



Corso di Automazione industriale

Lezione 7

PLC – Testo strutturato

Introduzione

Il Testo strutturato è il linguaggio più di alto livello della norma IEC 61131.

Come costrutti ricorda i linguaggi di programmazione Fortran e Pascal.

Ogni costruttore personalizza il linguaggio creando un «dialetto» ad hoc (quindi non compliant rispetto ad altri ambienti di sviluppo concorrenti).

Noi ci concentriamo sul «dialetto» disponibile in AS.

Struttura base

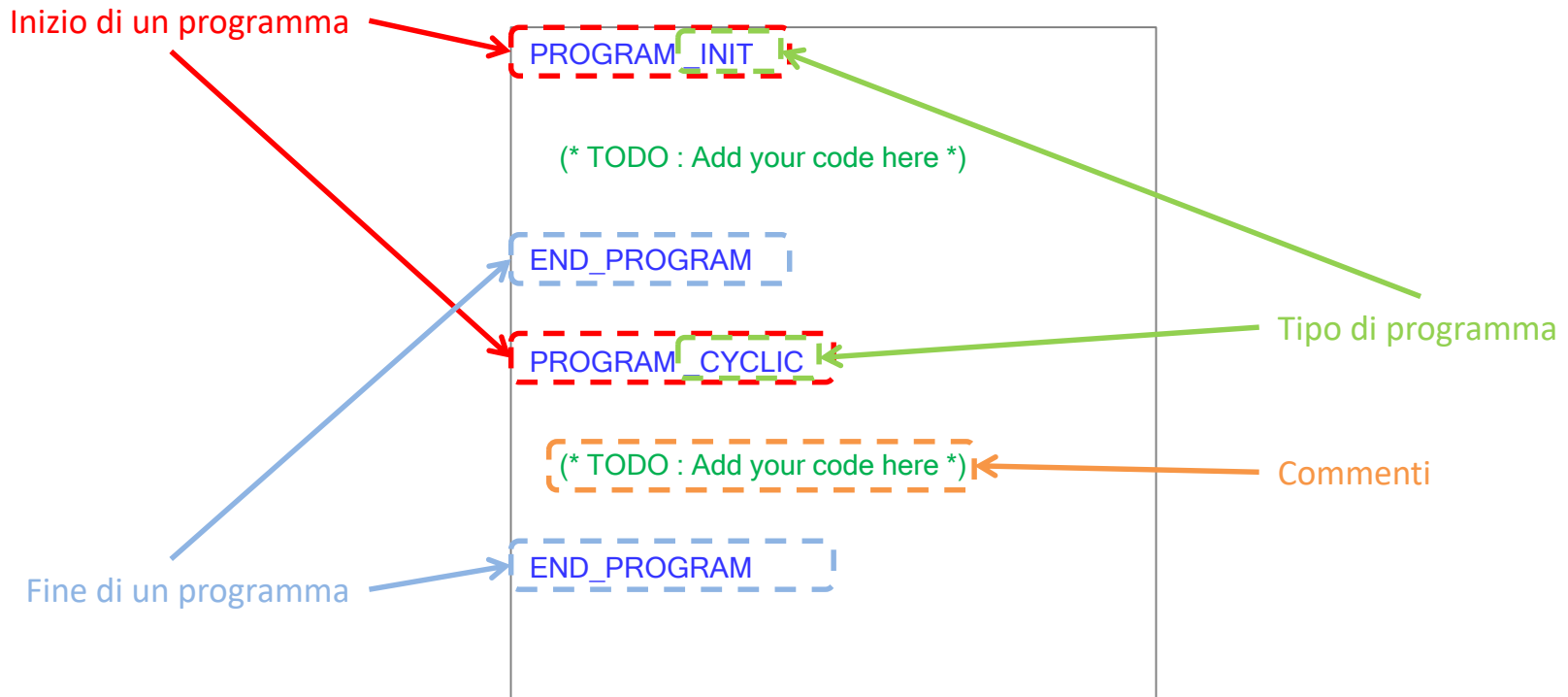
Ogni file che contiene testo strutturato è costituito da tre diversi programmi:

- Programma di inizializzazione
- Programma ciclico
- Programma di uscita

N.B.1: non è sempre strettamente necessario avere inizializzazione e uscita.

N.B.2: anche con gli altri linguaggi di programmazione IEC 61131, in AS, è possibile definire queste tre tipologie di esecuzione.

