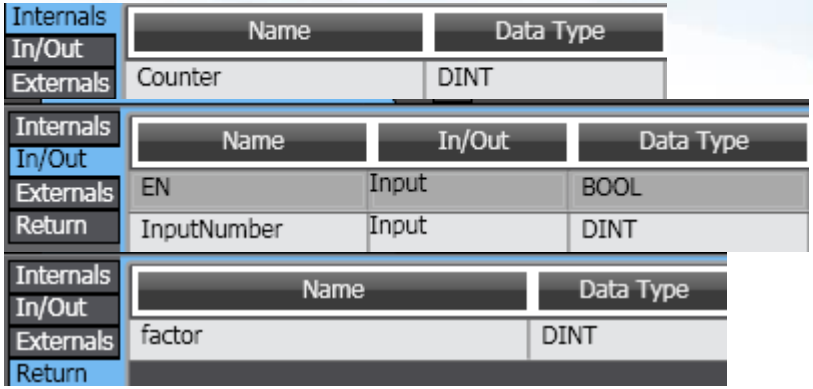
Hints:

•The value of 0! is 1.
•Check that the value entered is not negative (in that case, it will returns 0)
•Fit the result in a DWORD, maximum input number is 31

Exercise #6

Function program

```
Counter:=1;
FACTOR:=1;
REPEAT
   IF InputNumber=0 THEN
         FACTOR:=1;EXIT;
   ELSIF InputNumber<0 OR InputNumber >31 THEN
         FACTOR:=0;EXIT;
   END_IF;

   FACTOR:=FACTOR*Counter;
   Counter:=Counter+1;
UNTIL (Counter>InputNumber)
END_REPEAT;

RETURN ;
```



Example of use:

```
n:=FACTOR(InputNumber:=Number);
```

Function Block calls in ST

- A **Function Block** must be called using a previously declared instance for its FB type

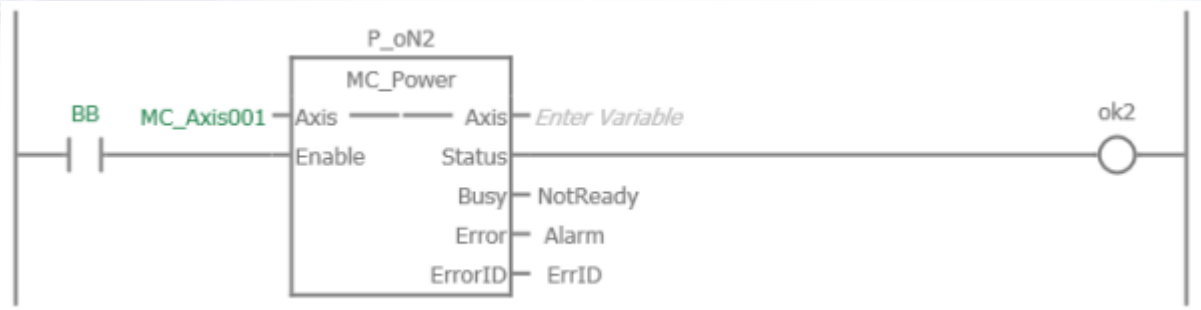

```
InstanceName (<Input_1>:=… ,
              <Input_n>:=…  ,
              <Output_1> =>… ,
              <Output_n> =>… ,
              <Input_output_1> :=… ,
              <Input_output_n>):=… );
```

- **Inputs** are assigned by using “ := “
- **Outputs** are assigned by using “ =>”
- **Input/Output** are assigned by using both (commonly “ := “)

