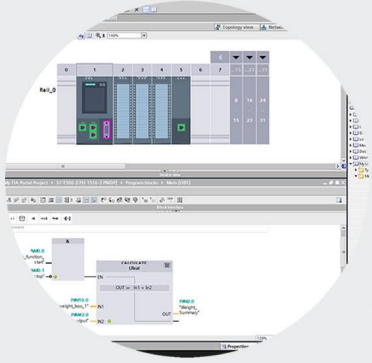
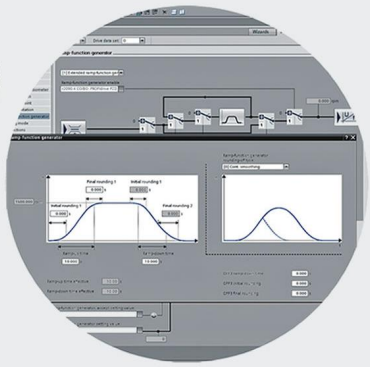
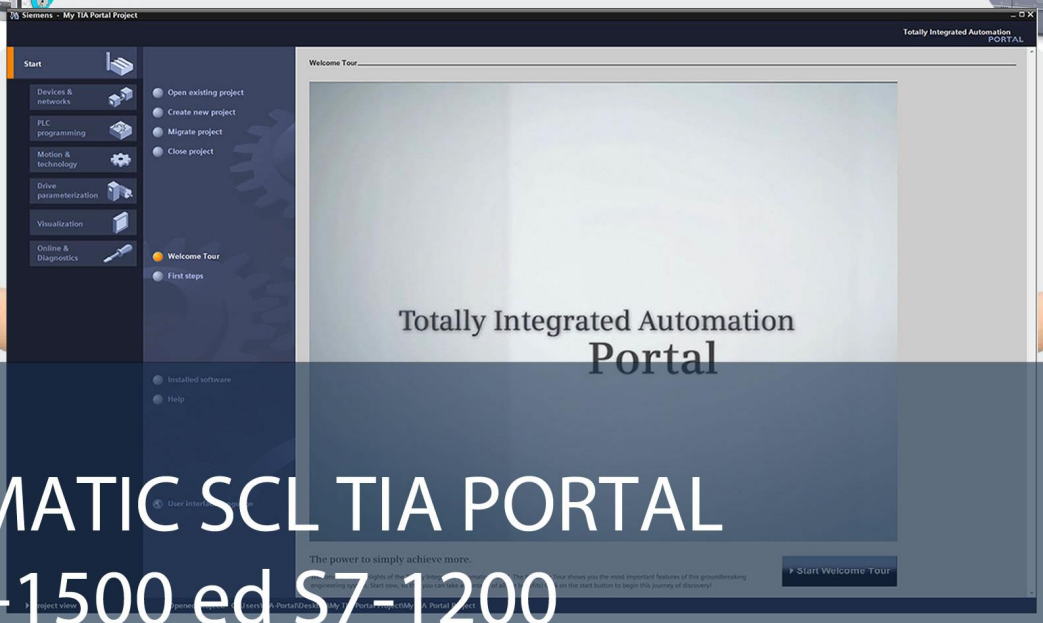
SIEMENS







SIMATIC SCL TIA PORTAL S7-1500 ed S7-1200

Software TIA Portal: Step7 V13 SP1

SIEMENS

SIMATIC SCL TIA PORTAL

S7-1500 ed S7-1200

Guida alla programmazione SCL

utilizzando software TIA Portal: Step7 V13 SP1

SIEMENS

Le informazioni riportate in questo manuale tecnico contengono descrizioni o caratteristiche che potrebbero variare con l'evolversi dei prodotti e non essere sempre appropriate, nella forma descritta, per il caso applicativo concreto.
Con riserva di modifiche tecniche.

Tutte le denominazioni dei prodotti possono essere marchi oppure denominazioni di prodotti della Siemens AG o di altre ditte fornitrici, il cui utilizzo da parte di terzi per propri scopi può violare il diritto dei proprietari.

Le informazioni fornite in questo documento devono intendersi sempre a titolo orientativo, da sottoporre all'attenzione di chi ha la responsabilità di certificare la macchina o l'impianto e non costituiscono in nessun caso vincolo o responsabilità specifiche.

SIEMENS

Indice

1. Introduzione

Questo documento è una guida per i primi passi alla programmazione tramite il linguaggio SCL di blocchi di PLC della serie S7-1200 ed S7-1500, avvalendosi dei software TIA Portal: Step V13 SP1.
L'SCL è un linguaggio di programmazione simile al PASCAL che segue lo standard DIN 61131-3 che standardizza i linguaggi di programmazione destinati ai controllori programmabili. Seguendo uno standard internazionale è molto simile ai linguaggi strutturati di altri costruttori che seguono la stessa norma.
L'SCL, essendo presente sia sulle CPU della serie S7-1200 sia su CPU della serie S7-1500 è, insieme al KOP e al FUP, un linguaggio che permette di scrivere del codice riutilizzabile su tipologie di controllori diversi.
E' inoltre un linguaggio molto comodo quando si hanno algoritmi complessi, caratterizzati da una grande quantità di dati da gestire e con molte operazioni cicliche da effettuare.

Durante la presentazione dei vari argomenti, verranno presentati esempi per meglio comprendere le possibilità di utilizzo di questo linguaggio di programmazione. Per approfondimenti su particolari istruzioni, si rimanda all'help online del TIA Portal o ai manuali dei PLC.

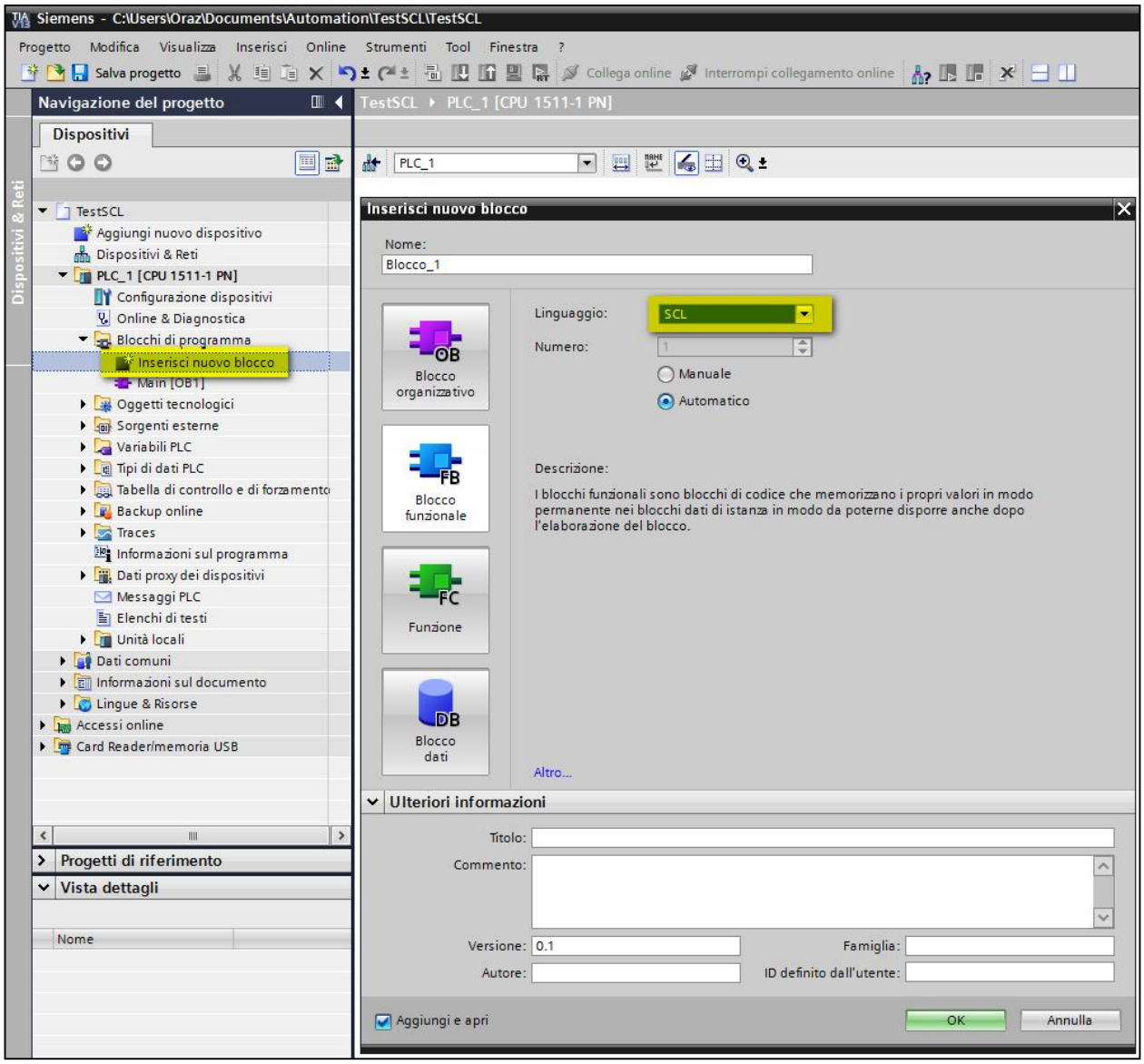
Per seguire senza difficoltà questo documento è richiesta la conoscenza dei PLC della serie Simatic S7-1200 ed S7-1500 e del software Simatic STEP7 V13 (TIA).

