SIEMENS

Ingegno per la vita

Indicizzazione e puntamento dati con S7-1500 e S7-1200

siemens.it/automazione

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.

Sommario

Introduzione 1
1. Indicizzazione simbolica 2
1.1. Indicizzazione in runtime di variabili interne al PLC 2
1.2. Indicizzazione in runtime su immagine di processo 5
1.3. Indicizzazione durante la progettazione 6
2. Operazioni su tipi di dati sconosciuti 8
2.1. Possibilità limitata di tipi di dato accettabili 8
2.2. Tipo di dato completamente indefinito 10
3. Operazioni su array non definiti 11
3.1. Tipo di dato definito ma lunghezza variabile 11
3.2. Tipo di dato e lunghezza sconosciuti 13
4. Costrutto AT 14
Link utili 16

Introduzione

Questo documento è una guida introduttiva all'utilizzo delle funzionalità di programmazione avanzata che permettono di scrivere del codice di programma che lavorerà su dati non ancora del tutto definiti. Questa è l'esigenza che si ha quando si vogliono creare dei blocchi di libreria da riutilizzare in diverse situazioni oppure quando si vuole realizzare un unico blocco riutilizzabile più volte nel programma PLC, con tipi di dati diversi, risparmiando così memoria PLC.

Chi arriva dal modo S7-300 ed S7-400 sa che questo tipo di situazioni venivano risolte con l'utilizzo di tipi di dati ANY e POINTER che però lavoravano sull'indirizzamento assoluto. S7-1500 ed S7-1200 mettono a disposizione nuovi strumenti che permettono di fare le stesse operazioni utilizzando DB ottimizzate e sfruttando così tutti i vantaggi della programmazione simbolica anche per le indicizzazioni.

1. Indicizzazione simbolica

Spesso capita di avere la necessità di scrivere una riga di codice di programma che non sappiamo ancora su quali dati dovrà andare a lavorare. Questo significa avere la necessità di indicizzare i dati. Esistono in realtà due tipi di indicizzazione:

- Indicizzazione durante il runtime: i dati su cui il codice deve lavorare vengono decisi durante il runtime del PLC in base al valore attuale di una o più variabili. Questi dati possono cambiare di ciclo in ciclo. In questo caso, per sfruttare l'indicizzazione simbolica, è obbligatorio disporre i dati in degli array essendo l'unico tipo di dato simbolico che può essere indicizzato.
- Indicizzazione durante la progettazione: i dati su cui il codice deve lavorare vengono decisi al momento del richiamo di un blocco nel codice. Una volta scelti quei dati, quel richiamo del blocco lavorerà sempre su quei dati, in ogni ciclo PLC. In questo caso possiamo scegliere se disporre i dati in array e quindi utilizzare una variabile di indice (in questo caso la variabile di indice non cambierà di valore durante il runtime) oppure se passare i dati come INPUT, INOUT o OUT della funzione che deve indicizzare.

Vediamo, in base all'obiettivo che si vuole raggiungere, qual è la soluzione migliore per indicizzare i dati.

1.1. Indicizzazione in runtime di variabili interne al PLC

Analizziamo in caso in cui abbiamo diversi motori, ognuno con i dati di '*Posizione*', '*Velocità*' e bit di '*Abilitazione*'. L'esigenza del programmatore è scrivere una riga di codice che controlla se la velocità del motore indicato da una variabile di tipo INT che chiamiamo '*NumeroMotore*' è superiore a una soglia.
Sui PLC di vecchia generazione (a meno dell'utilizzo dell'SCL), è necessario gestire la situazione con dei puntatori in AWL, e questo risulta ancora possibile sul PLC di nuova generazione con i puntatori o con le nuove istruzioni di PEEK e POKE.
Se però si vuole sfruttare la programmazione simbolica, sarebbe comodo disporre i dati dei vari motori in un array: l'array è l'unico tipo di dato simbolico che può essere indicizzato durante il runtime, dato che i singoli campi sono contrassegnati da numeri che permettono di saltare da un elemento all'altro.

ProgettoManualeIndirizzamento ▸ PLC_1 [CPU 1511-1 PN] ▸ Blocchi di programma ▸ Dati [DB1]

Mantieni valori attuali | Istantanea | Copia istantanee come valori di avvio | Carica valori di avvio come valori attuali

Dati

	Nome	Tipo di dati	Valore di avvio	A ritenzio...	Accessibile ...	Scrivi...	Visibile in ...	Valore di i..	Controllo	Commento
1	Static									
2	Motore	Array[0..10] of "DatiMotore"								
3	Motore[0]	"DatiMotore"								Motore Nastro
4	Velocità	Real	0.0							
5	Posizione	Real	0.0							
6	Abilitazione	Bool	false							
7	Motore[1]	"DatiMotore"								Motore Carrello
8	Velocità	Real	0.0							
9	Posizione	Real	0.0							
10	Abilitazione	Bool	false							
11	Motore[2]	"DatiMotore"								Motore Pompa
12	Velocità	Real	0.0							
13	Posizione	Real	0.0							
14	Abilitazione	Bool	false							
15	Motore[3]	"DatiMotore"								
16	Motore[4]	"DatiMotore"								

Una volta che i dati sono piazzati in un array, è possibile eseguire l'accesso con la sintassi seguente.

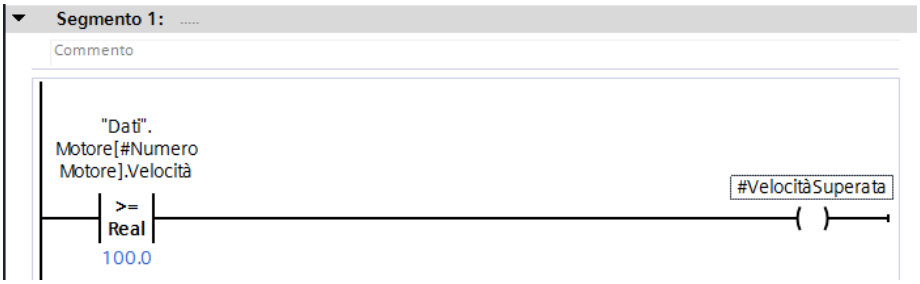
Segmento 1:

Commento

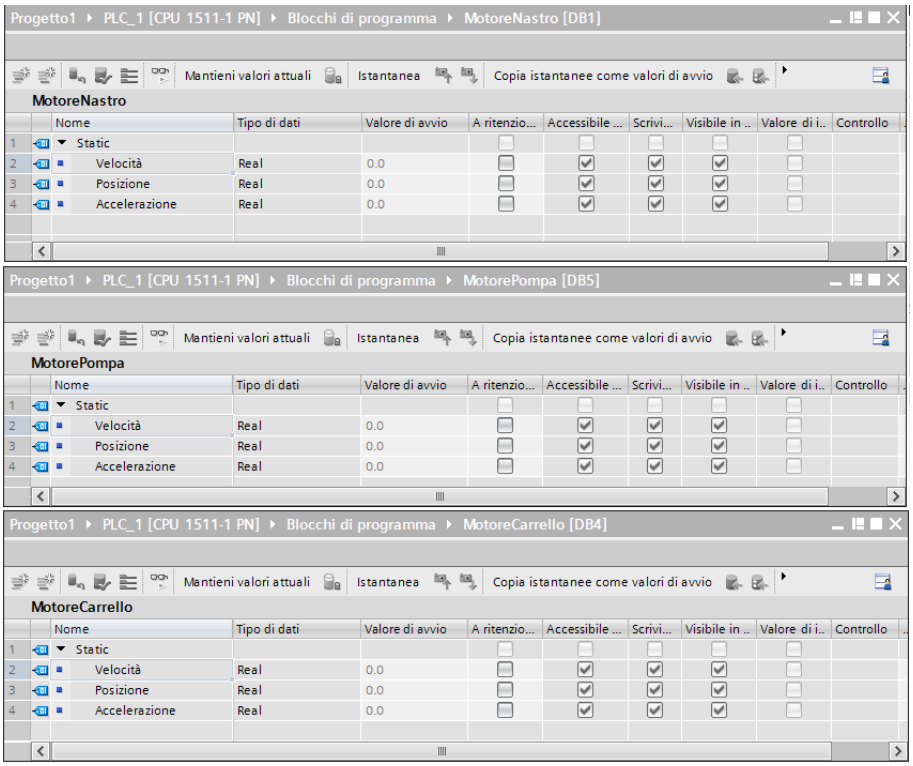
```
1    L    "Dati".Motore[#NumeroMotore].Velocità
2    L    100.0
3    >R
4    =    #VelocitàSuperata
```

La variabile '*NumeroMotore*' è una variabile di tipo INT o DINT dichiarata in qualsiasi area di memoria del PLC (merker, DB, statiche, temporanee...).

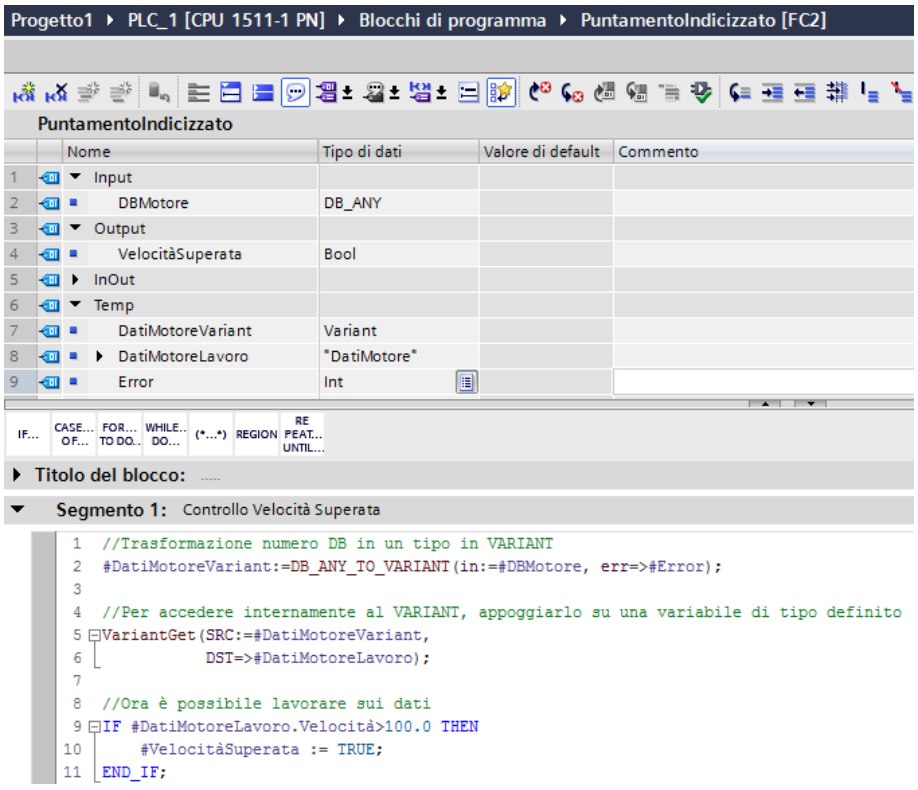
Questa sintassi può anche essere utilizzata in KOP come raffigurato di seguito rendendo possibile l'indicizzazione anche in questo tipo di linguaggio.



Se invece non possiamo disporre i dati in array ma abbiamo la necessità di mantenere i dati da indicizzare in DB separate, è necessario sfruttare un nuovo tipo di dato (DB_ANY) e l'istruzione DB_ANY_TO_VARIANT disponibile soltanto in linguaggio SCL e AWL. Per questo secondo metodo è necessario che le DB da indicizzare siano di un determinato tipo di dato come nell'immagine seguente:



Fatto questo è possibile andare, tramite l'indice *DBMotore* in questo caso passato in ingresso al blocco con un parametro attuale di tipo INT, andare a lavorare su DB differenti utilizzando il codice raffigurato di seguito.



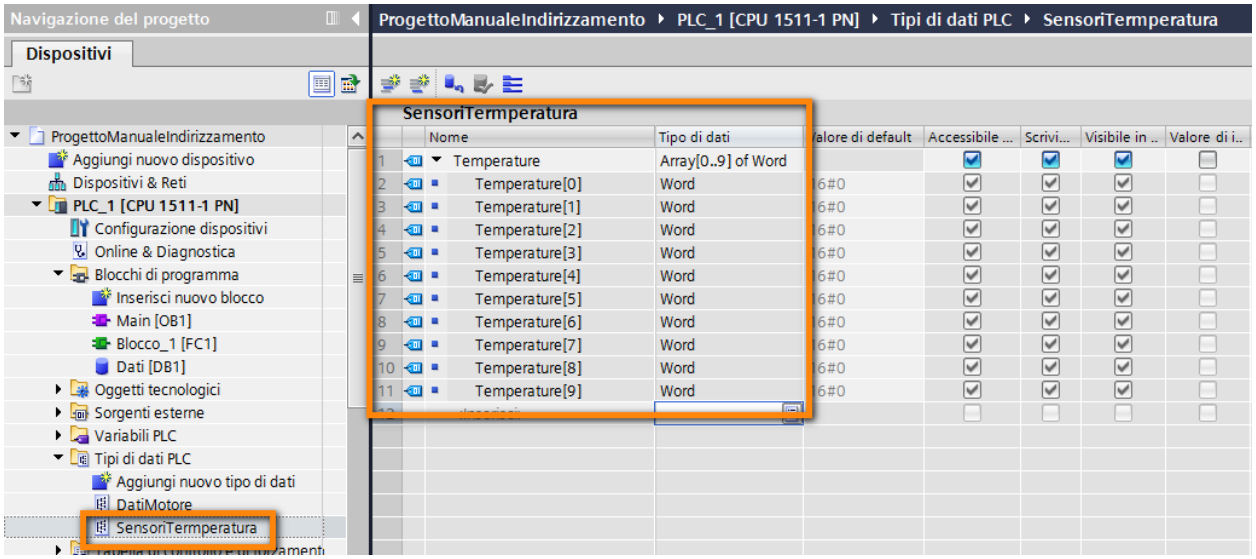
1.2. Indicizzazione in runtime su immagine di processo

E' possibile eseguire l'indicizzazione anche su aree di ingresso e uscite del PLC. Per farlo, anche in questo caso, è necessario definire un array sul quale poter sfruttare la sintassi vista al capitolo precedente. Non è però consentito definire una variabile di tipo array direttamente nella tabella delle variabili.