Struttura base



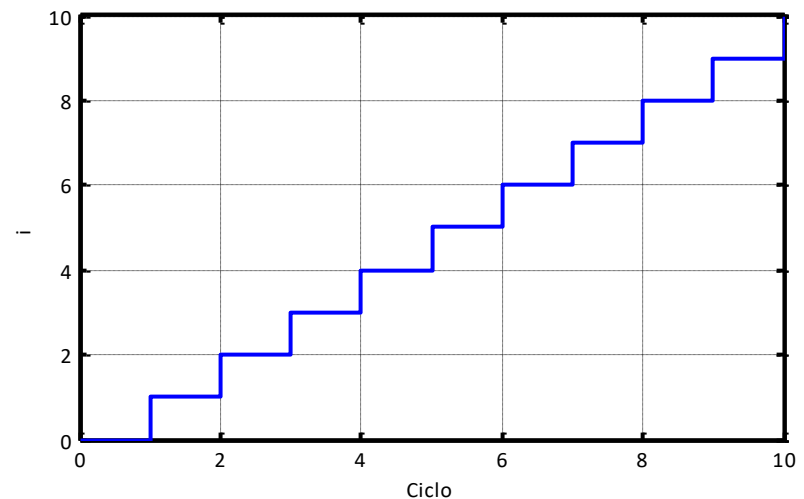
Struttura base

N.B.: l'esecuzione del programma `_INIT` avviene all'accensione del PLC, l'esecuzione del programma `_CYCLIC` avviene con cadenza definita dalla schedulazione del Task nel PLC. Ciò significa che tutte le istruzioni contenute nel programma `_CYCLIC` vengono eseguite ad ogni ciclo!

Esempio

```
PROGRAM_INIT
  i := 0;
END_PROGRAM

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



Parole chiave

Keyword	Significato
ACCESS	Accesso ad una variabile dinamica (puntatore)
BIT_CLR	A = BIT_CLR(IN, POS) A contiene il valore di IN con il bit in posizione POS cancellato (impostato a 0)
BIT_SET	A = BIT_SET(IN, POS) A contiene il valore di IN con il bit in posizione POS settato (impostato ad 1)
BIT_TST	Determina il valore di un bit: A := BIT_TST(IN, POS) A contiene il bit alla posizione POS di IN
EDGE	Identifica tutti i fronti dell'ingresso
EDGENEG	Identifica tutti i fronti negativi dell'ingresso
EDGEPOS	Identifica tutti i fronti positivi dell'ingresso

Operatori

Operatore	Significato
ABS	Valore assoluto
ACOS	Coseno inverso
ADR	Indirizzo della variabile
AND	AND bit a bit
ASIN	Seno inverso
ASR	Shift a destra di N bit: $A := ASR(IN, N)$; IN è shiftato di N bit a destra, la parte sinistra è riempita con il bit di segno
ATAN	Tangente inversa
COS	Coseno
EXP	Esponenziale

Operatori

Operatore	Significato
EXPT	Elevamento a potenza: $A := \text{EXPT}(\text{IN1}, \text{IN2}); A = \text{IN1}^{\text{IN2}}$
LIMIT	Limitazione: $A = \text{LIMIT}(\text{MIN}, \text{IN}, \text{MAX});$ MIN è il limite inferiore, MAX è il limite superiore. Se IN è minore di MIN, viene restituito MIN. Se IN è maggiore di MAX, viene restituito MAX. Altrimenti, viene restituito IN.
LN	Logaritmo naturale
LOG	Logaritmo in base 10
MAX	Massimo tra due numeri
MIN	Minimo tra due numeri
MOD	Resto della divisione tra variabili di tipo USINT, SINT, INT, UINT, UDINT, DINT
MOVE	Assegnamento; "A := B;" equivale a "A := MOVE (B);"

Operatori

Operatore	Significato
MUX	Selezione: $A = \text{MUX}(\text{CHOICE}, \text{IN1}, \text{IN2}, \dots, \text{INX})$ CHOICE definisce quale delle variabili IN1, IN2, ... INX è assegnata ad A
NOT	Not bit a bit
OR	Or bit a bit
ROL	Rotazione bit a bit verso sinistra: $A := \text{ROL}(\text{IN}, N)$; IN è shiftato N volte verso sinistra un bit alla volta, il bit più a sinistra viene spostato a destra
ROR	Rotazione bit a bit verso destra: $A := \text{ROR}(\text{IN}, N)$; IN è shiftato N volte verso destra un bit alla volta, il bit più a destra viene spostato a sinistra
SEL	Selezione binaria: $A := \text{SEL}(\text{CHOICE}, \text{IN1}, \text{IN2})$; CHOICE deve essere di tipo BOOL. Se CHOICE è FALSE, allora viene restituito IN1. Altrimenti, viene restituito IN2

Operatori

Operatore	Significato
SHL	Shift bit a bit verso sinistra: $A := \text{SHL}(\text{IN}, N)$; IN è shiftato di N bit verso sinistra, la parte destra è riempita con zeri
SHR	Shift bit a bit verso sinistra: $A := \text{SHR}(\text{IN}, N)$; IN è shiftato di N bit verso destra, la parte sinistra è riempita con zeri
SIN	Seno
SIZEOF	Restituisce il numero di byte della variabile (o del tipo)
SQRT	Radice quadrata
TAN	Tangente
TRUNC	Restituisce la parte intera di un numero
XOR	XOR bit a bit

Strutture

IF – THEN – ELSE

Attenzione!!!