2. Inserimento di un blocco SCL

La creazione di un blocco SCL in TIA Portal è possibile seguendo due strade diverse in base a due differenti esigenze: creare un nuovo blocco SCL all'interno del quale sviluppare il codice, oppure generare un blocco già scritto su file sorgente esterno di tipo testuale.

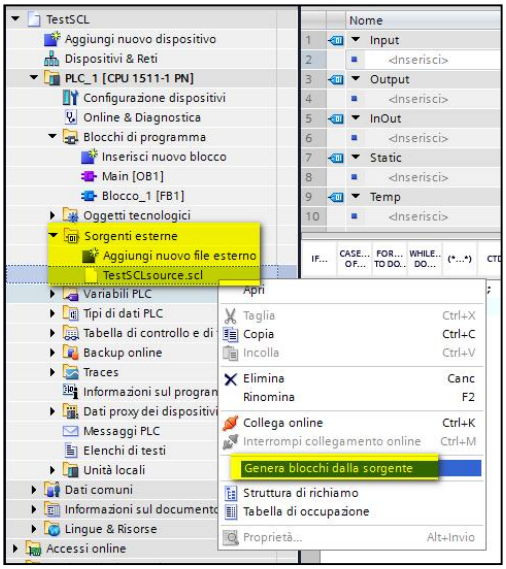
2.1. Creazione di un nuovo blocco SCL

Per inserire un nuovo blocco (FB, FC o OB) vuoto all'interno del quale sviluppare il proprio codice SCL, è sufficiente, al momento dell'inserimento del blocco, scegliere il tipo di linguaggio SCL.



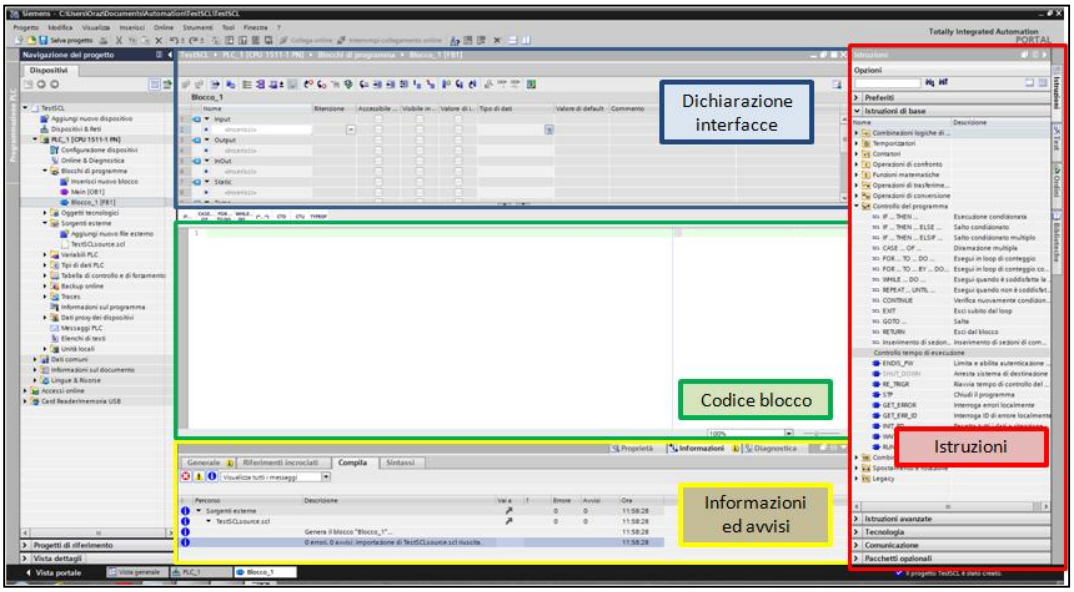
2.2. Generazione di un blocco SCL da file sorgente

Per generare un nuovo blocco (FB, FC o OB) a partire da un file sorgente con estensione '*.scl*' è necessario (dopo aver inserito il file nel progetto tramite la voce '*Aggiungi nuovo file esterno*' nella cartella '*Sorgenti esterne*'), cliccare col pulsante destro sul file sorgente e scegliere la voce '*Genera blocchi dalla sorgente*'.



3. Editor del blocco

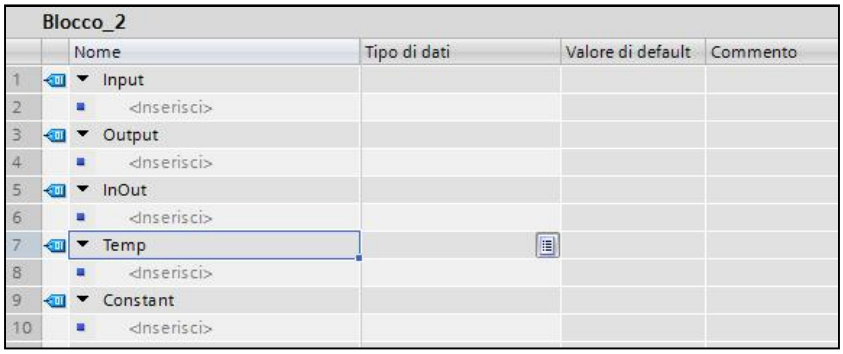
Con TIA Portal l'editor del blocco SCL è simile a quello che troviamo negli altri linguaggi di programmazione in modo da non dover imparare a muoversi in un editor completamente diverso.



- In alto abbiamo la finestra di '*Dichiarazione delle interfacce*' in forma tabellare equivalente a quella che troviamo in qualsiasi altro linguaggio di programmazione;
- A destra troviamo la finestra delle istruzioni da cui è possibile, per drag&drop, trascinare nella finestra del codice i comandi da utilizzare;
- Al centro è presente la finestra nella quale vengono scritte le istruzioni che vogliamo far eseguire al blocco;
- Nella finestra più in basso invece vengono riportati avvisi e informazioni sulla correttezza della sintassi utilizzata e su eventuali errori commessi.

4. Dichiarazione delle variabili di interfaccia

Come in tutti gli altri linguaggi, la dichiarazione delle variabili di interfaccia di un blocco avviene in un editor tabellare. I tipi di dati dichiarabili sono identici a quelli negli altri linguaggi.



5. Accesso a variabili e aree di memoria

5.1. Variabili locali

La sintassi per l'accesso alle variabili locali di un blocco prevede l'aggiunta del carattere '#' in testa al nome della variabile stessa. Se si utilizza l'opzione di auto-completamento della variabile, in automatico il sistema inserisce il carattere necessario.

5.2. Variabili globali

La sintassi per l'accesso alle variabili globali (merker, ingressi, uscite, timer, counter e DB) prevede l'inserimento del nome della variabile (nel caso di merker, ingressi e uscite) o del nome della DB (nel caso di accesso a DB) tra doppi apici. Se si utilizza l'opzione di auto-completamento della variabile, in automatico il sistema inserisce il carattere necessario.

```
1
2  "DBvariabili".Velocità := 50;
3
4  "VariabileGlobaleMerker" := 40;
5
```

5.3. Indicizzazione di variabili (simbolica)

L'indicizzazione simbolica delle variabili, presuppone che queste siano state dichiarate come array, che è l'unico elemento che può essere indicizzato simbolicamente. Se ci troviamo in questa condizione possiamo indicizzare l'elemento dell'array utilizzando una qualsiasi variabile INT o DINT come indice, come si vede dalla figura seguente.

	Name	Data type	Default v
1	Input		
2	Indice	Int	
3	NumMotore	DInt	
4	Output		
5	InOut		
6	Temp		
7	DatiDaIndicizzare	Array[0..10] of Int	
8	DatiMotore	Array[0..100] of Struct	
9	DatiMotore[0]	Struct	
10	Velocità	Real	
11	Posizione	Real	
12	DatiMotore[1]	Struct	
13	DatiMotore[2]	Struct	

```
1
2  #DatiDaIndicizzare[#Indice] := 600;
3
4  #DatiMotore[#NumMotore].Velocità := 1000.0;
5
```

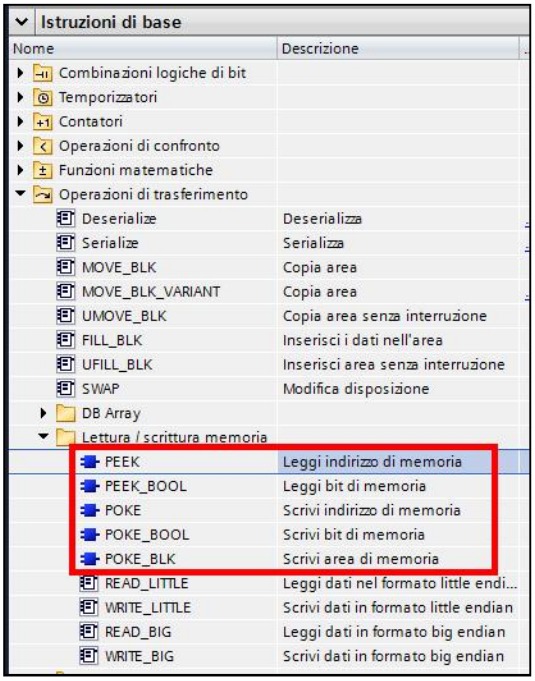
In questo esempio, in base al valore della variabile '*#Indice*' e della variabile '*#NumMotore*', il codice andrà ad eseguire le istruzioni di assegnamento su variabili diverse.