Function Block calls in ST

Example

Name	Data Type
P_oN2	MC_Power
NotReady	BOOL
Alarm	BOOL
ErrID	WORD
ok2	BOOL

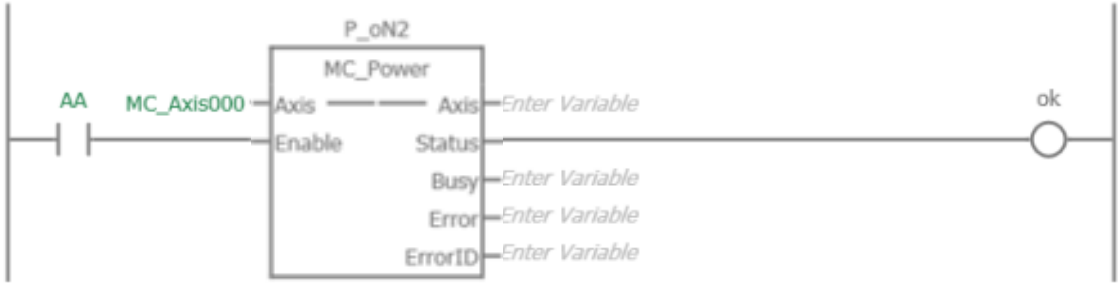



```
P_oN2(
    Axis := MC_Axis001,
    Enable := BB,
    Status =>ok2,
    Busy =>NotReady,
    Error =>Alarm,
    ErrorID =>ErrID);
```

Function Block calls in ST

- The **unused inputs and outputs** for a Function Block call can be left empty or even can be removed from the Function Block call.
- Empty parameters for a Function Block will take default values.

Example




```
P_oN2 (
   Axis := MC_Axis000,
   Enable := AA,
   Axis =>,
   Status => ok,
   Busy =>,
   Error =>,
   ErrorID =>);
```

⇔

In **Sysmac Studio: remove unused I/Os**

```
P_oN2 (
   Axis := MC_Axis000,
   Enable := AA,
   Status => ok);
```

Function & Function Block calls in ST

- There are two ways to fill the parameters:
  - By assigning the variables to its I/Os

```
(FunctionBlock) MC_Power ([_sAXIS_REF] Axis, [BOOL] Enable, [BOOL] Status, [BOOL] Busy, [BOOL] Error, [WORD] ErrorID)
Axis : Axis
Power Servo
```

```
power_1(Axis:=MC_Axis000,
        Enable:=Run,
        Status=>State,
        Busy=>Processing,
        Error=>Error_1,
        ErrorID=>ID_Error);
```

  - By assigning the variables in the order required

```
power_1(MC_Axis000,
        Run,
        State,
        Processing,
        Error_1,
        ID_Error);
```

Function Block calls in ST

- It is possible to get the value for a single Function Block output in this way:

```
P_oN2 (
   Axis := MC_Axis000,
   Enable := AA,
   Status => ok,
   Busy => NotReady,
   Error =>Err,
   ErrorID =>ErrCode);
```

```
ok:= P_oN2.Status ;
```

```
NotReady:= P_oN2.Busy ;
```

```
P_oN2(Enable:=AA,
      Axis:=MC_Axis000);
```

Function calls in ST

- A **Function** can be called “as it is“ in ST because is no needed to declare, neither use an instance name

```
<variable>:= FunctionName(<parameter_1>,…,<parameter_n>)
```

Example:

```
Production_1:=ProductionSpeed(
              Enable:=TRUE,
              SetPoint:=45,
              Diameter:=345.6);

Production_2:=ProductionSpeed(
              Enable:=TRUE,
              SetPoint:=46,
              Diameter:=220);
```

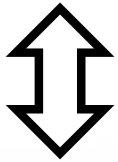
Function calls in ST

- The **unused inputs** for a Function call can be left empty or even can be removed from the function call.
- Empty parameters for a function will take default values.

Example:

```
Production_3:= ProductionSpeed(
      Enable:= ,
      Diameter:=345.6);
```

In **Sysmac Studio: remove unused I/Os**



```
Production_3:= ProductionSpeed(
      Diameter:=345.6);
```

Data Types handling in ST

- Any Type of variable, expressed as a **Structure** of data, can be accessed using " **.** " following the hierarchy for the mentioned type.

```
<variable>:= <Type> . <Type element>;
```

- A Structure containing as a member another structure, is accessed in the same way

```
<variable>:= <Type_1> . <Type_2> . <Type_2 element>;
```

Data Types handling in ST

Example:

