Texte structuré (ST)

15

Objet de ce sous-chapitre

Ce chapitre décrit le langage de programmation Littéral structuré ST conforme à la norme CEI 61131.

Contenu de ce chapitre

Ce chapitre contient les sous-chapitres suivants :

15.1 Remarques générales sur le littéral structuré ST

Objet de ce sous-chapitre

Ce sous-chapitre vous donne un aperçu général sur le littéral structuré ST.

Contenu de ce sous-chapitre

Ce sous-chapitre contient les sujets suivants :

Informations générales sur le texte structuré (ST)

Présentation

Le langage Littéral structuré (ST) vous permet par exemple d'appeler des blocs fonction, d'exécuter des fonctions, de lancer des affectations, d'exécuter des instructions conditionnelles et de répéter des instructions.

Expression

Le langage ST utilise ce que l'on appelle des "expressions".

Les expressions sont des constructions comprenant opérateurs et opérandes qui livrent une valeur lors de leur exécution.

Opérateur

Les opérateurs sont des symboles pour les opérations à exécuter.

Opérande

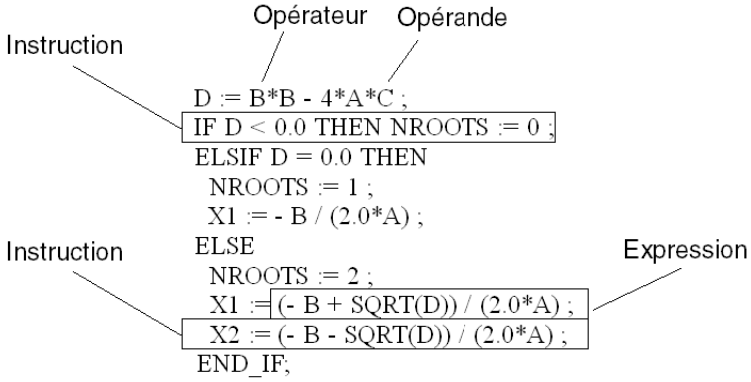
Les opérateurs sont utilisés sur les opérandes. Les opérandes sont par exemple des variables, des valeurs littérales, des entrées/sorties FFB, etc.

Instructions

Les instructions permettent d'affecter les valeurs retournées par des expressions à des paramètres réels et de structurer et commander les expressions.

Représentation d'une section ST

Représentation d'une section ST :



Taille de la section

La taille d'une ligne d'instruction est limitée à 300 caractères.

La longueur d'une section ST n'est pas limitée au sein de l'environnement de programmation. La longueur d'une section ST n'est limitée que par la taille de la mémoire de l'automate.

Syntaxe

Il n'est pas fait de différence entre majuscules et minuscules lors de la saisie des identificateurs et des mots-clés.

Exception : Les caractères d'espacement et de tabulation ne sont pas autorisés dans :

- les mots-clés,
- les valeurs littérales,
- les valeurs,
- les identificateurs,
- les variables et
- combinaisons du limiteur [par ex., `(*` pour commentaires)].

Ordre d'exécution

L'interprétation d'une expression consiste à appliquer les opérateurs aux opérandes, selon la séquence qui est définie par le rang des opérateurs (voir Tableau des opérateurs *(voir page 514)*). Le système exécute d'abord l'opérateur présentant le rang le plus élevé dans l'expression, suivi de l'opérateur du rang inférieur suivant, etc. jusqu'à ce que l'opération soit terminée. Les opérateurs de même rang sont exécutés de gauche à droite, comme ils sont écrits dans l'expression. Cet ordre de traitement peut être modifié en utilisant des parenthèses.

Si par exemple `A`, `B`, `C` et `D` ont respectivement les valeurs 1, 2, 3 et 4, et que le calcul est effectué comme suit :

```
A+B-C*D
```

alors le résultat sera -9.

Pour un calcul tel que :

```
(A+B-C)*D
```

le résultat sera 0.

Si un opérateur a deux opérandes, l'opérande gauche est exécuté en premier. Par exemple, dans l'expression

```
SIN(A)*COS(B)
```

l'expression `SIN(A)` est calculée en premier, puis c'est au tour de `COS(B)` et enfin le produit est calculé.

Comportement en cas d'erreur

Les conditions suivantes seront traitées comme des erreurs lors de l'exécution d'une expression, par exemple :

- tentative de division par 0.
- les opérandes n'ont pas le type de données correct pour l'opération.
- le résultat d'une opération numérique dépasse la plage de valeurs de son type de données.

Si une erreur se produit lors de l'exécution d'une opération, le bit système correspondant (%S) est activé (si cela est pris en charge par l'automate utilisé).

Conformité CEI

Pour plus d'informations sur la conformité CEI du langage ST, voir Conformité CEI *(voir page 657)*.

Opérandes

Présentation

Un opérande peut être :
- une adresse,
- un libellé,
- une variable,
- une variable multi-éléments,
- un élément d'une variable multi-éléments,
- un appel de fonction ou
- une sortie FFB.