Gli indici utilizzati negli array possono anche essere delle operazioni come si vede nell'immagine seguente.

```
6
7  #DatiDaIndicizzare[#Indice+3] := 600;
8
9  #DatiMotore[#NumMotore+1].Velocità := 1000.0;
10
```

5.4. Indicizzazione di aree di memoria (assoluta)

L'indicizzazione assoluta viene fatta, su qualsiasi area di memoria (ingressi, uscite, merker e DB assolute), utilizzando dei blocchi già pronti di lettura (*PEEK*) e scrittura (*POKE*) che si trovano tra le istruzioni di base nella cartella '*operazioni di trasferimento*'.



L'istruzione di *PEEK* serve per leggere aree di memoria che variano in base a 4 variabili di ingresso che indicano il tipo di area, il numero della DB (se si punta a una DB), il numero del byte e il numero del bit. Al variare del valore di queste variabili, cambia l'area di memoria letta. Di seguito esempi di letture di aree di diverso tipo.

	Nome	Tipo di dati	Valore di default	Commento
1	Input			
2	TipoDiArea	Byte		
3	NumDB	DInt		
4	NumByte	DInt		
5	NumBit	Int		
6	Output			
7	BitLetto	Bool		
8	ByteLetto	Byte		
9	WordLetta	Word		
10	DwordLetta	DWord		
11	InOut			

```
1  //Esempio di lettura indicizzata di un bit
2  #BitLetto:= PEEK_BOOL(area:=#TipoDiArea, dbNumber:=#NumDB, byteOffset:=#NumByte, bitOffset:=#NumBit);
3
4  //Esempio di lettura indicizzata di un byte
5  #ByteLetto := PEEK_BYTE(area := #TipoDiArea, dbNumber := #NumDB, byteOffset := #NumByte);
6
7  //Esempio di lettura indicizzata di una word
8  #WordLetta := PEEK_WORD(area := #TipoDiArea, dbNumber := #NumDB, byteOffset := #NumByte);
9
10 //Esempio di lettura indicizzata di una Dword
11 #DwordLetta := PEEK_DWORD(area := #TipoDiArea, dbNumber := #NumDB, byteOffset := #NumByte);
```

L'istruzione di *POKE* serve per scrivere aree di memoria che variano in base a 4 variabili di ingresso che indicano il tipo di area, il numero della DB (se si punta a una DB), il numero del byte e il numero del bit. Al variare del valore di queste variabili, cambia l'area di memoria letta. La quinta variabile passata in ingresso, contiene il valore da scrivere nell'area di memoria puntata dal *POKE*. Di seguito esempi di scritture di aree di diverso tipo.

```
13 //Esempio di scrittura indicizzata di un bit
14 POKE_BOOL(area := #TipoDiArea, dbNumber := #NumDB, byteOffset := #NumByte, bitOffset := #NumBit,
15          value:=#BitScritto);
16
17 //Esempio di scrittura indicizzata di un byte
18 POKE(area := #TipoDiArea, dbNumber := #NumDB, byteOffset := #NumByte, value:=#ByteScritto);
19
20 //Esempio di scrittura indicizzata di una word
21 POKE(area := #TipoDiArea, dbNumber := #NumDB, byteOffset := #NumByte, value:=#WordScritta);
22
23 //Esempio di scrittura indicizzata di una Dword
24 POKE(area := #TipoDiArea, dbNumber := #NumDB, byteOffset := #NumByte, value:=#DwordScritta);
```

E' possibile inoltre indicizzare sia l'area di lettura che quella di scrittura con il blocco *POKE_BLK*. Con questo blocco è inoltre possibile leggere e scrivere in modo indicizzato aree maggiori di 4 byte. Di seguito un esempio di copia di 10 byte su aree indicizzate:

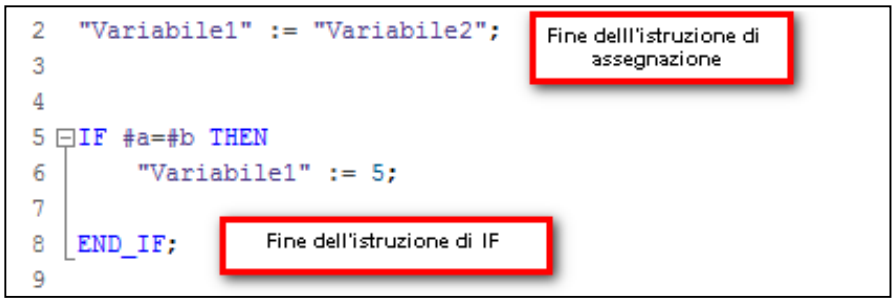
```
26 POKE_BLK(area_src:=#TipoDiAreaSorgente,
27          dbNumber_src:=#NumDBSorgente,
28          byteOffset_src:=#NumByteSorgente,
29          area_dest:=#TipoDiAreaDestinazione,
30          dbNumber_dest:=#NumDBDestinazione,
31          byteOffset_dest:=#NumByteDestinazione,
32          count:=10);
```

6. Istruzioni

Nella finestra di codice, a differenza dell'SCL dello Step7 Classico, è necessario scrivere esclusivamente le istruzioni che il PLC deve eseguire, esattamente come negli altri linguaggi (non è più necessario scrivere codice per dichiarare l'inizio e la fine del blocco, per dichiarare variabili, per dichiarare gli attributi ecc...). In questo capitolo vediamo qual è la sintassi per le principali istruzioni dell'SCL, classificate per tipo.

6.1. Chiusura di istruzioni

Per dichiarare la fine di un'istruzione è necessario utilizzare il carattere ';' come si vede dall'immagine seguente.

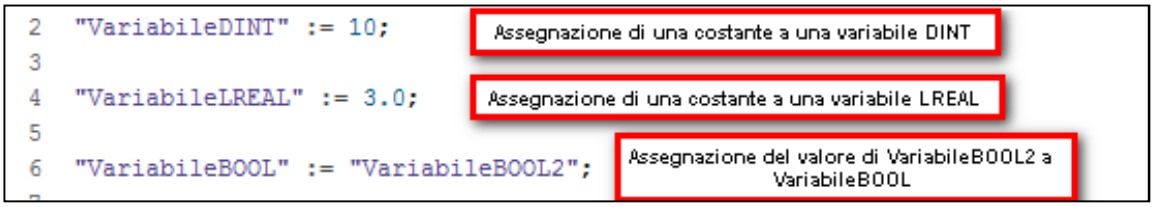


6.2. Istruzione di assegnazione

Per assegnare ad una variabile il valore di una costante o il valore di un'altra variabile, è necessario utilizzare l'istruzione ':='. Quest'istruzione può essere utilizzata con qualsiasi tipo di dato purché i due oggetti dell'assegnazione siano dello stesso tipo.

6.2.1. Assegnazione di variabili semplici

Ad una variabile semplice può essere assegnata una costante oppure il valore di un'altra variabile, purché questa sia dello stesso tipo della variabile di destinazione.

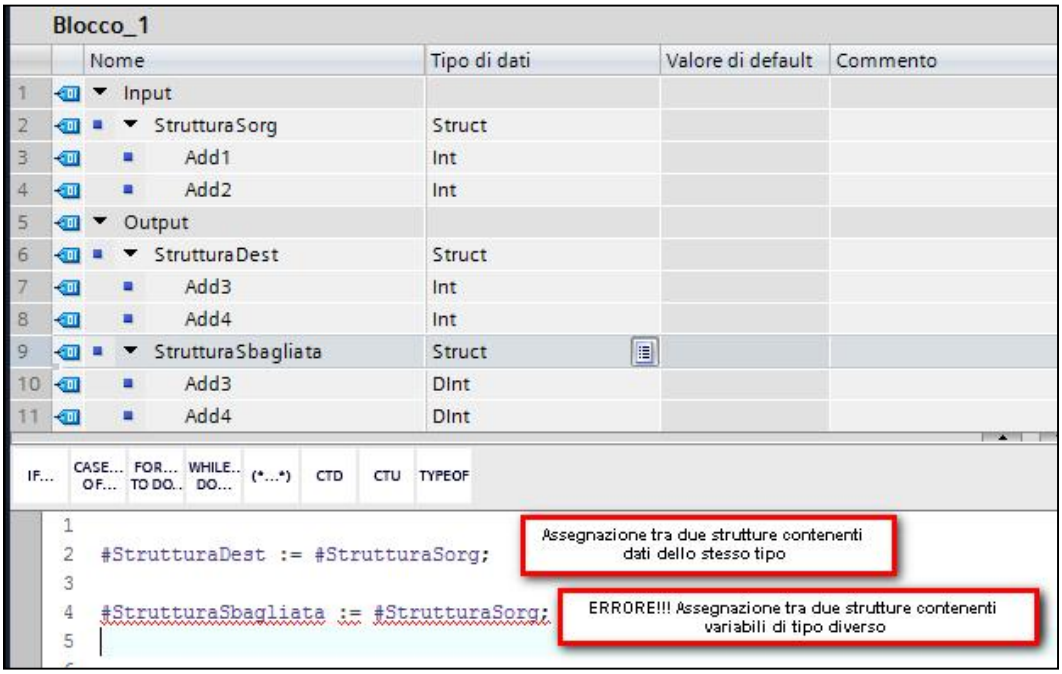


6.2.2. Assegnazione di STRUCT

Le condizioni per poter fare un'assegnazione tra due strutture sono:

- Le strutture devono contenere variabili con lo stesso tipo di dati;
- Non è importante che le variabili all'interno delle strutture si chiamino nello stesso modo.
-