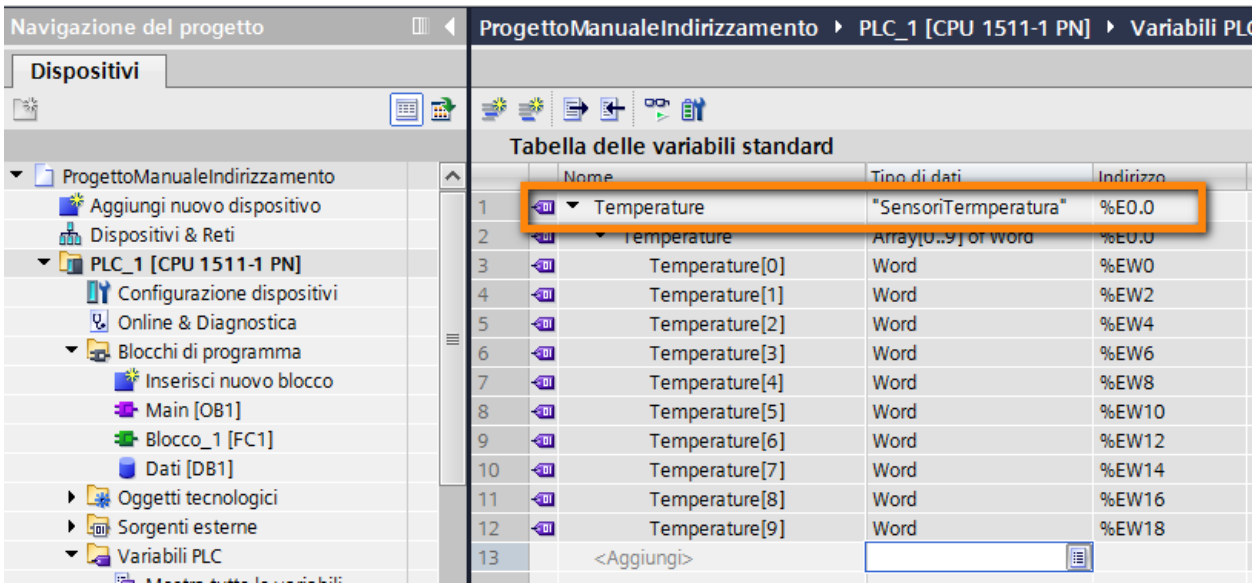
Una delle novità dei PLC di nuova generazione è che sull'immagine di processo, oltre a poter definire variabili di tipi di dati semplici, è possibile definire delle variabili di un tipo di dato utente (quelle che su S7-300 si chiamano UDT). Con questa nuova funzionalità possiamo creare variabili di input e di output che sono strutture o array.

Immaginiamo il caso in cui in ingresso al PLC abbiamo 10 segnali analogici inviati da altrettanti sensori di temperatura. L'obiettivo del programmatore è, in base a un indice, leggere una di queste temperature e spostarla all'interno di una variabile contenuta in una DB.

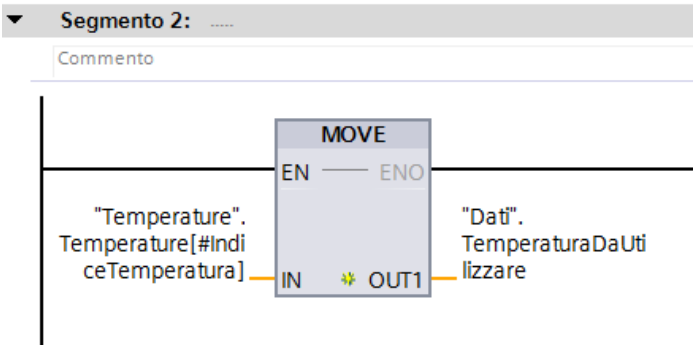
E' necessario prima di tutto realizzare il tipo di dato PLC contenente l'array delle 10 word di ingresso e lo chiamiamo '*SensoriTemperatura*'.



Creare ora la variabile '*Temperature*', del tipo di dato appena creato, che parte dal primo bit dell'area di ingresso dove sono mappati i sensori di temperatura.



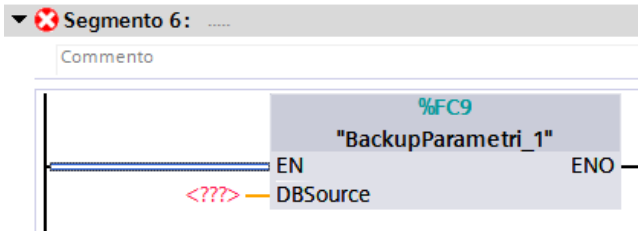
A questo punto, sfruttando una variabile '*IndiceTemperatura*' di tipo intero, è possibile indicizzare in qualsiasi linguaggio l'immagine di processo del PLC come di seguito.



1.3. Indicizzazione durante la progettazione

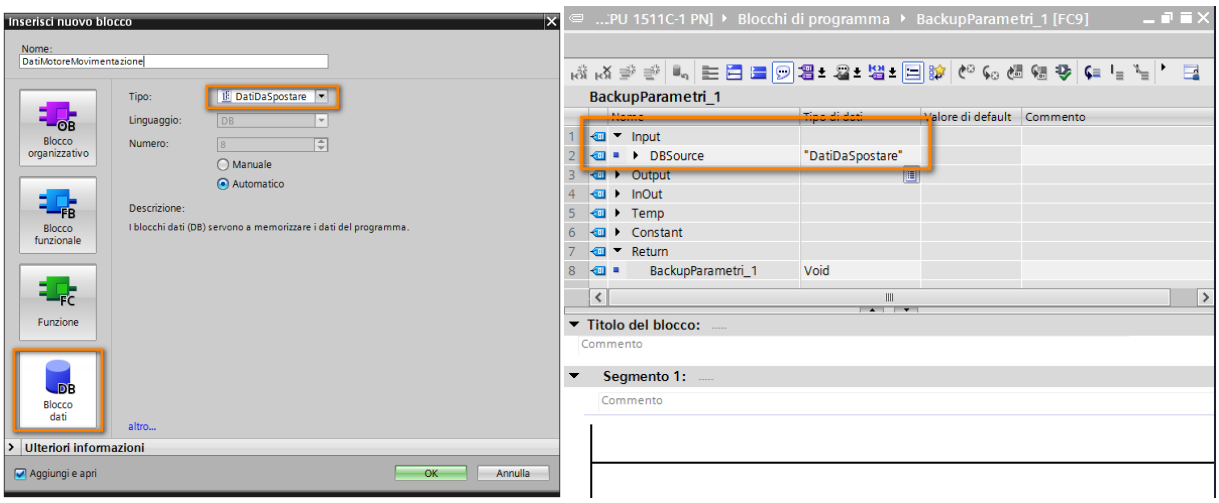
Nel caso in cui i dati su cui deve lavorare una funzione, vengono decisi al momento del richiamo del blocco (poi restano quelli per tutto il runtime), non è strettamente necessario che i dati siano dichiarati ad array. L'alternativa è passare i dati tramite le interfacce del blocco.