Types de données

Les types de données des opérandes à traiter dans une instruction doivent être identiques. Si des opérandes de différents types de données doivent être traités, une conversion de types doit obligatoirement être effectuée auparavant.

Dans l'exemple, la variable Integer `i1` est convertie en une variable Real, avant d'être ajoutée à la variable Real `r4`.

```
r3 := r4 + SIN(INT_TO_REAL(i1)) ;
```

Comme exception à cette règle, des variables du type de données `TIME` peuvent être multipliées par des variables du type de données `INT`, `DINT`, `UINT` ou `UDINT` ou divisées par ces dernières.

Opérations autorisées :
- `timeVar1 := timeVar2 / dintVar1;`
- `timeVar1 := timeVar2 * intVar1;`
- `timeVar := 10 * time#10s;`

Cette fonction est considérée comme " indésirable " par la norme CEI 61131-3.

Utilisation directe d'adresses

Les adresses peuvent être utilisées directement (sans déclaration préalable). Dans ce cas, le type de données est directement affecté à l'adresse. L'affectation a lieu via le "préfixe de taille".

Le tableau suivant indique les différents préfixes de taille :

Préfixe de taille / Symbole	Exemple	Type de données
pas de préfixe	`%I10`, `%CH203.MOD`, `%CH203.MOD.ERR`	`BOOL`
`X`	`%M`**`X`**`20`	`BOOL`
`B`	%Q**B**102.3	`BYTE`
`W`	`%K`**`W`**`43`	`INT`
`D`	`%Q`**`D`**`100`	`DINT`
`F`	`%M`**`F`**`100`	`REAL`

Utilisation d'autres types de données

Si d'autres types de données doivent être affectés en tant que types de données par défaut d'une adresse, cela doit faire l'objet d'une déclaration explicite. L'éditeur de variables facilite la déclaration de ces variables. Il n'est pas possible de déclarer directement le type de données d'une adresse dans une section ST (par ex. la déclaration `AT %MW1 : UINT` ; non permise).

Exemple : les variables ci-dessous sont déclarées dans l'éditeur de variables.

```
UnlocV1 : ARRAY [1..10] OF INT;
LocV1 :   ARRAY [1..10] OF INT AT %MW100;
LocV2 :   TIME AT %MW100;
```

Les appels ci-dessous sont donc corrects du point de vue de la syntaxe :

```
%MW200 := 5;
UnlocV1[2] := LocV1[%MW200];
LocV2      := t#3s;
```

Accès aux variables de champs

Lors d'un accès aux variables de champ (`ARRAY`), seuls les libellés et les variables du type `INT`, `UINT`, `DINT` et `UDINT` sont autorisés dans l'indication d'index.

L'index d'un élément ARRAY peut être négatif si la limite inférieure de la plage est négative.

Exemple : Emploi de variables de zone

```
var1[i] := 8 ;
var2.otto[4] := var3 ;
var4[1+i+j*5] := 4 ;
```

Opérateurs

Présentation

Un opérateur est un symbole pour :
- une opération arithmétique à effectuer ou
- une opération logique à exécuter ou
- un traitement de fonction (appel)

Les opérateurs sont génériques, ce qui signifie qu'ils s'adaptent automatiquement au type de données de l'opérande.

Tableau des opérateurs

Les opérateurs sont exécutés en fonction de leur rang, voir également *Ordre d'exécution, page 510*.

Opérateurs du langage ST :

Opérateur	Signification	Rang	Opérandes possibles	Description

`()`	Mise entre parenthèses	1 (le plus haut)	Expression	La mise entre parenthèses est utilisée pour modifier la séquence d'exécution des opérateurs.
				Exemple : Si les opérandes `A`, `B`, `C` et `D` ont respectivement les valeurs 1, 2, 3 et 4, alors
				`A+B-C*D`
				donne le résultat -9 et
				`(A+B-C)*D`
				donne le résultat 0.

`FUNCNAME` (liste des paramètres réels)	Traitement de fonction (appel)	2	Expression, valeur littérale, variable, adresse (tous les types de données)	Le traitement de fonction est utilisé pour exécuter des fonctions (voir *Appel de fonctions élémentaires, page 542*).
-	Négation	3	Expression, valeur littérale, variable, adresse du type de données `INT`, `DINT`, `UINT`, `UDINT` ou `REAL`	Dans le cas de la négation -, le système change le signe de la valeur de l'opérande. **Exemple :** Dans l'exemple, `OUT` a la valeur -4 si `IN1` est égal à 4. `OUT := - IN1 ;`
`NON`	Complément	3	Expression, valeur littérale, variable, adresse du type de données `BOOL`, `BYTE`, `WORD` ou `DWORD`	Dans le cas de `NOT`, le système effectue une inversion de chaque bit de l'opérande. **Exemple :** Dans l'exemple, `OUT` a la valeur 0011001100 si `IN1` est égal à 1100110011. `OUT := NOT IN1 ;`