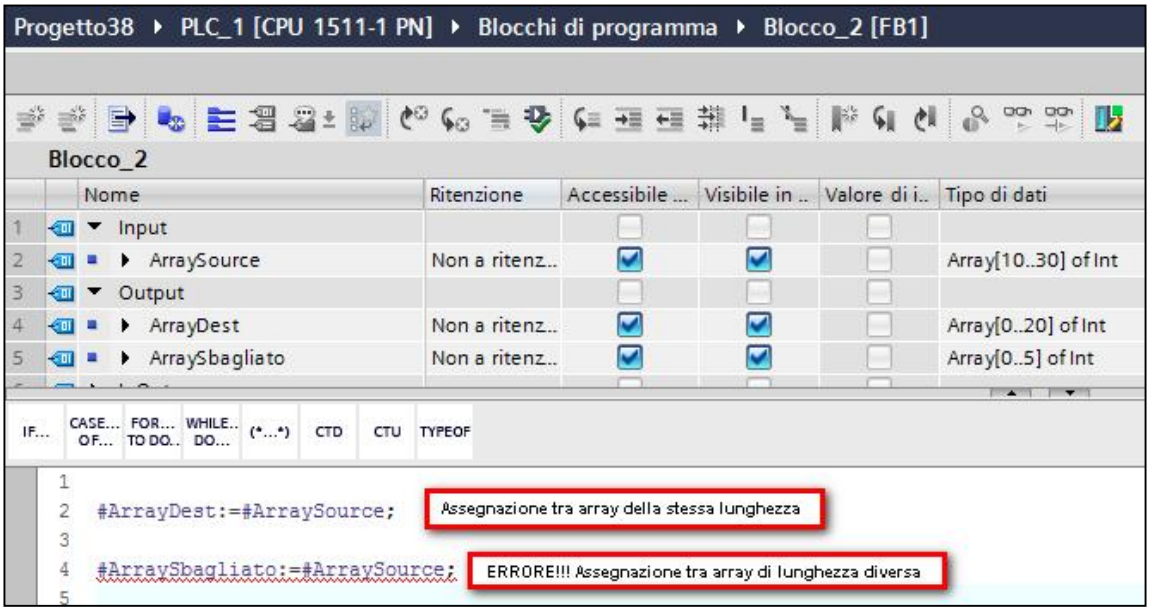
Di seguito un esempio di assegnazione corretta e uno di assegnazione sbagliata, segnalata dal compilatore dell'SCL.



6.2.3. Assegnazione di ARRAY

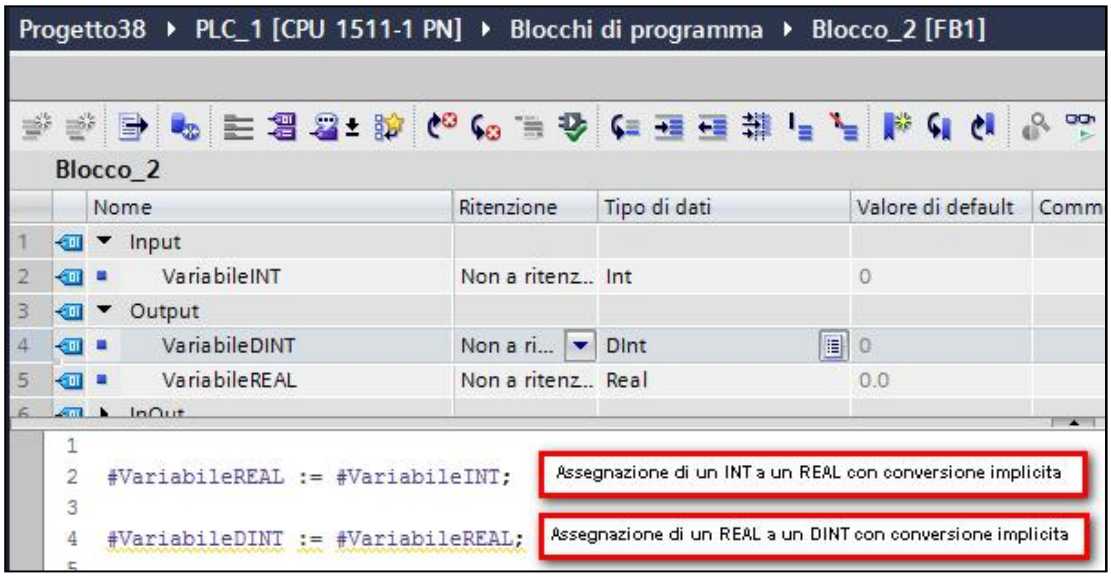
Le condizioni per poter fare un'assegnazione tra due array sono:

- Gli array devono avere elementi con lo stesso tipo di dati;
- Gli array devono avere la stessa quantità di elementi.



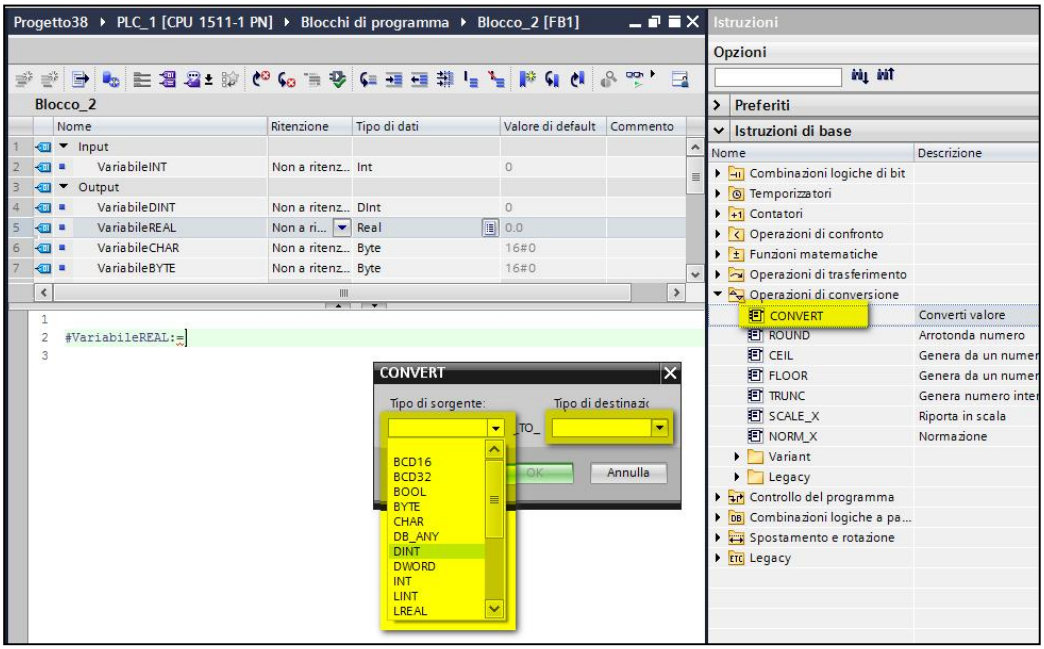
6.3. Istruzioni di conversione del tipo di dato

In SCL, con i tipi di dato semplice, esiste la possibilità di assegnare il valore di una variabile di un determinato tipo di dato ad una di un tipo di dato diverso, sfruttando la conversione implicita come rappresentato di seguito.



Come si vede dall'immagine, se la conversione può portare a perdite di segno o di precisione, viene segnalato dal TIA Portal con un avviso.

Se si volesse gestire la conversione in maniera esplicita, è possibile utilizzare l'istruzione '*Convert*' (*Istruzioni di base - Operazioni di conversione*) richiamandola per drag&drop all'interno della finestra di codice.



Si aprirà un wizard grafico per scegliere il tipo di dato sorgente e quello di destinazione che permetterà di inserire automaticamente la parola chiave corretta.

```
2  #VariabileREAL := INT_TO_REAL(#VariabileINT);
3
4  #VariabileDINT := REAL_TO_DINT(#VariabileREAL);
```

Se la conversione necessaria non è presente tra quelle disponibili, è possibile utilizzarne più di una in serie.

```
7  #DatoREAL:=INT_TO_REAL(BYTE_TO_INT(#DatoBYTE));
```

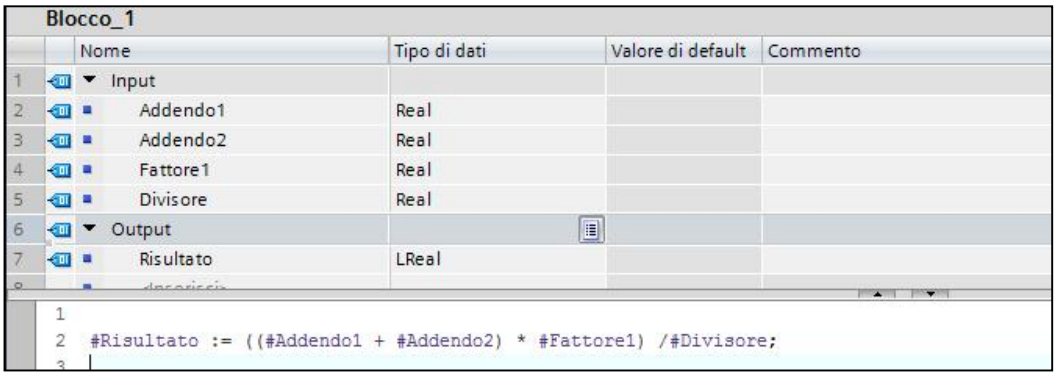
6.4. Espressioni aritmetiche

Le espressioni aritmetiche collegano due valori o due espressioni con operatori aritmetici. Di seguito gli operatori che possono essere utilizzati all'interno di espressioni matematiche:

Operazione	Operatore
Somma	+
Sottrazione	-
Moltiplicazione	*
Divisione	/
Potenza	**
Modulo	*MOD*
Parentesi	*( )*
Valore assoluto	*ABS*
Genera quadrato	*SQR*
Genera radice quadrata	*SQRT*
Logaritmo naturale	*LN*

Genera valore esponenziale	*EXP*
Seno	*SIN*
Coseno	*COS*
Tangente	*TAN*
Arcoseno	*ASIN*
Arcocoseno	*ACOS*
Arcotangente	*ATAN*

Gli operandi semplici presenti nella tabella, possono essere utilizzati anche con tipi di dato di tempo o di data e ora. Nella figura seguente un esempio di espressione aritmetica tra variabili reali con relativa assegnazione del risultato:



6.5. Espressioni logiche

Le espressioni logiche collegano due operandi tramite operatori logici oppure negano gli operandi. Di seguito gli operatori utilizzabili:

Operazione	**Operatore**
Negazione	*NOT*
Combinazione logica AND	*AND o &*
Combinazione logica OR	*OR*
Combinazione logica OR esclusivo	*XOR*

Possono essere utilizzati, oltre che operandi booleani, anche *BYTE, WORD, DWORD e LWORD.*

6.6. Espressioni di confronto

Le espressioni di confronto confrontano i valori di due operandi e forniscono un valore booleano. Il risultato è *TRUE* se il confronto è soddisfatto, oppure *FALSE* se il confronto non è soddisfatto. Gli operatori utilizzabili sono i seguenti:

Operazione	**Operatore**
Uguale	=

Diverso	<>
Maggiore-uguale	>=
Minore-uguale	<=
Maggiore	>
Minore	<

Possono essere utilizzati, oltre che con numeri, anche con tipi di dati di tempo o di data e ora.

6.7. Istruzioni di controllo di programma

La stesura di algoritmi SCL, anche complessi, si servono di istruzioni tipiche di linguaggi di alto livello (come PASCAL), che possono essere classificate in 3 gruppi:

- Istruzioni di selezione dove il PLC deve seguire una strada piuttosto che un'altra:
  - IF
  - CASE
- Istruzioni di elaborazione di loop con cui il programmatore chiede al PLC di eseguire più volte le stesse righe di codice
  - FOR
  - WHILE
  - REPEAT
- Istruzioni di salto con cui viene comunicato al PLC di saltare delle righe di codice:
  - CONTINUE
  - EXIT
  - GOTO
  - RETURN

6.7.1. Istruzione IF

L'istruzione *IF* consente di programmare diramazioni, costringendo il PLC a scegliere una strada piuttosto che un'altra, in base al valore di una condizione booleana. Di seguito la struttura dell'istruzione *IF*:

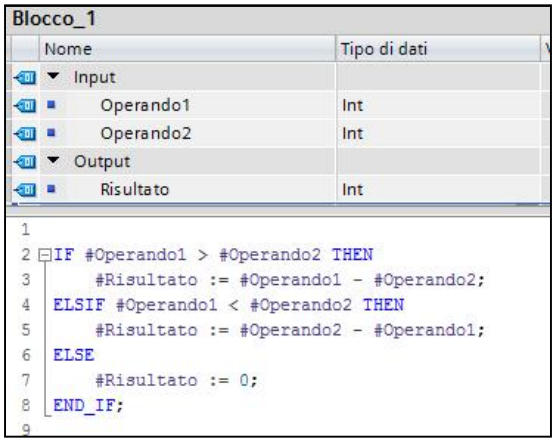
```
IF espressione THEN
    operazione;
ELSIF espressione THEN
    operazione;
ELSE
    operazione;
END_IF;
```

Le regole principali dell'istruzione *IF* sono le seguenti:

- La condizione è un'espressione con valore *TRUE* o *FALSE,* quindi deve essere definita da una variabile booleana, da un'espressione logica o un'espressione di confronto;
- Viene eseguita soltanto la prima sequenza di operazioni con espressione logica uguale a *TRUE*;
- Se nessuna espressione è uguale a *TRUE*, vengono eseguite le operazioni dell'*ELSE*;
- Vi può essere un numero qualsiasi di *ELSIF*;
- *ELSIF* ed *ELSE* sono opzionali.

Esempio:

Vengono passati in ingresso ad un FC due operandi interi. Se il primo è maggiore del secondo, la funzione sottrae il secondo al primo. Se il secondo è maggiore del primo, la funzione sottrae il primo al secondo. Se i due operandi sono uguali, la funzione scrive sull'uscita il valore 0.



6.7.2. Istruzione CASE

L'istruzione *CASE* consente di programmare diramazioni, costringendo il PLC a scegliere una strada in base al valore di un'espressione numerica. Di seguito la struttura dell'istruzione *CASE*:

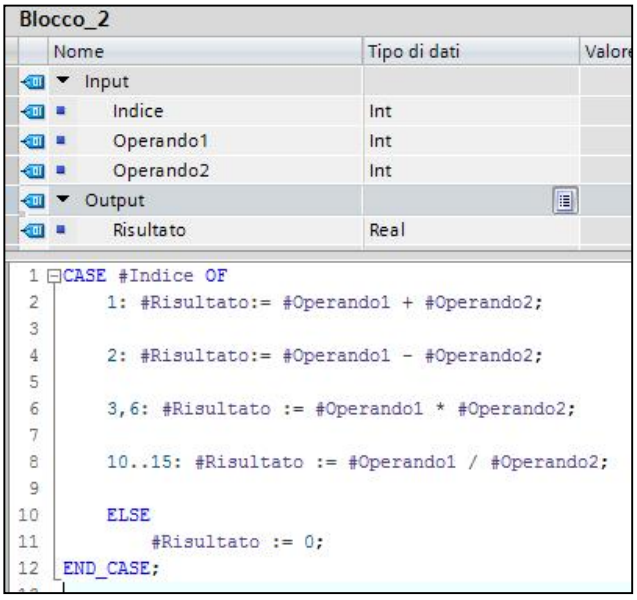
CASE** espressione **OF
costante1: operazioni;
costante2: operazioni;
costante3: operazioni;
ELSE
operazione;
END_CASE;

Le regole principali dell'istruzione *CASE* sono le seguenti:

- La condizione è un'espressione con risultato *INT*, quindi deve essere definita da una variabile o un'espressione aritmetica;
- Viene eseguita la sequenza di istruzioni con costante uguale a quello dell'espressione;
- Se nessuna costante è uguale a quello dell'espressione, vengono eseguite le operazioni dell'*ELSE*;
- L'*ELSE* è opzionale;
- Le costanti possono essere anche un campo di valori (es. 4..10) o un elenco di valori (es. 5,16,17,20).

Esempio:

Vengono passati, in ingresso ad un FC, due operandi e un indice interi. Se l'indice vale 1 il risultato in uscita dall'FC è la somma tra i due operandi, se vale 2 il risultato è la differenza, se l'indice vale 3 o 6 il risultato è la moltiplicazione, se l'indice è un valore tra 10 e 15 il risultato viene calcolato facendo la divisione tra i due operandi. In tutti gli altri casi il risultato è 0.



6.7.3. Istruzione FOR

L'istruzione *FOR* fa elaborare al PLC ripetutamente una serie di operazioni finchè un indice è compreso all'interno di un campo di valori specificato:

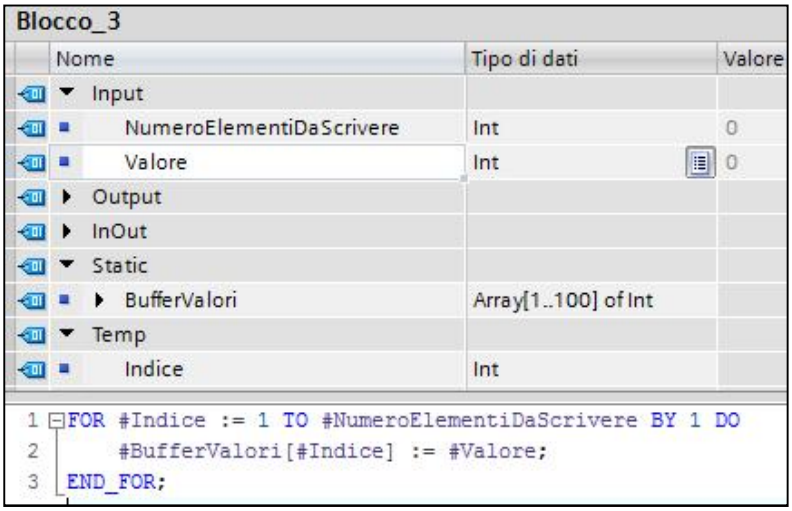
FOR** assegnazione ValoreIniziale all'Indice **TO** ValoreFinale **BY** Passo **DO
operazioni;
END_FOR;

Le regole principali dell'istruzione *FOR* sono le seguenti:

- L'indice di esecuzione deve essere di tipo *INT* o *DINT* (sia variabile locale che globale) e i valori iniziali e finali devono essere dello stesso tipo;
- All'inizio del *FOR* l'indice di esecuzione viene inizializzato;
- Le operazioni vengono ripetute finchè l'indice si trova compreso tra il *ValoreInziale* e il *ValoreFinale*;
- Ad ogni ciclo delle operazioni, la variabile Indice viene automaticamente incrementata o decrementata del valore del *Passo*;
- Se l'ampiezza del passo viene omessa, l'indice verrà incrementato di 1 ad ogni ciclo;
- L'indice può essere utilizzato anche all'interno delle operazioni del *FOR*.