Immaginiamo ad esempio il caso in cui creiamo un blocco che dobbiamo far lavorare, ogni volta che lo richiamo, su una DB differente.

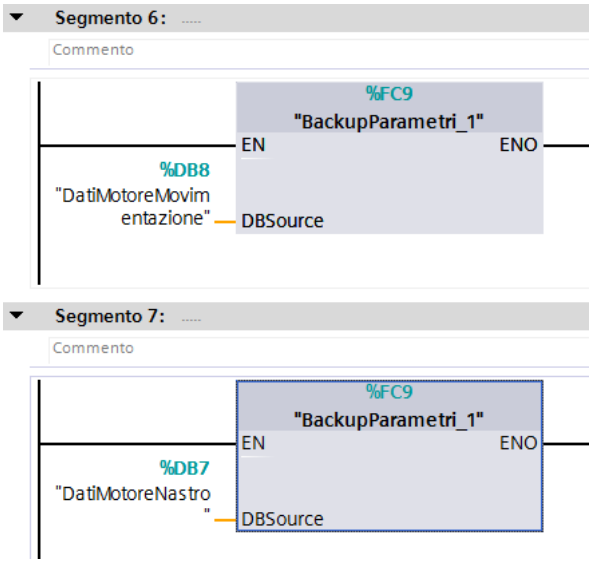


Se non è possibile disporre i dati in un array all'interno di un'unica DB, l'unica soluzione è passare in ingresso l'intera struttura di dati.

Creiamo quindi un tipo di dato da usare sia nella creazione delle DB da puntare, sia nella creazione dell'ingresso dell'FC.



A questo punto, al momento del richiamo dell'FC, si passano in ingresso i dati su cui si vuole lavorare e che di volta in volta potranno essere diversi.



In questo caso, dato che, i dati passati in ingresso a un FC non vengono copiati ma vengono 'puntati' direttamente dal blocco FC, non abbiamo perdite di performance dovute alla copia dei dati. Lo stesso vale per OUT e IN/OUT delle FC. Se invece volessimo usare un FB, per non eseguire la copia dei dati, è necessario dichiararli come IN/OUT.

2. Operazioni su tipi di dati sconosciuti

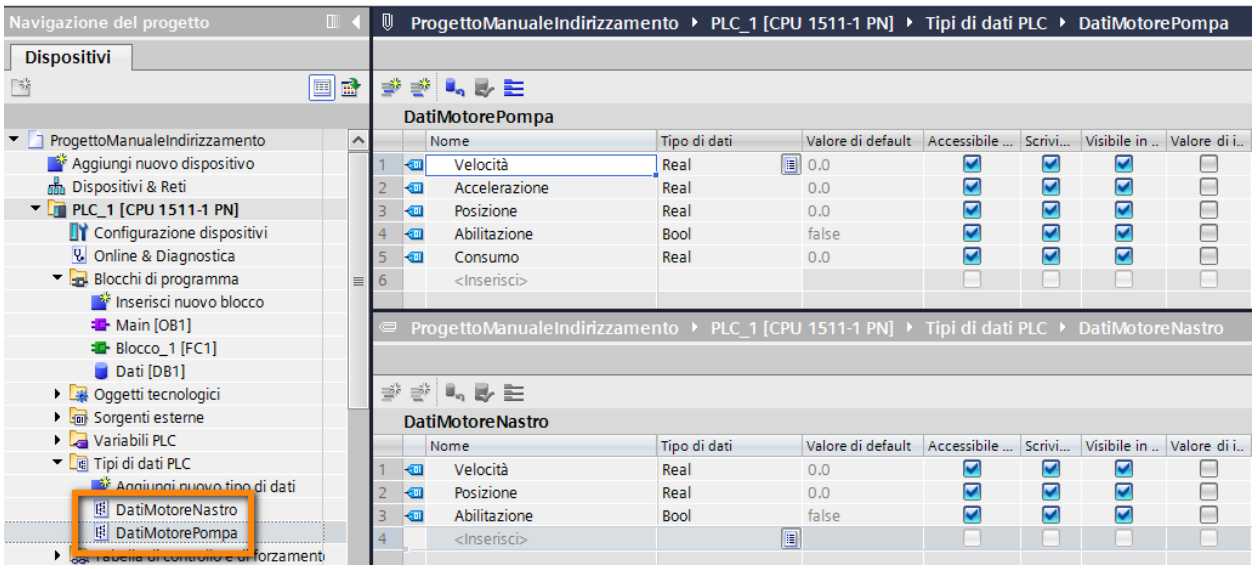
Altra esigenza che, soprattutto in un'ottica di standardizzazione del codice, può capitare di incontrare è quella di dover scrivere del codice che deve lavorare su un dato di interfaccia di un FC o FB di cui non si conosce il tipo di dato perché questo verrà definito solo al momento del richiamo del blocco.
In questo caso sarà possibile lavorare in modo simbolico grazie all'utilizzo del tipo di dato VARIANT nelle interfacce di FB ed FC. Si tratta di un tipo di dato di dimensioni e tipo indefinito che può essere associato a qualsiasi variabile e sul quale possono essere fatte le seguenti operazioni:

- Richiedere se il tipo della variabile puntata dal VARIANT è uguale (o diverso) a quello di un'altra variabile definita (istruzioni *EQ_Type* o *NE_Type*);
- Leggere o scrivere la variabile puntata dal VARIANT (istruzioni *VariantGet* e *VariantPut*);
- Richiedere se la variabile puntata dal VARIANT è di tipo array (istruzione *IS_ARRAY*);
- Richiedere, se il VARIANT punta ad una variabile array, quanti elementi ha (istruzione *CountOfElements*);
- Richiedere, se il VARIANT punta ad una variabile array, se il tipo degli elementi è uguale (o diverso) a quello di un'altra variabile definita (istruzioni*EQ_ElemType* o *NE_ElemType*);
- Richiedere se il tipo di variabile puntata dal VARIANT è uguale a quella puntata da un'altra variabile VARIANT (istruzioni *TypeOf* o *TypeOfElement* solo in SCL).
- Leggere o scrivere, se il VARIANT punta ad una variabile array, i singoli elementi dell'array (istruzione *MOVE_BLK_VARIANT*).

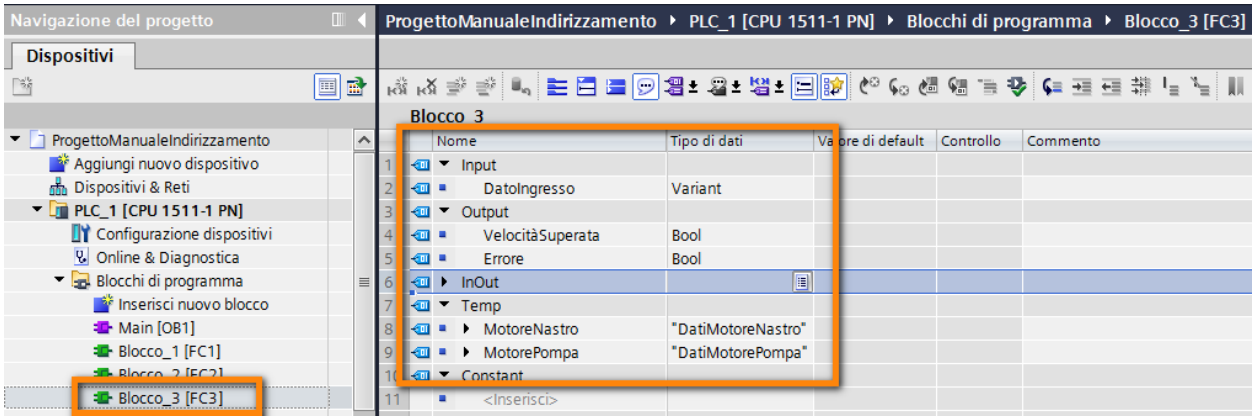
In questo capitolo analizzeremo due situazioni diverse che può capitare di dover gestire con dei tipi di dati diversi dagli array.