Opérateur	Signification	Rang	Opérandes possibles	Description
**	Elévation de puissance	4	Expression, valeur littérale, variable, adresse du type de données REAL (base) et INT, DINT, UINT, UDINT ou REAL (exposant)	Dans le cas de l'élévation à une puissance plus haute **, la valeur du premier opérande (base) est augmentée de la valeur du second opérande (exposant). **Exemple :** Dans l'exemple, OUT est égal à 625,0 si IN1 est 5,0 et IN2 4,0. `OUT := IN1 ** IN2 ;`
*	Multiplication	5	Expression, valeur littérale, variable, adresse du type de données INT, DINT, UINT, UDINT ou REAL	Dans le cas de la multiplication *, la valeur du premier opérande est multipliée par la valeur du deuxième opérande. **Exemple :** Dans l'exemple, OUT est égal à 20,0 si IN1 est 5,0 et IN2 4,0. `OUT := IN1 * IN2 ;` **Remarque :** La fonction MULTIME de la bibliothèque obsolète est destinée aux multiplications du type de données Time.
/	Division	5	Expression, valeur littérale, variable, adresse du type de données INT, DINT, UINT, UDINT ou REAL	Dans le cas de la division /, la valeur du premier opérande est divisée par la valeur du deuxième opérande. **Exemple :** Dans l'exemple, OUT est égal à 4,0 si IN1 est 20,0 et IN2 5,0. `OUT := IN1 / IN2 ;` **Remarque :** La fonction DIVTIME de la bibliothèque obsolète est destinée aux divisions du type de données Time.
MOD	Modulo	5	Expression, valeur littérale, variable, adresse du type de données INT, DINT, UINT ou UDINT	Dans le cas de MOD, la valeur du premier opérande est divisée par la valeur du deuxième opérande et le reste de la division (Modulo) est sorti comme résultat. **Exemple :** Dans l'exemple donné, • OUT est 1 si IN1 est 7 et IN2 2 • OUT est 1, si IN1 est 7 et IN2 -2 • OUT est -1, si IN1 est -7 et IN2 2 • OUT est -1, si IN1 est -7 et IN2 -2 `OUT := IN1 MOD IN2 ;`
+	Addition	6	Expression, valeur littérale, variable, adresse du type de données INT, DINT, UINT, UDINT, REAL ou TIME	Dans le cas de l'addition +, la valeur du premier opérande est ajoutée à la valeur du deuxième opérande. **Exemple :** Dans l'exemple donné, OUT est égal à 9 si IN1 est 7 et IN2 2. `OUT := IN1 + IN2 ;`

Opérateur	Signification	Rang	Opérandes possibles	Description
-	Soustraction	6	Expression, valeur littérale, variable, adresse du type de données INT, DINT, UINT, UDINT, REAL ou TIME	Dans le cas de la soustraction -, la valeur du deuxième opérande est soustraite à la valeur du premier opérande. **Exemple :** Dans l'exemple, OUT est égal à 6 si IN1 est 10 et IN2 4. `OUT := IN1 - IN2 ;`
<	Comparaison "inférieur à"	7	Expression, valeur littérale, variable, adresse du type de données BOOL, BYTE, INT, DINT, UINT, UDINT, REAL, TIME, WORD, DWORD, STRING, DT, DATE ou TOD	< permet de comparer la valeur du premier opérande à celle du deuxième opérande. Si la valeur du premier opérande est inférieure à celle du second, le résultat est un 1 booléen. Si la valeur du premier opérande est supérieure ou égale à celle du second, le résultat est un 0 booléen. **Exemple :** Dans l'exemple, OUT est égal à 1 si IN1 est inférieur à 10. Sinon, il est sur 0. `OUT := IN1 < 10 ;`
>	Comparaison "supérieur à"	7	Expression, valeur littérale, variable, adresse du type de données BOOL, BYTE, INT, DINT, UINT, UDINT, REAL, TIME, WORD, DWORD, STRING, DT, DATE ou TOD	> permet de comparer la valeur du premier opérande à celle du deuxième opérande. Si la valeur du premier opérande est supérieure à celle du second, le résultat est un 1 booléen. Si la valeur du premier opérande est inférieure ou égale à celle du second, le résultat est un 0 booléen. **Exemple :** Dans l'exemple, OUT est égal à 1 si IN1 est supérieur à 10. Si IN1 est inférieur à 0, alors il est sur 0. `OUT := IN1 > 10 ;`
<=	Comparaison "inférieur ou égal à"	7	Expression, valeur littérale, variable, adresse du type de données BOOL, BYTE, INT, DINT, UINT, UDINT, REAL, TIME, WORD, DWORD, STRING, DT, DATE ou TOD	<= permet de comparer la valeur du premier opérande à celle du deuxième opérande. Si la valeur du premier opérande est inférieure ou égale à celle du second, le résultat est un 1 booléen. Si la valeur du premier opérande est supérieure à celle du second, le résultat est un 0 booléen. **Exemple :** Dans l'exemple, OUT est égal à 1 si IN1 est inférieur ou égal à 10. Sinon, il est égal à 0. `OUT := IN1 <= 10 ;`