Esempio:

L'FB scrive nei primi N elementi di un array di interi il valore di una variabile. L'array è dichiarato tra le statiche dell'FB, mentre il numero di elementi da scrivere e il valore da scrivere sono ingressi dell'FB.



Al momento del richiamo del blocco, l'istruzione *FOR* inizializza la variabile *Indice* a 1; al primo ciclo, sfruttando la possibilità di utilizzo dell'indice all'interno delle operazioni, viene quindi scritto l'elemento *BufferValori[1]*; finita l'operazione, l'indice viene incrementato di 1 e, se il suo valore non ha ancora superato il valore della variabile *NumeroDiElementiDaScrivere*, torna ad eseguire l'operazione, questa volta con indice 2, scrivendo l'elemento *BufferValori[2]*. Il ciclo FOR continuerà così a ripetere l'operazione finchè l'indice non supera la variabile *NumeroElementiDaScrivere*, dopodichè passerà alla riga di codice successiva all'*END_FOR*.

6.7.4. Istruzione WHILE

L'istruzione *WHILE* fa elaborare al PLC ripetutamente una serie di operazioni finchè è soddisfatta la condizione di esecuzione:

WHILE *Espressione* ***DO***
operazioni;
END_WHILE;

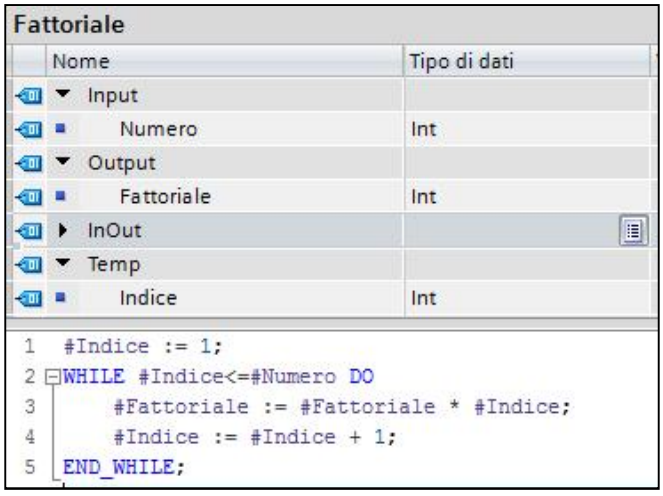
Le regole principali dell'istruzione *WHILE* sono le seguenti:

- La condizione è un'espressione con valore *TRUE* o *FALSE*, quindi deve essere definita da una variabile booleana, da un'espressione logica o un'espressione di confronto;
- Se l'espressione ha valore *FALSE*, le operazioni vengono saltate dato che la condizione viene valutata all'inizio di ogni ciclo;
- Le operazioni vengono ripetute finchè il risultato dell'espressione è TRUE.

ATTENZIONE!!! Dato che non vi sono indici che vengono incrementati automaticamente come nell'istruzione *FOR*, bisogna fare molta attenzione a non rimanere 'intrappolati' nel *WHILE* provocando il superamento di tempo ciclo del PLC. E' importante che, nelle operazioni, ci sia anche qualche istruzione che possa far cambiare lo stato dell'espressione.

Esempio:

Un FC che calcola il fattoriale del numero (il prodotto di tutti i numeri interi positivi minori o uguali al numero stesso) passato in ingresso.



Come si vede, l'indice deve essere inizializzato e incrementato dall'utente dato che il *WHILE* (a differenza del *FOR*) non lo fa in automatico.

6.7.5. Istruzione REPEAT

L'istruzione *REPEAT* fa elaborare al PLC ripetutamente una serie di operazioni finchè è soddisfatta la condizione di esecuzione:

REPEAT
operazioni;
UNTIL *espressione;*
END_REPEAT;

Le regole principali dell'istruzione *REPEAT* sono le seguenti:

- La condizione è un'espressione con valore *TRUE* o *FALSE*, quindi deve essere definita da una variabile booleana, da un'espressione logica o un'espressione di confronto;
- Se anche l'espressione avesse valore *FALSE*, le operazioni vengono eseguite almeno una volta dato che la condizione viene valutata alla fine di ogni ciclo (unica differenza rispetto al *WHILE*);
- Le operazioni vengono ripetute finchè il risultato dell'espressione è *TRUE*.

ATTENZIONE!!! Dato che non vi sono indici che vengono incrementati automaticamente come nell'istruzione *FOR*, bisogna fare molta attenzione a non rimanere 'intrappolati' nel *REPEAT* provocando il superamento di tempo ciclo del PLC. E' importante che, nelle operazioni, ci sia anche qualche istruzione che possa far cambiare lo stato dell'espressione.

Esempio:

Un FB che cerca in un array di 100 elementi di tipo intero, un determinato valore passato come parametro di ingresso. L'FB restituisce il numero del primo elemento in cui ha trovato il valore ricercato.

Ricerca di un valore

	Nome	Tipo di dati	Valore di defa
	Input		
	ValoreDaCercare	Int	0
	Output		
	NumeroElementoTrovato	Int	0
	InOut		
	Static		
	BufferValori	Array[1..100] of Int	

```
1  #NumeroElementoTrovato := 0;
2  REPEAT
3      #NumeroElementoTrovato := #NumeroElementoTrovato + 1;
4  UNTIL #BufferValori[#NumeroElementoTrovato]=#ValoreDaCercare
5      OR #NumeroElementoTrovato=100
6  END_REPEAT;
```

Come si vede, l'indice deve essere inizializzato e incrementato dall'utente dato che il anche il *REPEAT*, come il *WHILE* non lo fa in automatico. Dato che questa funzione deve sicuramente analizzare almeno un elemento, può essere utilizzato il *REPEAT*. Il PLC esce dall'istruzione ciclica solo nel caso in cui l'elemento puntato è uguale al valore da cercare, oppure se ha raggiunto la fine dell'array (quindi il valore non è presente in alcun elemento).

6.7.6. Istruzione CONTINUE

L'istruzione *CONTINUE* fa saltare il ciclo attualmente in esecuzione, dell'istruzione ciclica all'interno della quale si trova:

```
WHILE espressione   DO
        operazioni1;
        IF espressione THEN
                CONTINUE
        END_IF;
        operazioni2;
END_WHILE;
```

Nell'esempio di struttura rappresentato nell'immagine precedente, se l'espressione dell'*IF* risulta vera, vengono saltate le '*operazioni2*' del ciclo attuale e si torna a valutare se l'espressione del *WHILE* è vera o falsa.

6.7.7. Istruzione EXIT

L'istruzione *EXIT* esce completamente dall'istruzione ciclica dove è richiamata:

```
WHILE espressione   DO
        operazioni1;
        IF espressione THEN
                EXIT
        END_IF;
END_WHILE;
```

Nell'esempio di struttura rappresentato nell'immagine precedente, se l'espressione dell'*IF* risulta vera, si esce dal *WHILE* anche se l'espressione del *WHILE* risulta ancora vera.

6.7.8. GOTO

L'istruzione *GOTO* permette di saltare ad un etichetta presente all'interno dello stesso blocco:

```
IF espressione THEN
        GOTO Etichetta1;
END_IF;
Operazioni1;
Etichetta1: Operazioni2;
```

Nell'esempio di struttura rappresentato nell'immagine precedente, se l'espressione dell'*IF* risulta vera, vengono saltate le '*Operazioni1*' passando direttamente ad eseguire le '*Operazioni2*'. Se invece l'espressione dell'*IF* è falsa, vengono eseguite entrambe.

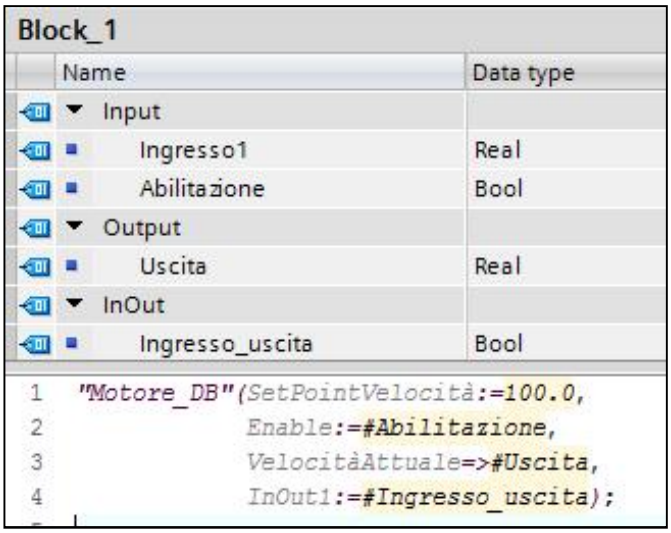
6.7.9. Istruzione RETURN

L'istruzione *RETURN* permette di uscire dall'FC o FB attualmente in esecuzione e tornare all'esecuzione del blocco richiamante.

6.8. Richiamo di funzioni

6.8.1. Richiamo di FB con istanza singola

Di seguito un esempio di richiamo di FB con la relativa DB di istanza.