2.1. Possibilità limitata di tipi di dato accettabili

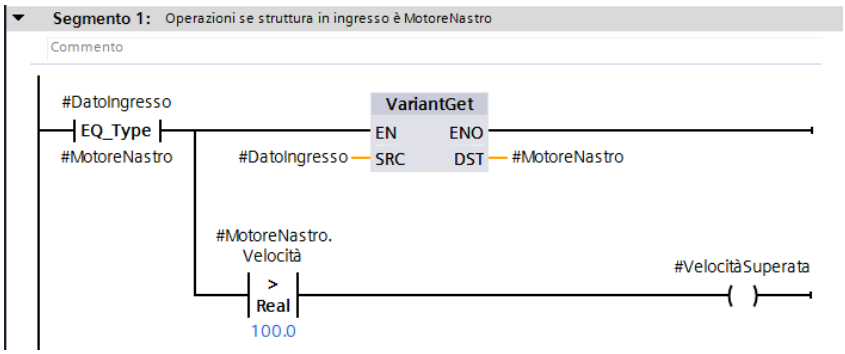
Può capitare che la variabile da accettare in ingresso abbia un set di tipi di dato accettabili.
Immaginiamo, ad esempio, che la variabile in ingresso possa essere del tipo di dato '*DatiMotorePompa*' oppure di tipo '*DatiMotoreNastro*' (due tipi di dato utente). È necessario creare un blocco che, in entrambi i casi, è in grado di controllare se la velocità contenuta all'interno di queste strutture superi una determinata soglia.
È necessario innanzitutto creare i due tipi di dato PLC da utilizzare come illustrato di seguito.



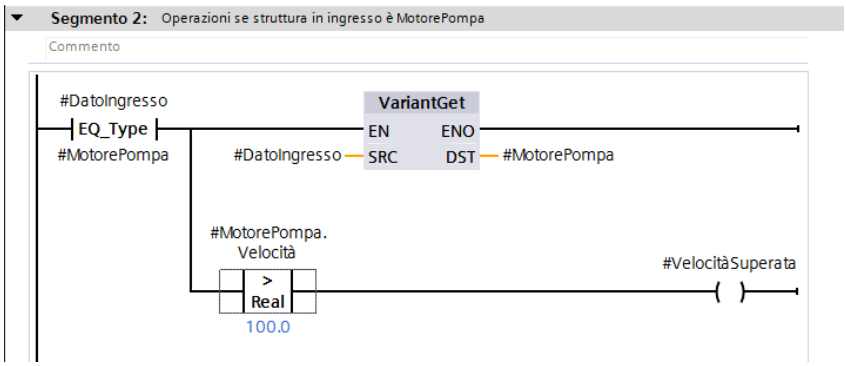
Realizzare quindi il blocco (FC o FB) con in ingresso la variabile di tipo Variant e, tra le variabili interne, una variabile di tipo '*DatiMotoreNastro*' e una di tipo '*DatiMotorePompa*'. In uscita creare la variabile booleana '*VelocitàSuperata*' e la variabile '*Errore*'.



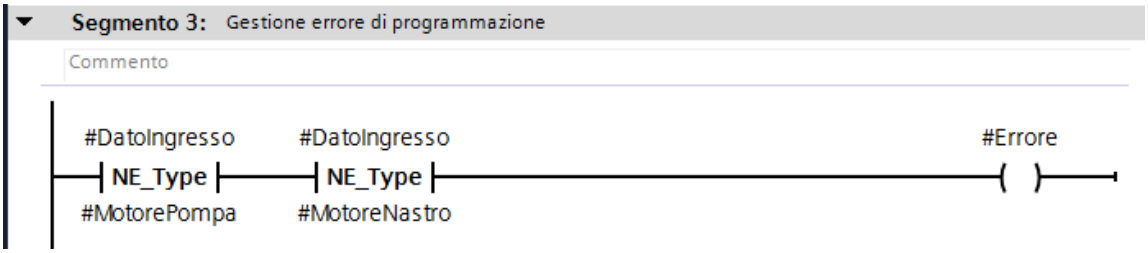
E' necessario quindi usare il blocco EQ_TYPE per sapere se il tipo di dato in ingresso è uguale al tipo di dato della variabile interna '*MotoreNastro*'. In caso di risposta positiva è necessario copiare il valore attuale del dato di ingresso sulla variabile interna così da poter accedere alla variabile '*Velocità*' interna alla struttura e controllarne il valore.



Scrivere poi la parte di codice che deve intervenire nel caso in cui la variabile di ingresso sia di tipo '*MotorePompa*'. Anche in questo caso la variabile di ingresso viene appoggiata sulla relativa variabile interna per poter accedere alle variabili contenute nella struttura.



Nel caso in cui il programmatore sbaglia e appoggia sulla variabile VARIANT un tipo di dato diverso da quelli attesi dal blocco, è possibile gestire l'errore con il codice seguente.



2.2. Tipo di dato completamente indefinito

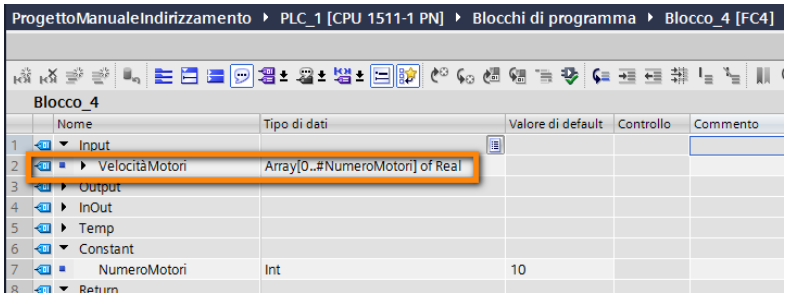
Se il tipo di dato, a differenza del caso precedente, è totalmente indefinito, di certo non sarà necessario entrare all'interno della struttura per utilizzare delle variabili. In questo caso, ciò che potremo fare sarà soltanto leggere o scrivere questa variabile su un'altra dello stesso tipo. Sarà quindi sufficiente utilizzare, all'interno del blocco, i comandi *VariantGet* e *VariantPut.*

3. Operazioni su array non definiti

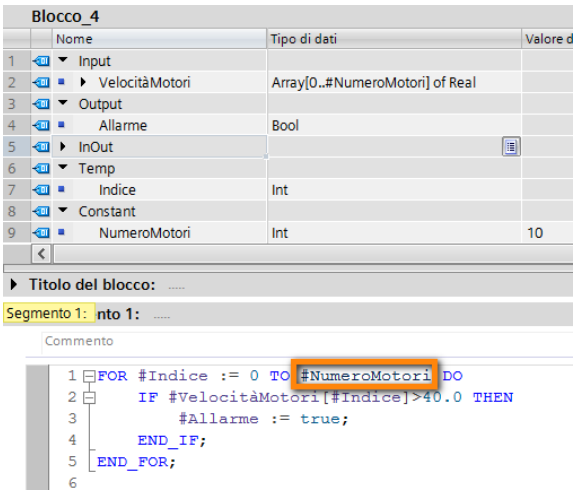
Molto spesso, quando si utilizzano accessi indiretti, si ha l'esigenza di organizzare i dati in array per poterli indicizzare in simbolico. In questo capitolo vedremo quali strumenti mettono a disposizione i PLC di nuova generazione per poter lavorare al meglio con array che non sono totalmente definiti al momento della stesura del codice.