Opérateur	Signification	Rang	Opérandes possibles	Description
>=	Comparaison "supérieur ou égal à"	7	Expression, valeur littérale, variable, adresse du type de données BOOL, BYTE, INT, DINT, UINT, UDINT, REAL, TIME, WORD, DWORD, STRING, DT, DATE ou TOD	>= permet de comparer la valeur du premier opérande à celle du deuxième opérande. Si la valeur du premier opérande est supérieure ou égale à celle du second, le résultat est un 1 booléen. Si la valeur du premier opérande est inférieure à celle du second, le résultat est un 0 booléen. **Exemple :** Dans l'exemple, OUT est égal à 1 si IN1 est supérieur ou égal à 10. Sinon, il est égal à 0. `OUT := IN1 >= 10 ;`
=	Egalité	8	Expression, valeur littérale, variable, adresse du type de données BOOL, BYTE, INT, DINT, UINT, UDINT, REAL, TIME, WORD, DWORD, STRING, DT, DATE ou TOD	= permet de comparer la valeur du premier opérande à celle du deuxième opérande. Si la valeur du premier opérande est égale à celle du second, le résultat est un 1 booléen. Si la valeur du premier opérande est différente de celle du second, le résultat est un 0 booléen. **Exemple :** Dans l'exemple, OUT est égal à 1 si IN1 est égal à 10. Sinon, il est égal à 0. `OUT := IN1 = 10 ;`
<>	Inégalité	8	Expression, valeur littérale, variable, adresse du type de données BOOL, BYTE, INT, DINT, UINT, UDINT, REAL, TIME, WORD, DWORD, STRING, DT, DATE ou TOD	<> permet de comparer la valeur du premier opérande à celle du deuxième opérande. Si la valeur du premier opérande est différente de celle du second, le résultat est un 1 booléen. Si la valeur du premier opérande est égale à celle du second, le résultat est un 0 booléen. **Exemple :** Dans l'exemple, OUT est égal à 1 si IN1 n'est pas égal à 10. Sinon, il est sur 0. `OUT := IN1 <> 10 ;`
&	ET logique	9	Expression, valeur littérale, variable, adresse du type de données BOOL, BYTE, WORD ou DWORD	& permet d'établir une liaison ET logique entre les opérandes. Pour les types de données BYTE, WORD et DWORD, le lien est fait par bit. **Exemple :** Dans les exemples, OUT est égal à 1 si IN1, IN2 et IN3 sont sur 1. `OUT := IN1 & IN2 & IN3 ;`
AND	ET logique	9	Expression, valeur littérale, variable, adresse du type de données BOOL, BYTE, WORD ou DWORD	AND permet d'établir une liaison ET logique entre les opérandes. Pour les types de données BYTE, WORD et DWORD, le lien est fait par bit. **Exemple :** Dans les exemples, OUT est égal à 1 si IN1, IN2 et IN3 sont sur 1. `OUT := IN1 AND IN2 AND IN3 ;`

Opérateur	Signification	Rang	Opérandes possibles	Description

XOR	OU exclusif logique	10	Expression, valeur littérale, variable, adresse du type de données BOOL, BYTE, WORD ou DWORD	XOR permet d'établir une liaison OU exclusif logique entre les opérandes. Pour les types de données BYTE, WORD et DWORD, le lien est fait par bit.
				Exemple : Dans l'exemple, OUT est sur 1 si IN1 et IN2 ne sont pas égaux. Si IN1 et IN2 ont le même état (tous deux 0 ou 1), OUT est sur 0.
				`OUT := IN1 XOR IN2 ;`
				Si plus de deux opérandes sont reliés, le résultat de l'opération est à l'état 1 pour un nombre impair d'états 1 et à l'état 0 pour un nombre pair d'états 1.
				Exemple : Dans l'exemple, OUT est égal à 1 si 1 ou 3 opérandes sont sur 1. OUT est sur 0 si 0, 2 ou 4 opérandes sont sur 1.
				`OUT := IN1 XOR IN2 XOR IN3 XOR IN4 ;`

OR	OU logique	11 (le plus bas)	Expression, valeur littérale, variable, adresse du type de données BOOL, BYTE, WORD ou DWORD	OR permet d'établir une liaison OU logique entre les opérandes. Pour les types de données BYTE, WORD et DWORD, le lien est fait par bit. **Exemple :** Dans l'exemple, OUT est sur 1 si IN1, IN2 ou IN3 est sur 1. `OUT := IN1 OR IN2 OR IN3 ;`

15.2 Instructions

Objet de ce sous-chapitre

Ce sous-chapitre décrit les instructions du langage de programmation Littéral structuré ST.