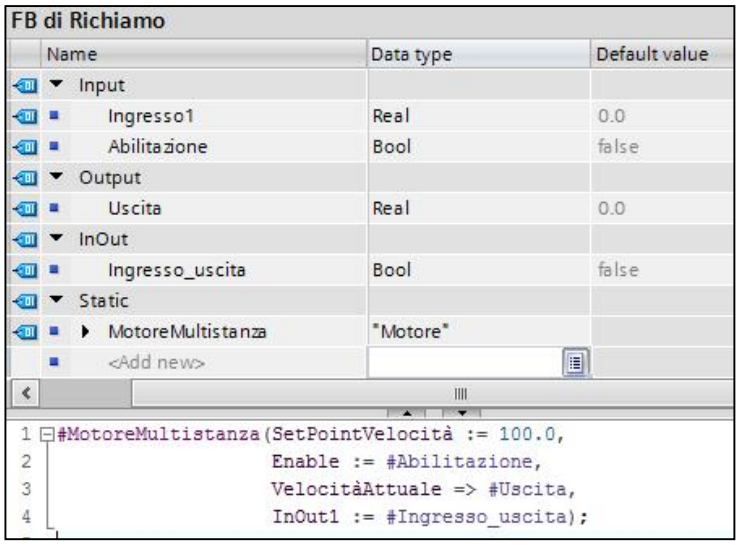


Come si vede, la sintassi del richiamo prevede l'inserimento del nome della DB di istanza dell'FB e, tra parentesi, l'assegnazione di ingressi e uscite:

- Gli Input e gli InOut vengono assegnati utilizzando i caratteri ' := ';
- Gli Output vengono assegnate utilizzando i caratteri ' => ';

6.8.2. Richiamo di FB in multi-istanza

Di seguito un esempio di richiamo di FB in multi-istanza.

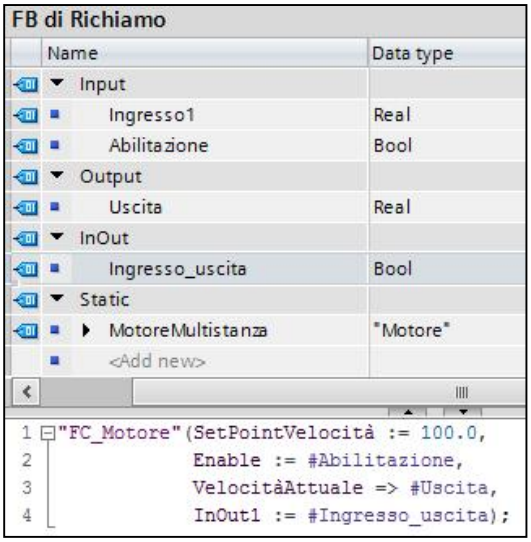


Come si vede, la sintassi del richiamo prevede l'inserimento del nome della variabile definita tra le statiche dell'FB richiamante e, tra parentesi, l'assegnazione di ingressi e uscite:

- Gli Input e gli InOut vengono assegnati utilizzando i caratteri ' := ';
- Gli Output vengono assegnate utilizzando i caratteri ' => ';

6.8.3. Richiamo di FC senza valore di Return

Di seguito un esempio di richiamo di FC che non ha, tra le interfacce, il valore di *RETURN*.



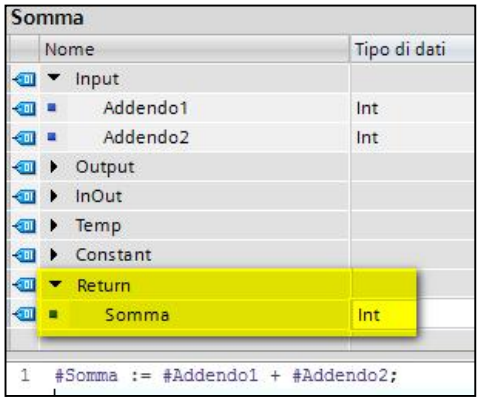
Come si vede, la sintassi del richiamo prevede l'inserimento del nome dell'FC e, tra parentesi, l'assegnazione di ingressi e uscite:

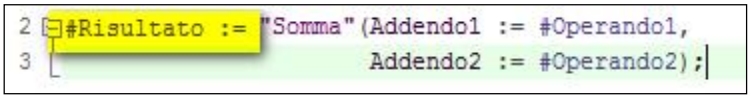
- Gli Input e gli InOut vengono assegnati utilizzando i caratteri ' := ';

- Gli Output vengono assegnate utilizzando i caratteri ' => ';

6.8.4. Richiamo di FC con valore di Return

Mentre in tutti gli altri linguaggi, l'assegnazione del valore di *RETURN* viene gestito come una qualsiasi uscita, in SCL non è così. Di seguito vediamo il richiamo di un FC, tra le cui interfacce è stato definito un valore *RETURN* di tipo *INT*:



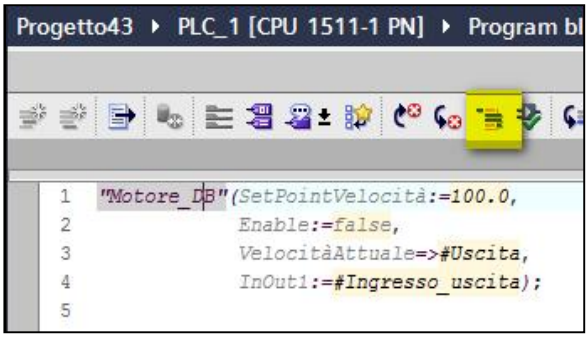


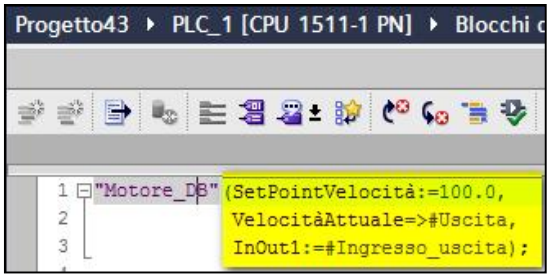
Come si vede, la sintassi del richiamo prevede l'inserimento, davanti al nome dell'FC della sintassi per l'assegnazione del valore *RETURN*. L'assegnazione di ingressi e uscite poi è uguale :

- Gli Input e gli InOut vengono assegnati utilizzando i caratteri ' := ';
- Gli Output vengono assegnate utilizzando i caratteri ' => ';

6.8.5. Nascondere parametri inutilizzati nei richiami

Nel caso di utilizzo di FB, è possibile non associare a tutti i parametri formali di un blocco dei parametri attuali. In tal caso è possibile fare in modo che sull'editor SCL queste variabili vengano nascoste in modo da compattare il codice. Questo è possibile farlo, selezionando il richiamo su cui si vuole eseguire l'operazione e cliccando sul tasto 'Amplia/riduce l'elenco parametri dei richiami dei blocchi'.

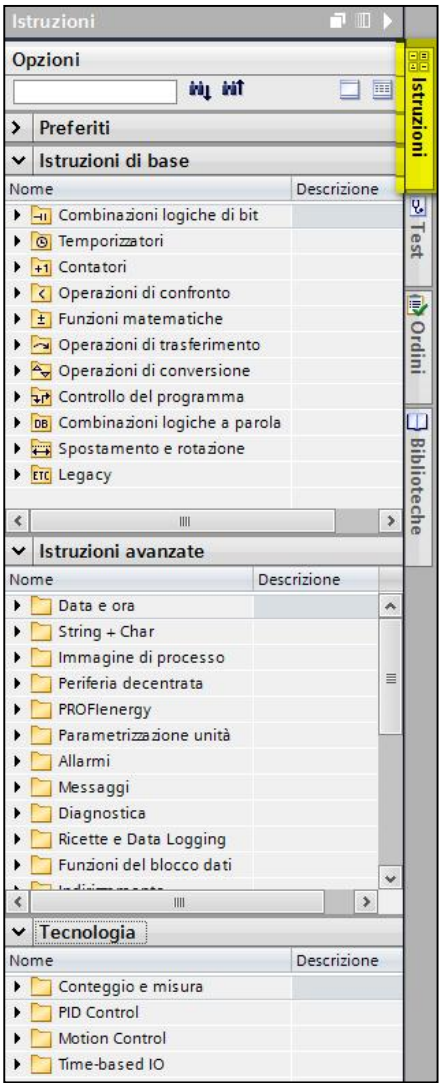




Come si vede dall'esempio, la variabile '*Enable*' che non era stata utilizzata dall'utente (il relativo parametro attuale è rappresentato in grigio ed è uguale al valore di default), dopo l'attivazione della funzionalità, non viene più visualizzato. Cliccando una seconda volta sul tasto, le variabili compaiono nuovamente.

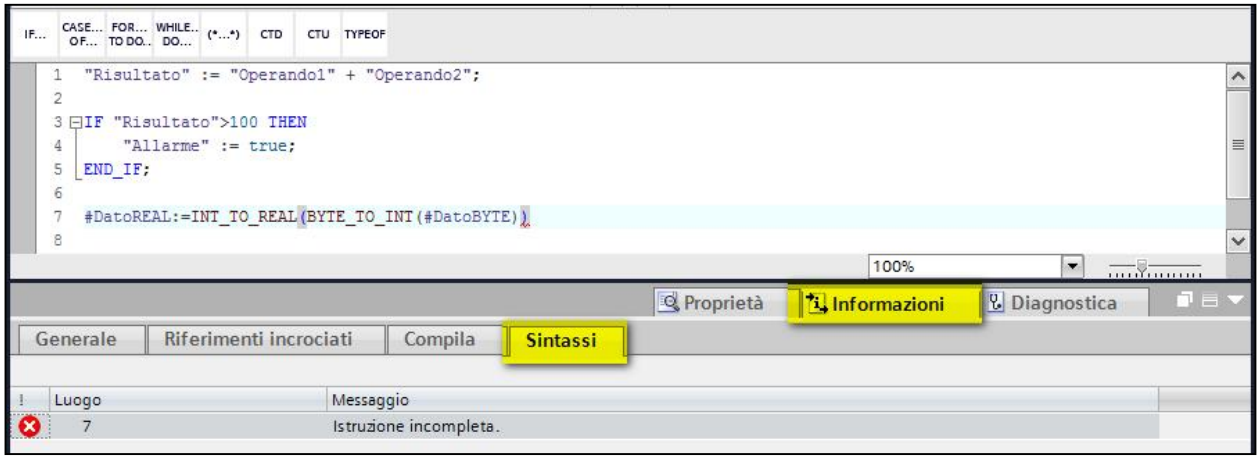
6.9. Altre istruzioni

Le altre istruzioni utilizzabili all'interno del codice SCL, sono FC o FB presenti all'interno del catalogo istruzioni a disposizione nella finestra di sinistra. Basterà quindi portarle nel codice per drag&drop e utilizzare la sintassi vista nel capitolo precedente per assegnare ingressi e uscite.