3.1. Tipo di dato definito ma lunghezza variabile

Analizziamo il caso in cui è necessario scrivere del codice che lavora su un array di elementi di cui è noto il tipo di dato ma non la quantità degli elementi. Questa verrà stabilita al momento dell'utilizzo del blocco. Per esempio, ci si potrebbe trovare in questo caso, se dovessimo realizzare un FC o FB da riutilizzare in progetti diversi. In base al progetto però potremmo avere più o meno elementi. Possiamo per esempio definire come ingresso al blocco una variabile di tipo *'Array[0..#NumeroMotori] of Real*' dove *NumeroMotori* è una costante (in questo caso vale 10 ed è definita internamente al blocco) che definisce qual è l'ultimo elemento dell'array in questo determinato progetto.



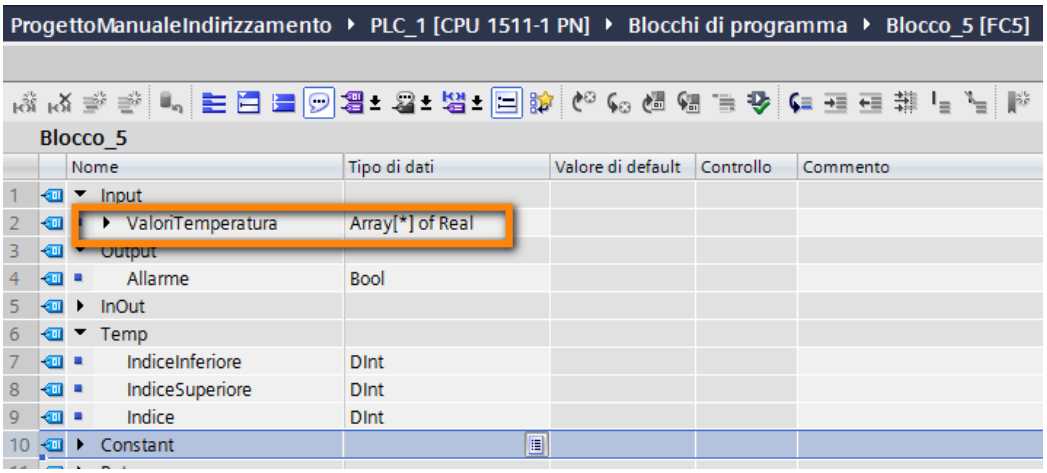
La costante potrà poi essere utilizzata anche all'interno del codice come nell'esempio seguente:



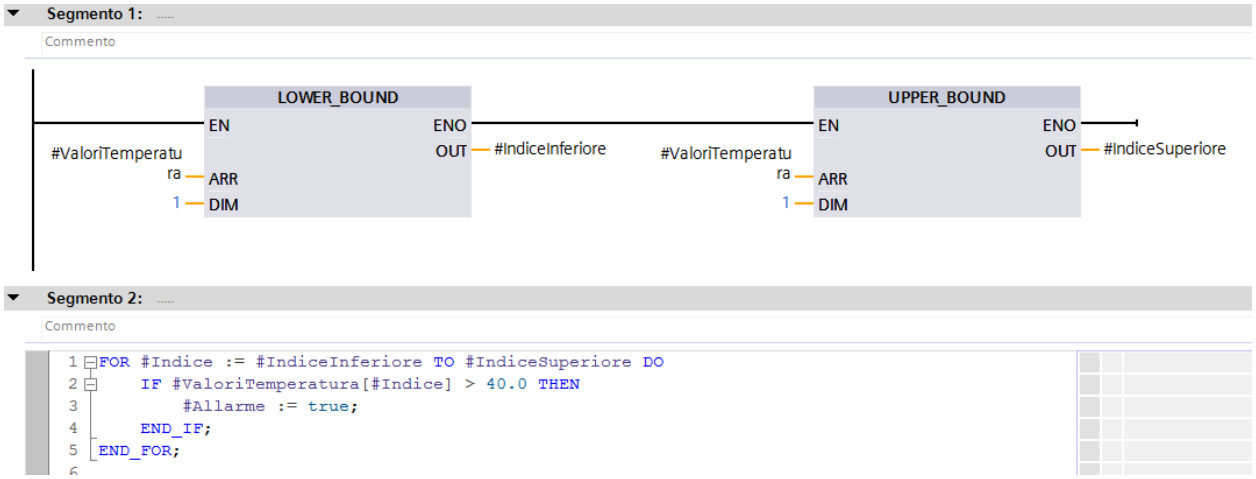
In questo modo, cambiando il valore della costante nel prossimo progetto, il codice inizierà a lavorare su un numero di elementi diverso senza doverlo modificare.

Questo strumento non può essere utilizzato se è necessario utilizzare l'FC o FB più volte nello stesso progetto con numeri di elementi di volta in volta diversi. In questo caso, esiste un altro strumento che

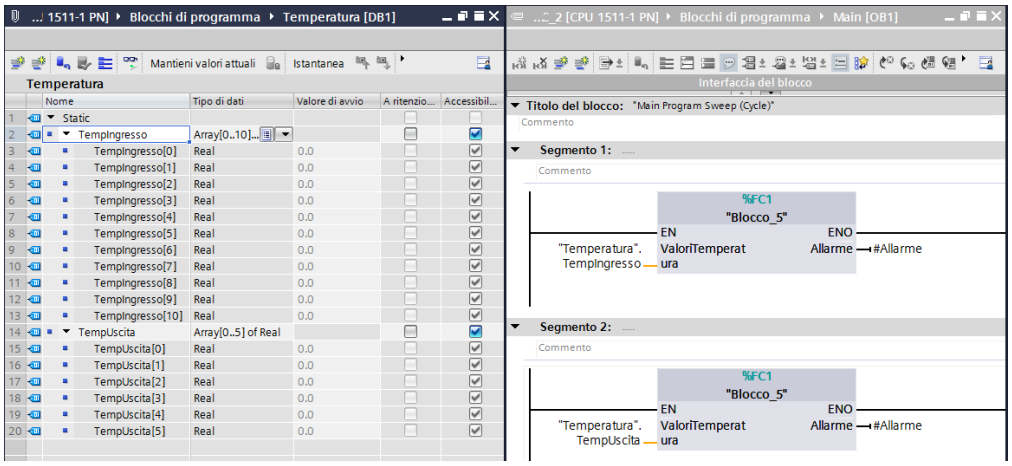
permette di realizzare, tra la variabili di interfaccia di un FB o FC, una variabile di tipo array con numero di elementi totalmente indefinito con la sintassi *'Array[*] of TipoDiDato*'. In questo modo, per ogni richiamo del blocco, viene definito un numero di elementi diversi su cui lavorare.