Contenu de ce sous-chapitre

Ce sous-chapitre contient les sujets suivants :

Instructions

Description

Les instructions sont les "commandes" du langage de programmation ST.

Les instructions doivent être terminées par des points-virgules.

Une ligne peut contenir plusieurs instructions (séparées par des points-virgules).

Un seul point-virgule représente une instruction d'espacement *(voir page 538)*.

Affectation

Présentation

L'affectation remplace la valeur courante d'une variable à élément unique ou multiple par le résultat de l'évaluation d'une expression.

Une affectation est composée d'une indication de variables à gauche, suivie de l'opérateur d'affectation `:=`, suivi de l'expression à évaluer.

Les deux variables (côtés gauche et droit de l'opérateur d'affectation) doivent être du même type de données.

Les variables ARRAY font exception. A l'issue de l'activation explicite de l'option correspondante, l'affectation de deux variables ARRAY ayant des longueurs différentes est possible.

Affectation de la valeur d'une variable à une autre variable

Les affectations sont utilisées pour affecter la valeur d'une variable à une autre variable.

L'instruction

```
A := B ;
```

est par exemple utilisée pour remplacer la valeur de la variable `A` par la valeur courante de la variable `B`. Si `A` et `B` ont un type de données élémentaire, la valeur individuelle de `B` est transmise vers `A`. Si `A` et `B` ont un type de données dérivé, les valeurs de tous les éléments de `B` sont transmises vers `A`.

Affectation d'une valeur littérale à une variable

Les affectations sont utilisées pour affecter une valeur littérale à une variable.

L'instruction

```
C := 25 ;
```

est par exemple utilisée pour affecter la valeur 25 à la variable `C`.

Affectation de la valeur d'une opération à une variable

Les affectations sont utilisées pour affecter à une variable une valeur qui est le résultat d'une opération.

L'instruction

```
X := (A+B-C)*D ;
```

est par exemple utilisée pour affecter à la variable `X` le résultat de l'opération `(A+B-C)*D`.

Affectation de la valeur d'un FFB à une variable

Les affectations sont utilisées pour affecter à une variable une valeur renvoyée par une fonction ou un bloc fonction.

L'instruction

```
B := MOD(C,A) ;
```

est par exemple utilisée pour appeler la fonction `MOD` (modulo) et affecter le résultat du calcul à la variable `B`.

L'instruction

```
A := MY_TON.Q ;
```

est par exemple utilisée pour affecter à la variable `A` la valeur de la sortie `Q` du bloc fonction `MY_TON` (instance du bloc fonction `TON`). (Il ne s'agit pas d'un appel de bloc fonction))

Affectations multiples

Les affectations multiples sont une extension de la norme CEI 61131-3 et doivent être activées de manière explicite.

Même à l'issue de l'activation, les affectations multiples ne sont PAS autorisées dans les cas suivants :

- dans la liste de paramètres d'un appel de bloc fonction
- dans la liste d'éléments pour l'initialisation de variables structurées

L'instruction

```
X := Y := Z
```

est permise.

Les instructions

```
FB(in1 := 1, In2 := In3 := 2) ;
```

et

```
strucVar := (comp1 := 1, comp2 := comp3 := 2) ;
```

ne sont pas permises.

Affectations entre variables ARRAY et WORD/DWORD.

Les affectations entre variables ARRAY et WORD/DWORD ne sont possibles que si une conversion de types a été effectuée au préalable, par ex. :

```
%Q3.0:16 := INT_TO_AR_BOOL(%MW20) ;
```

Les fonctions de conversion suivantes sont disponibles (bibliothèque générale, famille ARRAY) :

- MOVE_BOOL_AREBOOL
- MOVE_WORD_ARWORD
- MOVE_DWORD_ARDWORD
- MOVE_INT_ARINT
- MOVE_DINT_ARDINT
- MOVE_REAL_ARREAL

Sélectionner l'instruction IF...THEN...END_IF

Description

L'instruction IF signifie qu'une instruction ou un groupe d'instructions peuvent être seulement exécutés si l'expression booléenne correspondante a la valeur 1 (vrai). Si la condition a pour valeur 0 (faux), l'instruction ou le groupe d'instructions ne sont pas exécutés.

L'instruction `THEN` marque la fin d'une condition et le début d'une instruction (des instructions).

L'instruction `END_IF` marque la fin de l'instruction (des instructions).

NOTE : Vous pouvez imbriquer autant d'instructions `IF...THEN...END_IF` que vous voulez pour créer des instructions de sélection complexes.

Exemple `IF...THEN...END_IF`

La condition peut être exprimée via une variable booléenne.

Si `FLAG` a la valeur 1, les instructions sont exécutées, si `FLAG` a la valeur 0, elles ne sont pas exécutées.

```
IF FLAG THEN
   C:=SIN(A) * COS(B) ;
   B:=C - A ;
END_IF ;
```

La condition peut également être exprimée via une opération qui livre un résultat booléen.