7. Debug del codice offline

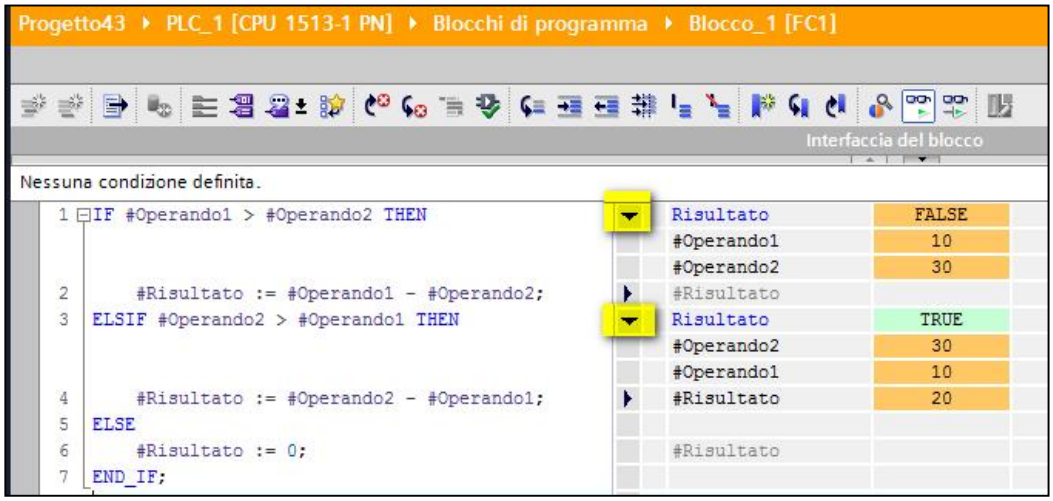
Il codice scritto viene controllato immediatamente da TIA Portal che comunica all'utente eventuali errori di sintassi tramite la finestra delle informazioni, sotto la voce 'Sintassi' come si vede dalla figura seguente.



8. Debug del codice online

Lo strumento che permette di eseguire il debug del codice SCL consiste, come negli altri linguaggi, nell'andare online sul codice e visualizzare lo stato attuale delle variabili aggiornate all'inizio del ciclo PLC.

Lo stato delle variabili è visualizzabile su una colonna che compare a destra del codice. Qui vengono visualizzati gli stati delle uscite presenti su ogni riga. Le righe che non vengono attualmente eseguite dal PLC sono rappresentate in grigio.



E' possibile, cliccando sui relativi triangoli neri, espandere il debug di una relativa riga per vedere anche lo stato di tutte le altre variabili presenti e fare un'analisi più approfondita.

9. Utilità

9.1. Commenti

E' possibile, per avere maggior chiarezza sul codice scritto, inserire dei commenti che verranno colorati, di default, di colore verde.

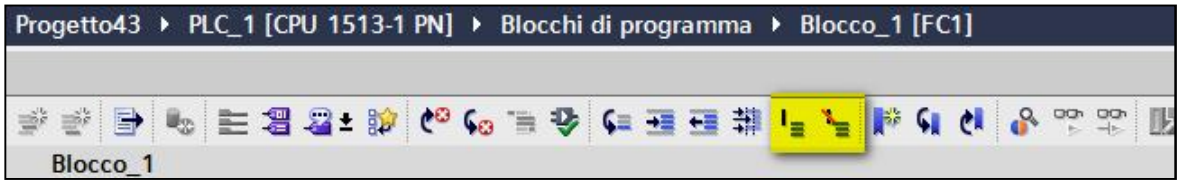
E' possibile commentare la parte destra di una riga, utilizzando i caratteri '//', oppure commentare più righe utilizzando i caratteri '(*' per aprire il commento e '*)' per chiuderlo:

```
(* Controllo dell'operazione da effettuare
-----------------------------------------
-----------------InizioBlocco------------- *)

IF #Operando1 > #Operando2 THEN     //Operando 1 maggiore del 2
    #Risultato := #Operando1 - #Operando2;
ELSIF #Operando2 > #Operando1 THEN  //Operando 2 maggiore dell'1
    #Risultato := #Operando2 - #Operando1;
ELSE
    #Risultato := 0;
END_IF;

(*-----FineBlocco---------------------------
-----------------------------------------*)
```

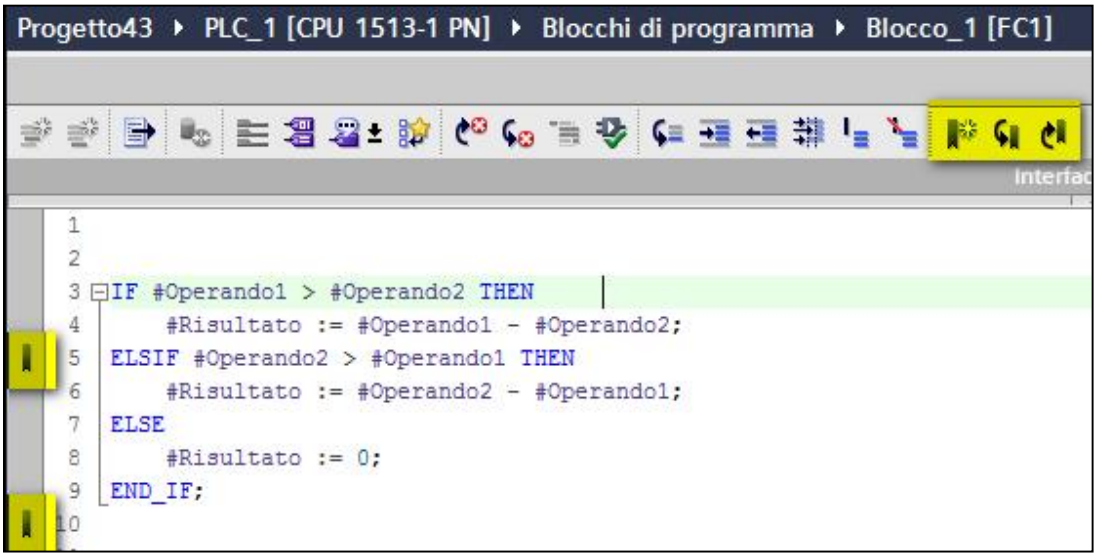
E' possibile commentare/decommentare in modo rapido le righe di codice selezionate cliccando sui tasti rapidi presenti sulla barra degli strumenti.



Questi tasti aggiungono/eliminano, all'inizio delle righe selezionare i caratteri "//'.

9.2. Segnalibro

E' possibile, per avere una migliore navigazione all'interno del codice scritto, inserire dei segnalibro associati ad alcune righe. In questo modo sarà poi possibile saltare da un segnalibro all'altro sfruttando i relativi tasti rapidi nella barra degli strumenti:



9.3. Compattare istruzioni

E' possibile, per avere una migliore navigazione all'interno del codice scritto, compattare le istruzioni cicliche e di selezione dell'SCL ad un'unica, in modo tale da risparmiare spazio. Queste poi possono essere espanse nuovamente cliccando sul relativo tasto immediatamente a sinistra del codice.

```
1  IF #Operando1 > #Operando2 THEN
2      #Risultato := #Operando1 - #Operando2;
3  ELSIF #Operando2 > #Operando1 THEN
4      #Risultato := #Operando2 - #Operando1;
5  ELSE
6      #Risultato := 0;
7  END_IF;
8  #Risultato := 40;
9  #Risultato4 := #Risultato * 6;
10 FOR #Indice := 0 TO 100 BY 1 DO
11     ;
12 END_FOR;
13 WHILE #Indice<100 DO
14     ;
15 END_WHILE;
```

```
1  IF #Operando1  ... THEN ... END_IF;
8  #Risultato := 40;
9  #Risultato4 := #Risultato * 6;
10 FOR #Indice := 0 TO 100 BY 1 DO ... END_FOR;
13 WHILE #Indice<100 DO ... END_WHILE;
```

Quando un'istruzione viene compattata, viene mantenuto il commento scritto di fianco alla fine dell'istruzione in modo tale da avere comunque un codice comprensibile.

```
1  IF #Operando1 > #Operando2 THEN
2      #Risultato := #Operando1 - #Operando2;
3  ELSIF #Operando2 > #Operando1 THEN
4      #Risultato := #Operando2 - #Operando1;
5  ELSE
6      #Risultato := 0;
7  END_IF; //Operazione da eseguire
```

```
1 IF #Operando1  ... THEN ... END_IF; //Operazione da eseguire
8
```