Struct
Union
Enumerated

	Name	Base Type
▼	Motor_Data	STRUCT
	Enable	BOOL
	SetPoint	INT
	Value	INT
	Encoder	DINT
	State	Status
▼	Status	STRUCT
	StandStill	BOOL
	Alarm	BOOL
	StateCode	INT

Name	Data Type
Motor_1	Motor_Data

```
Motor_1.Enable:=TRUE;
Motor_1.Setpoint:=459;
Motor_1.Value:=0;
EncoderValue:= Motor_1.Encoder;
Motor_1.State.StandStill:=TRUE;
Motor_1.State.StateCode:=100;
```

Exercise #7

- Create a **Function block** in ST with the next layout:
  - **Execute** input, rise edge sensitive.
  - For each rising edge in Execute, the output variable **Value** increase up to count 5 and then will reset.
  - The boolean variable **Output**, will alternate ON-OFF-ON… for each Execute Edge.
  - A **Reset** input will reset both output values.

- Call it from a ST program and/or LD program

•FB definition and coding

Internals
In/Out
Externals

Name	In/Out	Data Type	Edge	Initial Value
Execute	Input	BOOL	Up	
Reset	Input	BOOL	No Edge	
output	Output	BOOL	No Edge	
value	Output	INT	No Edge	

```
IF Execute THEN

    Output:=NOT Output;
    value:=value+INT#1;
    IF value>5 THEN value:=0;END_IF;

END_IF;

IF Reset THEN

    output:=FALSE;
    value:=0;

END_IF;
```

•FB call in ST and LD

Internals / Externals

Name	Data Type
FB_Signal	Signal
X	BOOL
rst	BOOL
Counter	INT
Out_1	bool

```
FB_Signal(    Execute:=X,
              Reset:=rst,
              Value=>Counter,
              Output=>Out_1);
```

⇔

```
FB_Signal(Execute:=X);
FB_Signal(Reset:=rst);
Out_1:=FB_Signal.Output;
Counter:=FB_Signal.Value;
```

Note:
Here , FB_Signal is called once

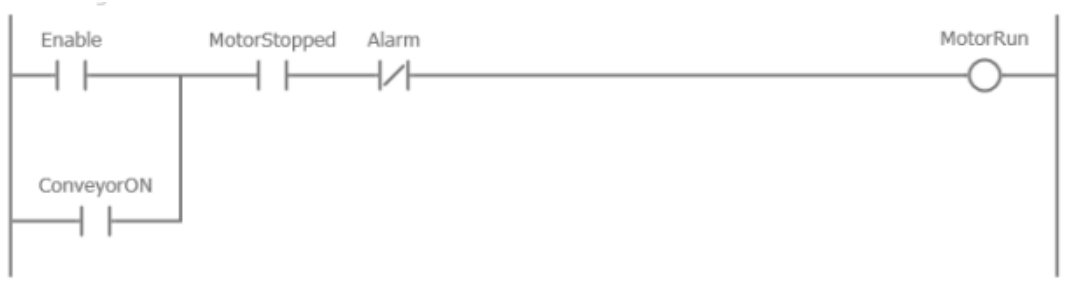
Note:
Here , FB_Signal is called two times

Comparing ST vs. LD structures

Assignment

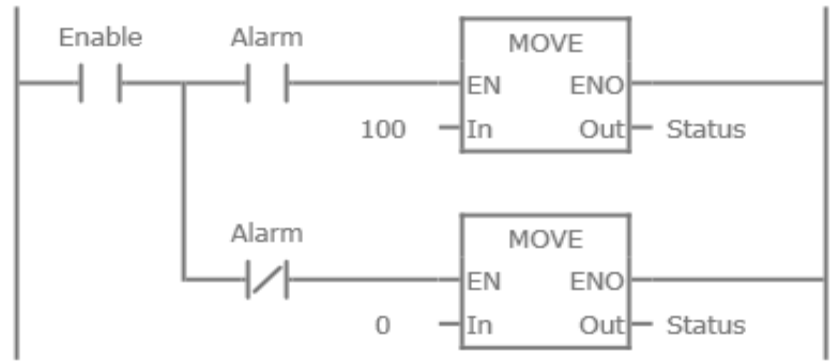

```
Starter:=TRUE;
```

Logical condition




```
MotorRun:=(Enable OR ConveyorON)
          AND MotorStopped
          AND NOT Alarm;
```