Si `A` est supérieur à `B`, les instructions sont exécutées ; si `A` est inférieur ou égal à `B`, elles ne sont pas exécutées.

```
IF A>B THEN
   C:=SIN(A) * COS(B) ;
   B:=C - A ;
END_IF ;
```

Exemple `IF NOT...THEN...END_IF`

`NOT` permet d'inverser la condition (les deux instructions sont exécutées si le résultat est 0).

```
IF NOT FLAG THEN
   C:=SIN_REAL(A) * COS_REAL(B) ;
   B:=C - A ;
END_IF ;
```

Voir également

ELSE *(voir page 526)*

ELSIF *(voir page 527)*

Sélectionner l'instruction ELSE

Description

L'instruction ELSE vient toujours après une instruction IF...THEN, ELSIF...THEN ou CASE.

Si l'instruction ELSE vient après IF ou ELSIF, l'instruction ou le groupe d'instructions sont exécutés seulement si les expressions booléennes correspondantes des instructions IF et ELSIF ont la valeur 0 (faux). Si la condition de l'instruction IF ou ELSIF est 1 (vrai), l'instruction ou le groupe d'instructions ne sont pas exécutés.

Si l'instruction ELSE vient après CASE, l'instruction ou le groupe d'instructions ne sont exécutés que si aucun repère ne contient la valeur du sélecteur. Si un repère contient la valeur du sélecteur, l'instruction ou le groupe d'instructions ne sont pas exécutés.

NOTE : Vous pouvez imbriquer autant d'instructions IF...THEN...ELSE...END_IF que vous voulez pour créer des instructions de sélection complexes.

Exemple ELSE

```
IF A>B THEN
   C:=SIN(A) * COS(B) ;
   B:=C - A ;
ELSE
   C:=A + B ;
   B:=C * A ;
END_IF ;
```

Voir également

IF *(voir page 524)*

ELSIF *(voir page 527)*

CASE *(voir page 529)*

Instruction de sélection ELSIF...THEN

Description

L'instruction ELSIF vient toujours après une instruction IF...THEN. L'instruction ELSIF détermine qu'une instruction ou un groupe d'instructions sont exécutés seulement si l'expression booléenne correspondante de l'instruction IF a la valeur 0 (faux) et que l'expression booléenne correspondante de l'instruction ELSIF a la valeur 1 (vrai). Si la condition de l'instruction IF est 1 (vrai) ou que la condition de l'instruction ELSIF est 0 (faux), l'instruction ou le groupe d'instructions ne sont pas exécutés.

L'instruction THEN caractérise la fin de la (des) condition(s) ELSIF et le début de l'instruction (des instructions).

NOTE : Vous pouvez imbriquer autant d'instructions IF...THEN...ELSIF...THEN...END_IF que vous voulez pour créer des instructions de sélection complexes.

Exemple ELSIF...THEN

```
IF A>B THEN
   C:=SIN(A) * COS(B) ;
   B:=SUB(C,A) ;
ELSIF A=B THEN
   C:=ADD(A,B) ;
   B:=MUL(C,A) ;
END_IF ;
```

Exemple d'instructions imbriquées

```
IF A>B THEN
     IF B=C THEN
         C:=SIN(A) * COS(B) ;
     ELSE
         B:=SUB(C,A) ;
     END_IF ;
ELSIF A=B THEN
     C:=ADD(A,B) ;
     B:=MUL(C,A) ;
ELSE
     C:=DIV(A,B) ;
END_IF ;
```

Voir également

IF *(voir page 524)*

ELSE *(voir page 526)*

Sélection de l'instruction CASE...OF...END_CASE

Description

L'instruction CASE est composée d'une expression de type données INT (le "sélecteur") et d'une liste de groupes d'instructions. Chaque groupe porte un repère composé d'un ou de plusieurs entiers (INT, DINT, UINT, UDINT) ou de plages de valeurs entières. L'on exécutera le premier groupe d'instructions dont le repère contient la valeur calculée du sélecteur. Sinon aucune des instructions n'est exécutée.

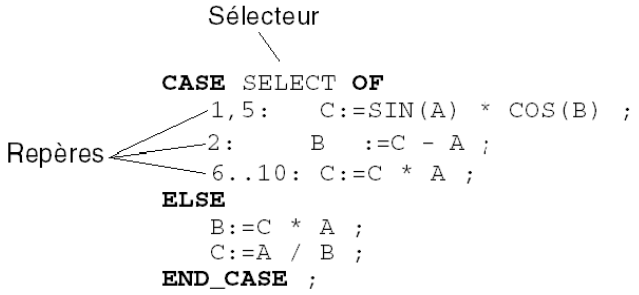
L'instruction OF caractérise le début des repères.