Per capire quanti elementi ha la variabile attuale associata al momento del richiamo del blocco dal programmatore, si possono utilizzare le due istruzioni LOWER_BOUND e UPPER_BOUND che restituiscono il numero del primo e dell'ultimo elemento. E su questi è poi possibile far lavorare il codice, come rappresentato di seguito.



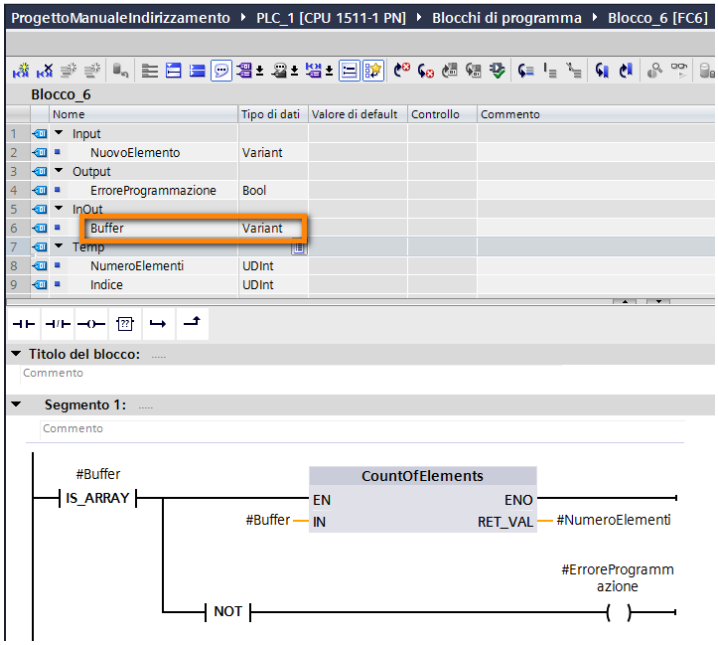
A questo punto è possibile richiamare questo blocco più volte nello stesso progetto facendolo lavorare di volta in volta su un numero di elementi diversi, come nell'esempio seguente dove gli passiamo in ingresso prima un array di 11 elementi e poi uno di 6.



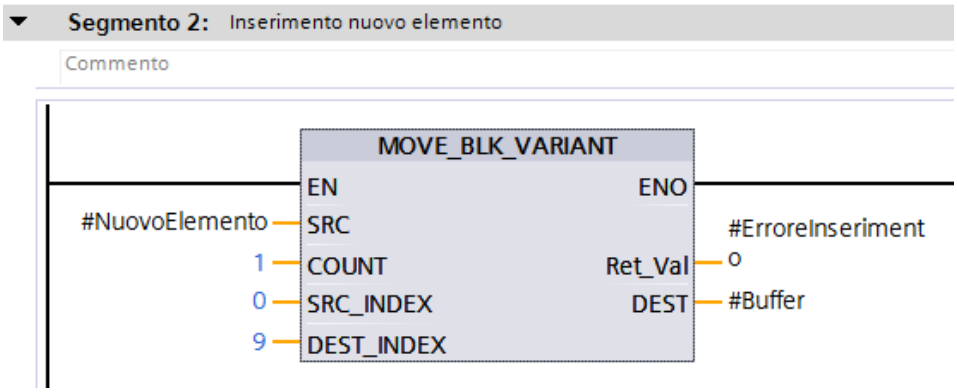
3.2. Tipo di dato e lunghezza sconosciuti

Il caso appena visto presupponeva che si conoscesse il tipo di dato degli elementi dell'array. Può capitare però di dover scrivere del codice che deve lavorare su array con elementi indefiniti. Pensiamo per esempio ad un blocco che deve gestire un buffer di dati con logica FIFO: in questo caso il blocco in questione deve lavorare con qualsiasi tipo di dato, che siano INT, REAL o strutture non ha importanza dato che la logica deve prendere i dati 'a pacchetto' e spostarli all'interno del buffer. Inoltre, di volta in volta, il blocco deve poter lavorare con buffer di lunghezza diversa e quindi è necessario gestire un array di lunghezza variabile.

Questa situazione è risolvibile simbolicamente con l'utilizzo del tipo di dato VARIANT, sfruttando l'istruzione 'CountOfElements' che permette di chiedere ad un tipo di dato VARIANT quanti elementi ha il suo dato attuale. È inoltre possibile identificare un eventuale errore di programmazione chiedendo se la variabile attuale è un array come ci si aspetta, sfruttando l'istruzione IS_ARRAY.



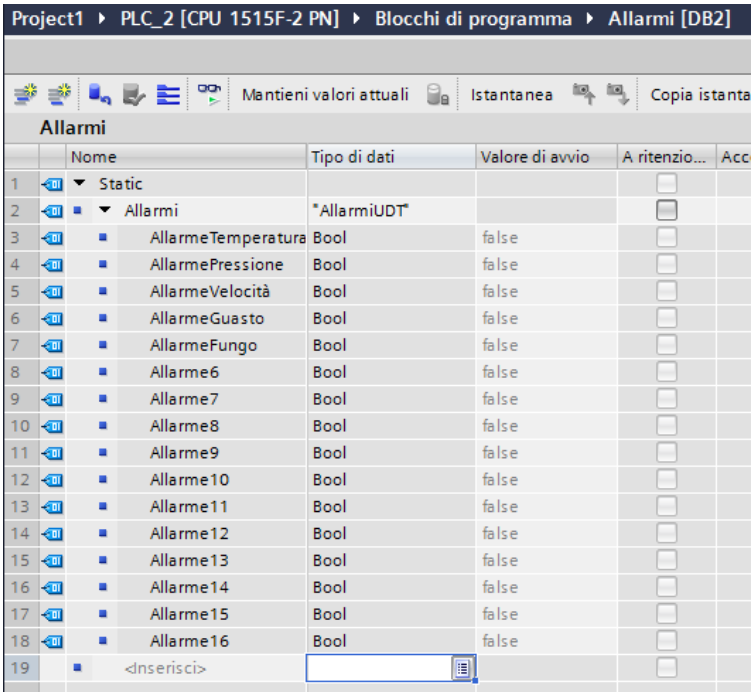
Una volta scoperto il numero di elementi del buffer, è possibile andare a leggere o scrivere i singoli elementi dell'array grazie all'istruzione MOVE_BLK_VARIANT. Nell'immagine seguente si vede come è possibile inserire il nuovo elemento nel nono elemento dell'array '*Buffer*', indipendentemente da quale tipo di dati abbiano.