```
IF <espressione1> THEN
  <lista_istruzioni1>
  ELSEIF <espressione2> THEN
    <lista_istruzioni2>
    .
    .
  ELSE
    <lista_istruzioniN>
END_IF;
```

Strutture

IF – THEN – ELSE

```
PROGRAM_INIT
  t := 0;
  Out := 8;
END_PROGRAM

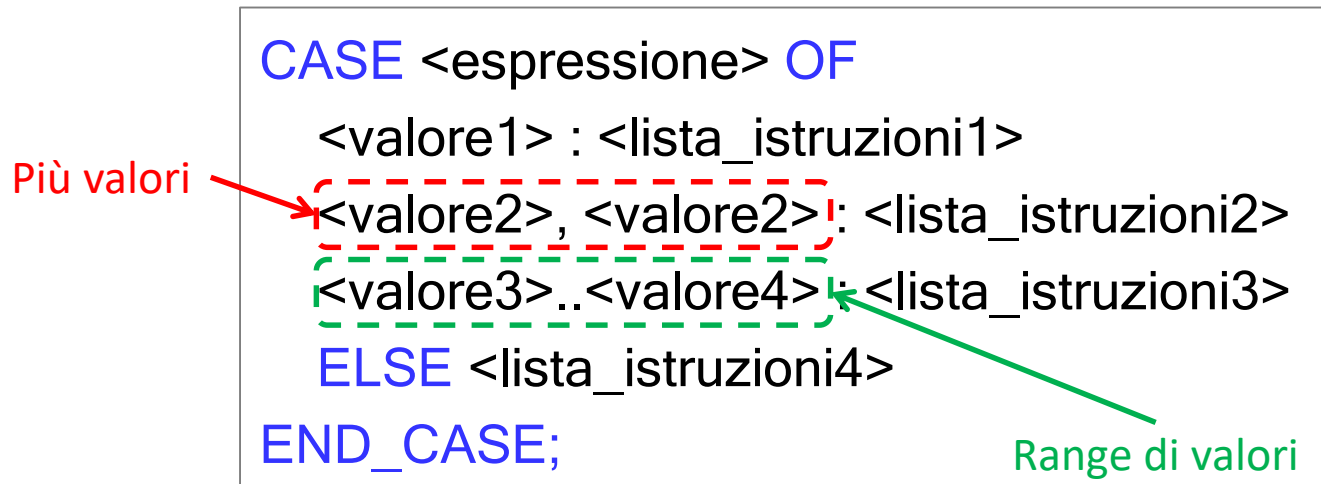
PROGRAM_CYCLIC
  IF t < 4 THEN
    t := t + 1;
  END_IF;
  IF t < 2 THEN
    Out := 0;
  ELSIF t < 2 THEN
    Out := 1;
  ELSIF t > 3 THEN
    Out := 2;
  ELSE
    Out := 3;
  END_IF;
END_PROGRAM
```

Risultati:

Ciclo di esecuzione	Out
0	8
1	0
2	3
3	3
4	2

Strutture

CASE



Strutture

CASE