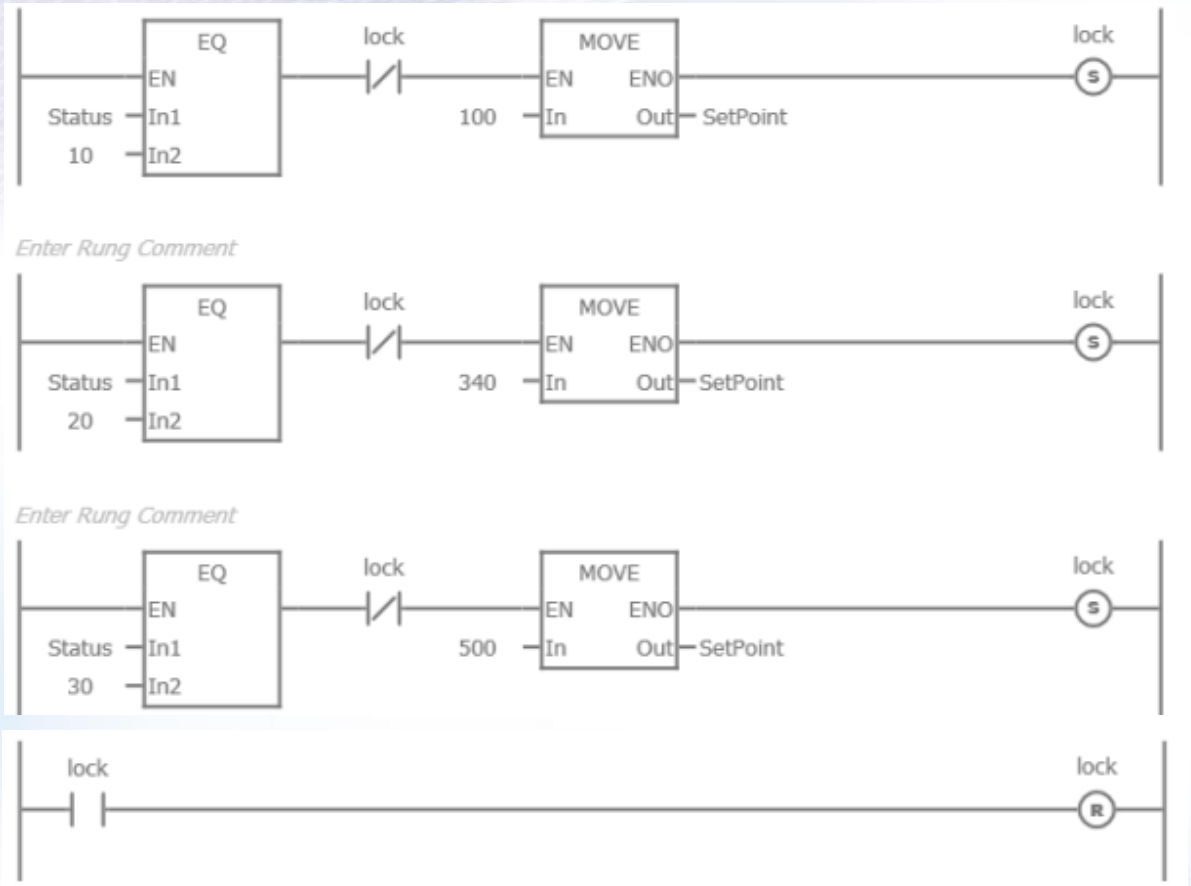
Conditional assignment




```
IF Enable THEN
    IF Alarm THEN
        Status:=100;
    ELSE
        Status:=0;
    END_IF;
END_IF;
```