A l'intérieur d'une instruction CASE, on peut définir une instruction ELSE dont les instructions seront exécutées si aucun repère ne contient la valeur du sélecteur.

L'instruction END_CASE marque la fin de l'instruction (des instructions).

Exemple CASE...OF...END_CASE

Exemple CASE...OF...END_CASE



Voir également

ELSE *(voir page 526)*

Instruction récurrente FOR...TO...BY...DO...END_FOR

Description

L'instruction FOR est utilisée si le nombre d'occurrences peut être défini à l'avance. Sinon, on utilise WHILE *(voir page 533)* ou REPEAT *(voir page 534)*.

L'instruction FOR reprend une chaîne d'instructions jusqu'à l'instruction END_FOR. Le nombre d'occurrences est déterminé par la valeur initiale, la valeur finale et la variable de commande.

Variable de commande, valeur initiale et valeur finale doivent avoir le même type de données (DINT ou INT).

Variable de commande, variable initiale et variable finale peuvent être modifiées via une des instructions récurrentes. Cela constitue un complément de la norme CEI 61131-3.

L'instruction FOR incrémente la valeur des variables de commande d'une valeur initiale à une valeur finale. Par défaut, la valeur de l'incrément est définie sur 1. Pour le cas où une autre valeur doit être utilisée, il est possible d'indiquer explicitement une valeur d'incrément (variable ou constante). La valeur des variables de commande est contrôlée avant chaque nouveau parcours de la boucle. La boucle est abandonnée si ladite valeur est en dehors de la plage de la valeur initiale et de la valeur finale.

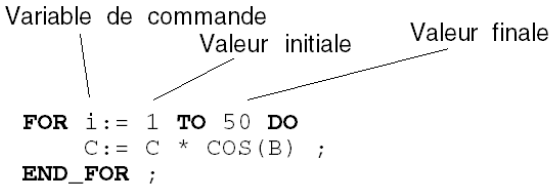
Avant le premier parcours de la boucle, on contrôlera si l'incrément des variables de commande progresse vers la valeur finale à partir de la valeur initiale. Si ce n'est pas le cas (par exemple valeur initiale ≤valeur finale et incrément négatif), la boucle n'est pas traitée. La valeur de la variable de commande n'est pas définie en dehors de la boucle.

L'instruction DO marque la fin de définition d'une occurrence et le début d'une instruction (des instructions).

La répétition peut être prématurément quittée avec l'instruction EXIT. L'instruction END_FOR marque la fin de l'instruction (des instructions).

Exemple : FOR avec incrément 1

FOR avec incrément 1

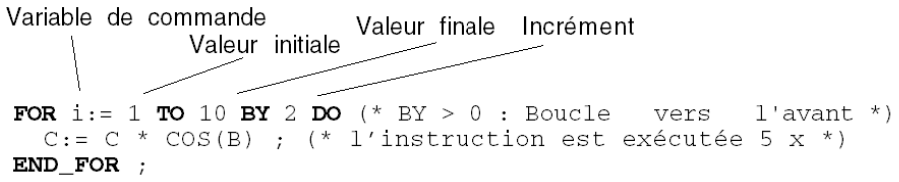


FOR avec incrément différent de 1

Si un autre incrément que 1 doit être utilisé vous pouvez le définir avec BY. Incrément, valeur initiale, valeur finale et variable de commande doivent avoir le même type de données (DINT ou INT). Le critère qui détermine le sens de traitement (comptage, décomptage) est le signe de l'expression BY. Si cette expression est positive, la boucle est comptée, si elle est négative, la boucle est décomptée.

Exemple : Comptage à deux étapes

comptage à deux étapes



Exemple : Décomptage

décomptage

```
FOR i:= 10 TO 1 BY -1 DO (* BY < 0 : boucle décomptée *)
  C:= C * COS(B) ; (* Instr. est exécutée 10 x *)
END_FOR ;
```

Exemple : Boucles "uniques"

Les boucles dans l'exemple sont parcourues exactement une fois puisque valeur initiale = valeur finale. Et ce n'est pas important si l'incrément est positif ou négatif.

```
FOR i:= 10 TO 10 DO (* Boucle unique *)
  C:= C * COS(B) ;
END_FOR ;
```

ou

```
FOR i:= 10 TO 10 BY -1 DO (* Boucle unique *)
  C:= C * COS(B) ;
END_FOR ;
```

Exemple : Boucles critiques

Si l'incrément dans l'exemple est `j` > 0, alors les instructions sont exécutées.

Si `j` < est 0, les instructions ne sont pas exécutées car la situation Valeur initiale< Valeur finale ne permet qu'un incrément ≥ 0.

Si `j` = 0, les instructions sont exécutées et on obtient une boucle continue car avec un incrément de 0 la valeur finale n'est jamais atteinte.

```
FOR i:= 1 TO 10 BY j DO
  C:= C * COS(B) ;
END_FOR ;
```

Instruction de répétition WHILE...DO...END_WHILE

Description

L'instruction WHILE provoque l'exécution répétée d'une chaîne d'instructions jusqu'à ce que l'expression booléenne correspondante soit 0 (fausse). Si l'expression est fausse dès le départ, le groupe d'instructions n'est pas exécuté.