4. Costrutto AT

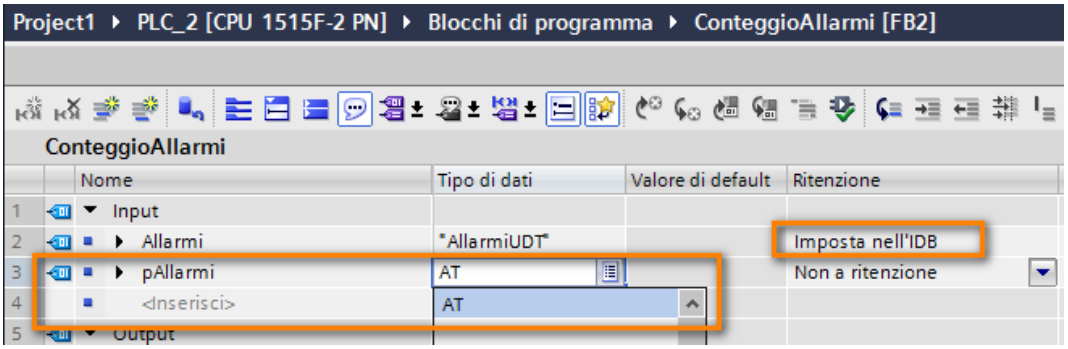
Può capitare di trovarsi nella situazione in cui può essere comodo lavorare sulla stessa variabile con due tipi di dati diversi da poter utilizzare a seconda delle esigenze. Questo non è possibile farlo sulle variabili globali (merker, DB globali, variabili statiche di FB) ma è possibile farlo sulle interfacce di input FB (input, output e in/out) grazie al costrutto AT.

Immaginiamo di aver definito gli allarmi macchina in una DB, all'interno di una struttura di bit come di seguito:

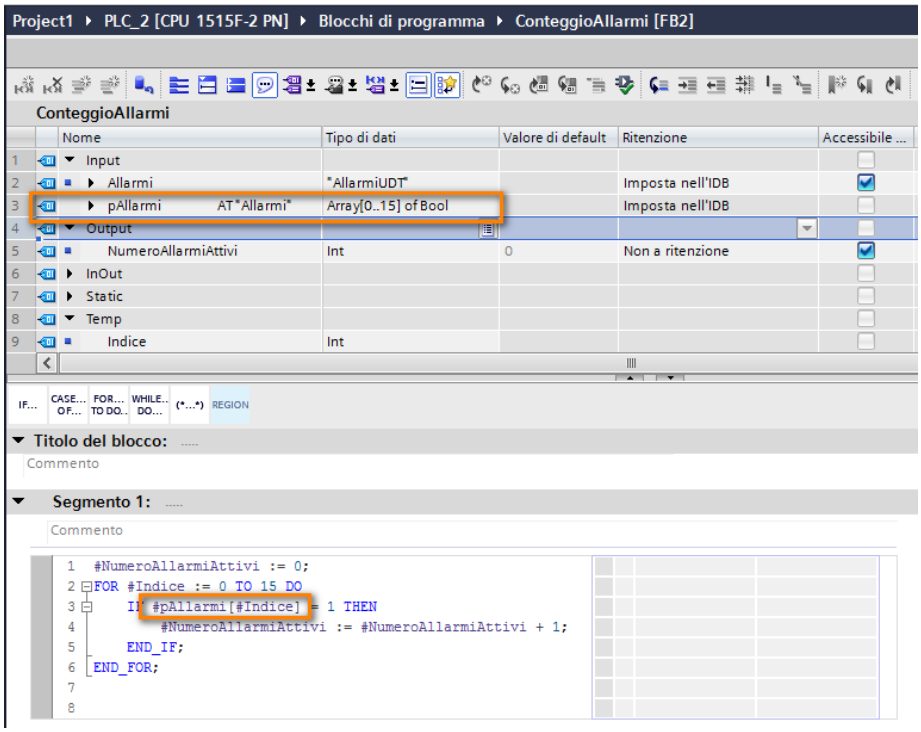


	Nome	Tipo di dati	Valore di avvio	A ritenzio...	Acc
1	Static				
2	Allarmi	"AllarmiUDT"			
3	AllarmeTemperatura	Bool	false		
4	AllarmePressione	Bool	false		
5	AllarmeVelocità	Bool	false		
6	AllarmeGuasto	Bool	false		
7	AllarmeFungo	Bool	false		
8	Allarme6	Bool	false		
9	Allarme7	Bool	false		
10	Allarme8	Bool	false		
11	Allarme9	Bool	false		
12	Allarme10	Bool	false		
13	Allarme11	Bool	false		
14	Allarme12	Bool	false		
15	Allarme13	Bool	false		
16	Allarme14	Bool	false		
17	Allarme15	Bool	false		
18	Allarme16	Bool	false		
19	<Inserisci>				

Ora però vogliamo sapere quanti bit di questa struttura sono attivi: per fare questa operazione sarebbe comodo avere questi bit in un array in modo da poterli 'spazzolare' tutti con un ciclo FOR. Per fare questo è possibile creare un FB con un ingresso dello stesso tipo della struttura allarmi. E' importante, perché il costrutto AT possa funzionare, impostare la voce della colonna ritenzione su 'Imposta nell'IDB'.
Quindi è possibile definire la variabile successiva con 'Tipo di dati' AT come nell'immagine di seguito:



Cliccando invio, il sistema creerà una variabile sovrapposta a quella precedente ma che potrà essere definita con un tipo diverso che nel nostro caso sarà un array di 16 BOOL. In questo modo, all'interno dell'FB in questione, sarà possibile lavorare sulla struttura di allarmi come se fosse un array.



Link utili

Informazioni più approfondite su diverse funzionalità di programmazione sono disponibili sulla guida *'Programming Guideline for S7-1200/1500'* scaricabile dal seguente link:

https://support.industry.siemens.com/cs/it/it/view/81318674

Ulteriori informazioni possono essere consultate sul manuale *'STEP 7 Professional V15 SP1*' scaricabile dal link seguente, al capitolo 3.11 della sezione '*Programmazione PLC*':

https://support.industry.siemens.com/cs/it/it/view/109755202

Informazioni relative al tema della programmazione simbolica con S7-1200 ed S7-1500 sono disponibili anche sul video tutorial disponibile al seguente link:

https://youtu.be/I-nMHaCy210?list=PL6D2D2A1F8B78E2FA

Per ulteriori informazioni visita il sito:
https://new.siemens.com/it/it/prodotti/automazione.html

I dati tecnici presentati in questo documento si basano su un caso di utilizzo reale o su parametri progettuali, pertanto non è possibile fare affidamento a essi per qualsivoglia applicazione specifica e non costituiscono garanzia di prestazioni per qualsiasi progetto.
I risultati effettivi dipendono da una serie di condizioni variabili. Di conseguenza, Siemens non emette alcuna rappresentanza, garanzia, assicurazione in relazione all'accuratezza, vigenza o completezza dei contenuti riportati nel presente documento. Su richiesta, verranno forniti dati tecnici specifici oppure specifiche riguardanti applicazioni particolari del cliente. L'azienda lavora continuativamente nell'ingegnerizzazione e nello sviluppo. Per tale ragione, si riserva il diritto di apportare modifiche in qualsiasi momento alla tecnologia e alle specifiche del prodotto contenute nel presente documento.