Comparing ST vs. LD structures

Branching structure

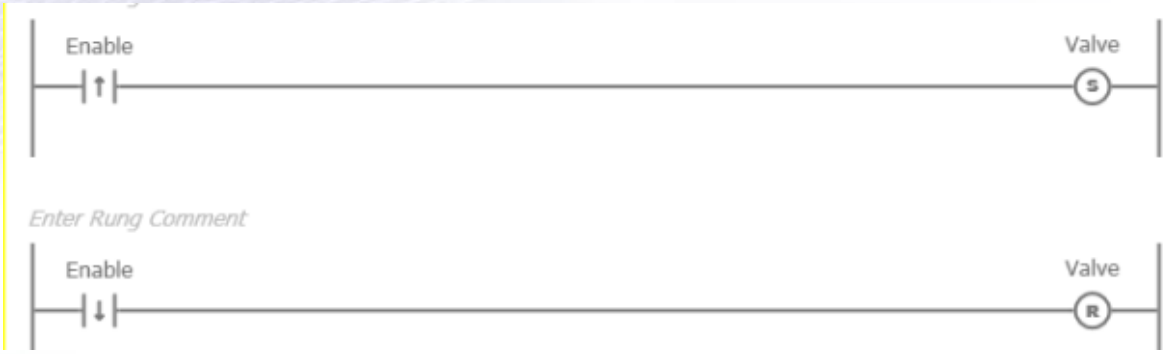



```
CASE Status OF

    10:  Setpoint:=100;

    20:  Setpoint:=340;

    30:  Setpoint:=500;

END_CASE;
```

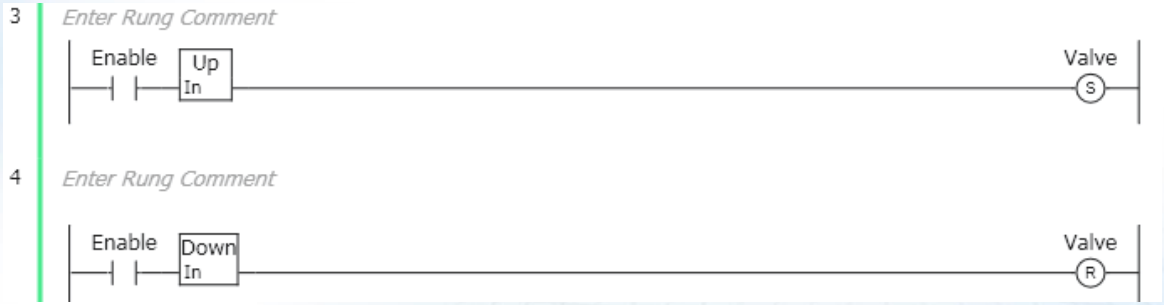
**Lock* Variable is used here to emulate the CASE behavior, it means to execute just one condition per cycle

Comparing ST vs. LD structures

Rising and Falling edge



You can use also…



FB Instance declaration:

Name	Data Type
Rising_edge	R_TRIG
Falling_edge	F_TRIG

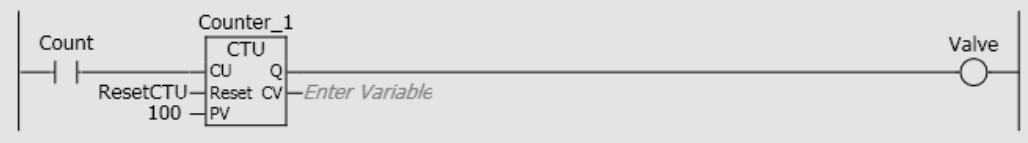
```
Rising_edge(clk:=Enable);
Falling_edge(clk:=Enable);

IF Rising_edge.q THEN
    Valve:=TRUE;
END_IF;

IF Falling_edge.q THEN
    Valve:=FALSE;
END_IF;
```

You <u>can't</u> use …(Not IEC FB, not compatible in ST)

```
Valve:=Up(Enable);

Valve:=Down(Enable);
```

Comparing ST vs. LD structures

Timers and counters



FB Instance declaration:

Name	Data Type
Timer_1	TON
Counter_1	CTU

```
Timer_1(In:=Enable,PT:=t#100ms,Q=>Valve);
```

```
Counter_1(  CU:=Count,Reset:=ResetCTU,
            PV:=100,Q=>Valve);
```