L'instruction DO marque la fin de définition d'une occurrence et le début d'une instruction (des instructions).

La répétition peut être prématurément quittée avec l'instruction EXIT.

L'instruction END_WHILE marque la fin de l'instruction (des instructions).

Dans les cas suivants WHILE ne doit pas être utilisé car cela forme une boucle continue qui entraîne un défaut bloquant du programme :

- WHILE ne doit pas servir à effectuer une synchronisation entre processus, par ex. une "boucle d'attente" avec une condition de fin définie en externe.
- WHILE ne doit pas être utilisé dans un algorithme pour lequel la satisfaction de la condition de fin de la boucle ou l'exécution d'une instruction EXIT ne peuvent pas être garanties.

Exemple WHILE...DO...END_WHILE

```
x := 1; WHILE x <= 100 DO x := x + 4; END_WHILE ;
```

Voir également

EXIT *(voir page 535)*

Instruction récurrente REPEAT...UNTIL...END_REPEAT

Description

L'instruction REPEAT provoque la répétition d'une chaîne d'instructions (au moins une fois) jusqu'à ce que la condition booléenne correspondante soit 1 (vraie).

L'instruction UNTIL marque la condition de fin.

La répétition peut être prématurément quittée avec l'instruction EXIT.

L'instruction END_REPEAT marque la fin de l'instruction (des instructions).

Dans les cas suivants REPEAT ne doit pas être utilisé car cela forme une boucle continue qui entraîne un défaut bloquant du programme :

- REPEAT ne doit pas servir à effectuer une synchronisation entre processus, par ex. une "boucle d'attente" avec une condition finale définie en externe.
- REPEAT ne doit pas être utilisé dans un algorithme pour lequel la satisfaction de la condition de fin de la boucle ou l'exécution d'une instruction EXIT ne peuvent pas être garanties.

Exemple REPEAT...UNTIL...END_REPEAT

```
x := -1
REPEAT
          x := x + 2
   UNTIL x >= 101
END_REPEAT ;
```

Voir également

EXIT *(voir page 535)*

Instruction récurrente EXIT

Description

L'instruction EXIT est utilisée pour terminer les instructions récurrentes (FOR, WHILE, REPEAT) avant que la condition de fin ne soit atteinte.

Si l'instruction EXIT se trouve dans une occurrence imbriquée, la boucle la plus interne (dans laquelle se trouve EXIT) est abandonnée. Et l'instruction suivante qui est exécutée est la première instruction après la fin de la boucle (END_FOR, END_WHILE ou END_REPEAT).

Exemple EXIT

Si FLAG a la valeur 0, SUM est 15 après exécution des instructions.

Si FLAG a la valeur 1, SUM est 6 après exécution des instructions.

```
SUM : = 0 ;
FOR I := 1 TO 3 DO
   FOR J := 1 TO 2 DO
      IF FLAG=1 THEN EXIT;
      END_IF ;
      SUM := SUM + J ;
   END_FOR ;
   SUM := SUM + I ;
END_FOR
```

Voir également

CASE *(voir page 529)*

WHILE *(voir page 533)*

REPEAT *(voir page 534)*

Appel de sous-programme

Appel de sous-programme

L'appel d'un sous-programme comprend le nom de la section du sous-programme suivi d'une liste de paramètres vide.

Les appels de sous-programmes ne fournissent pas de valeurs de retour.

Le sous-programme à appeler doit se trouver dans la même tâche que la section ST appelante.

Il est possible d'appeler des sous-programmes au sein de sous-programmes.

par ex.

```
Nom du sous-programme () ;
```

Les appels de sous-programme sont un complément de la norme CEI 61131-3 et doivent être activés de manière explicite.

Dans les sections d'actions SFC, les appels de sous-programmes ne sont autorisés que si le mode Multitoken a été activé.

RETURN

Description

Des instructions RETURN peuvent être utilisées dans des blocs de fonction dérivés DFB et des SR (sous-programmes).

Des instructions RETURN ne peuvent pas être utilisées dans le programme principal.

- Dans un DFB, une instruction RETURN force le retour au programme qui a appelé le DFB.
  - Le reste de la section de DFB contenant l'instruction RETURN n'est pas exécuté.
  - Les sections suivantes du DFB ne sont pas exécutées.

  Le programme qui a appelé le DFB sera exécuté après retour du DFB.
  Si le DFB est appelé par un autre DFB, le DFB appelant sera exécuté après le retour.
- Dans un SR, une instruction RETURN force le retour au programme qui a appelé le SR.
  - Le reste du SR contenant l'instruction RETURN n'est pas exécuté.

  Le programme qui a appelé le SR sera exécuté après retour du SR.