```
PROGRAM_INIT
  t := 0;
  Out := 8;
END_PROGRAM

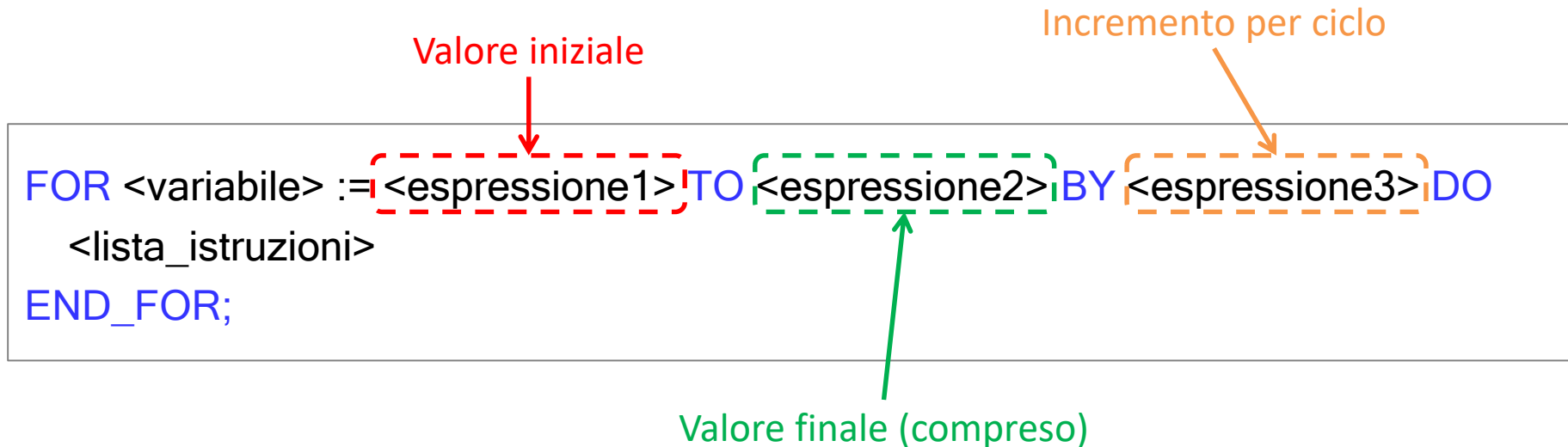
PROGRAM_CYCLIC
  IF t < 4 THEN
    t := t + 1;
  END_IF;
  CASE t OF
    1 : Out := 6;
    0, 2 : Out := 1;
    3..4 : Out := 2;
  ELSE Out := 4;
  END_CASE;
END_PROGRAM
```

Risultati:

Ciclo di esecuzione	Out
0	8
1	6
2	1
3	2
4	2

Strutture

FOR



N.B.: Se l'incremento porta <variabile> ad un valore maggiore di <espressione2> si stoppa il ciclo

Strutture

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
```

Cosa calcola questo codice?

Il fattoriale del numero A e lo assegna alla variabile U

Strutture

REPEAT (ciclo con controllo in coda)

```
REPEAT
  <lista_istruzioni>
UNTIL
  <espressione>
END_REPEAT;
```

Attenzione!!!

Se <espressione> è
falsa si esegue il repeat,
altrimenti si esce.

L'opposto del while in C

Strutture

REPEAT (ciclo con controllo in coda)

```
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
```

Cosa calcola questo codice?

Il valore espresso in binario della variabile A e lo mette nel vettore U

Strutture

WHILE (ciclo con controllo in testa)

```
WHILE <espressione> DO  
  <lista_istruzioni>  
END_WHILE;
```

Attenzione!!!
Se <espressione> è vera
si esegue il while,
altrimenti si esce.
Come il while in C

N.B.: Tralasciamo un esempio di questa istruzione: è equivalente a quella del C

Istruzioni aggiuntive

EXIT: equivale al break del linguaggio C, blocca l'esecuzione del ciclo in cui è inserita

RETURN: equivale al return del linguaggio C, blocca l'esecuzione della funzione, del function block o del programma in cui è inserita

Istruzioni aggiuntive

EXIT

```
PROGRAM_CYCLIC
U := 0;
FOR A:=0 TO 1 BY 1 DO
  FOR B:=0 TO 10 BY 1 DO
    IF B>=6 THEN
      EXIT;
    ELSE
      U := U + 1;
    END_IF;
  END_FOR
  U := U + 1;
END_FOR
END_PROGRAM
```

Quanto vale U alla fine dell'esecuzione?

Contiamo i cicli:

A=0, B=0 U = 1

...

A=0, B=5 U = 6

A=0, B=5 U = 7 ← Istruzione fuori dal primo for

A=1, B=0 U = 8

...

A=1, B=5 U = 13

A=1, B=5 U = 14 ← Istruzione fuori dal primo for

U vale 14

Function Block

Il Function Block è una porzione di codice di controllo con definite della variabili di interfaccia verso l'esterno (input, interne e output).

Un Function Block è costituito da un file st che contiene il software in testo strutturato.

N.B.: Ogni programma contiene anche un file fun con tutte le interfacce dei blocchi, delle funzioni, ecc...

Function Block

Esempio

Realizzare un FB che calcoli $U = A \cap \bar{B}$

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

```
(* File Esempio.st *)  
PROGRAM _INIT  
END_PROGRAM  
  
PROGRAM _CYCLIC  
    ExampleFB(A := In1, B := In2);  
    Out := ExampleFB.U;  
END_PROGRAM
```

N.B.: E' necessario dichiarare nella variabili del programma Esempio.st la variabile ExampleFB di tipo EsempioFB!!