Liste d'instructions (IL)

14

Objet de ce sous-chapitre

Ce chapitre décrit le langage de programmation Liste d'instructions IL conforme à la norme CEI 61131.

Contenu de ce chapitre

Ce chapitre contient les sous-chapitres suivants :

14.1 Remarques générales sur le langage liste d’instructions IL

Objet de ce sous-chapitre

Ce sous-chapitre vous donne un aperçu général du langage liste d’instructions IL.

Contenu de ce sous-chapitre

Ce sous-chapitre contient les sujets suivants :

Informations générales sur la liste d'instructions IL

Présentation

Le langage de programmation Liste d'instructions (IL) vous permet par exemple d'appeler des fonctions et des blocs fonction de façon conditionnelle ou inconditionnelle, de lancer des affectations et d'exécuter des sauts conditionnels ou inconditionnels au sein de la section.

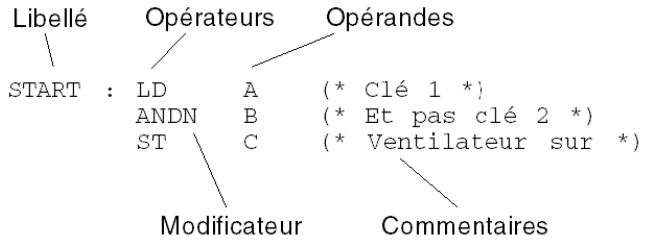
Instructions

Une liste d'instructions se compose d'une chaîne d'instructions.

Chaque instruction commence dans une nouvelle ligne et se compose :
- d'un opérateur *(voir page 469)*,
- le cas échéant, d'un modificateur *(voir page 467)* et,
- si nécessaire un ou plusieurs opérandes *(voir page 464)*

Si plusieurs opérandes sont utilisés, ceux-ci sont séparés par des virgules. Il est possible qu'un libellé *(voir page 480)* soit au début de l'instruction. Ce libellé est suivi par deux points. Un commentaire *(voir page 482)* peut suivre l'instruction.

Exemple :



Structure du langage de programmation

IL est un langage de programmation dit orienté accumulateur, ce qui signifie que chaque instruction utilise ou modifie le contenu de l'accumulateur (une sorte de mémoire temporaire). CEI 61131 désigne cet accumulateur comme "résultat".

Pour cette raison, une liste d'instructions doit toujours commencer par l'opérande `LD` ("Charger en commande accumulateur").

Exemple d'addition :

Commande	Signification
`LD 10`	La valeur 10 chargée dans l'accumulateur.
`ADD 25`	Le contenu de l'accumulateur est additionné à 25.
`ST A`	Le résultat est stocké dans la variable `A`. Le contenu de la variable `A` et de l'accumulateur est désormais 35. Toute nouvelle instruction fonctionnera avec le contenu de l'accumulateur "35" si elle ne commence pas avec `LD`.

Les opérations de comparaison se rapportent aussi toujours à l'accumulateur. Le résultat booléen de la comparaison est stocké dans l'accumulateur et devient ainsi son contenu actuel.

Exemple de comparaison :

Commande	Signification
`LD B`	La valeur `B` est chargée dans l'accumulateur.
`GT 10`	Le contenu de l'accumulateur est comparé à 10.
`ST A`	Le résultat de la comparaison est stocké dans la variable `A`. Si `B` est inférieur ou égal à 10, la valeur de la variable `A` et du contenu de l'accumulateur est 0 (FALSE). Si `B` est supérieur à 10, la valeur de la variable `A` et du contenu de l'accumulateur est 1 (TRUE).

Taille de la section

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

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

Syntaxe

Les identificateurs et les mots-clés ne sont pas sensibles à la casse.

Les caractères d'espacement et de tabulation n'ont aucune influence sur la syntaxe et ils peuvent être utilisés librement.

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'exécution des instructions s'effectue ligne par ligne, du haut vers le bas. Cette suite de caractères peut être modifiée par mise entre parenthèses.

Si, par exemple, `A`, `B`, `C` et `D` ont les valeurs 1, 2, 3 et 4, et sont calculées comme suit :

```
LD A
ADD B
SUB C
MUL C
ST E
```

le résultat dans `E` sera 0.

Pour un calcul tel que :

```
LD A
ADD B
SUB(
LD C
MUL D
)
ST E
```

le résultat dans `E` sera -9.

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.

Conformité CEI

Pour la description de la conformité CEI du langage IL, voir Conformité CEI *(voir page 657)*.

Opérandes

Présentation

Les opérateurs sont utilisés sur les opérandes.

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

Types de données

L'opérande et le contenu actuel de l'accumulateur doivent obligatoirement avoir le même type de données. 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`.

```
LD  i1
INT_TO_REAL
ADD r4
ST  r3
```

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 :
-
```
LD  timeVar1
DIV dintVar1
ST  timeVar2
```
-
```
LD  timeVar1
MUL intVar1
ST  timeVar2
```
-
```
LD  timeVar1
MUL 10
ST  timeVar2
```

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`	`%MX20`	`BOOL`
`B`	%QB102.3	`BYTE`
`W`	`%KW43`	`INT`
`D`	`%QD100`	`DINT`
`F`	`%MF100`	`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 exemple 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;
LD LocV1[%MW200]
ST UnlocV1[2]

LD t#3s
ST LocV2
```

Accès aux variables de champs

Lors d'un accès aux valeurs d'un tableau (ARRAY), seuls les libellés et les variables du type INT, DINT, UINT et UDINT sont autorisés pour l'index.

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

Exemple : enregistrement d'une valeur d'un tableau

```
LD var1[i]
ST var2.otto[4]
```

Modificateur

Présentation

Les modificateurs influencent l'exécution de l'opérateur (voir *Opérateurs, page 469*).

Tableau des modificateurs

Tableau des modificateurs :

Modificateur	Applicable sur les opérandes du type de données	Description
N	BOOL, BYTE, WORD, DWORD	Le modificateur N est utilisé pour inverser bit à bit la valeur d'un opérande. **Exemple :** Dans l'exemple C est égal à 1, si A est égal à 1 et B est égal à 0. LD A AND**N** B ST C
C	BOOL	Le modificateur C est utilisé pour exécuter l'instruction associée, si la valeur de l'accu est 1 (TRUE). **Exemple :** Dans l'exemple, le saut après START est realize uniquement lorsque A est égal à 1 (TRUE) et B à 1 (TRUE). LD A AND B JMP**C** START
CN	BOOL	Si le modificateur C est combiné avec le modificateur N, l'instruction associée est exécutée seulement si la valeur de l'accumulateur est un 0 booléen (FALSE). **Exemple :** Dans l'exemple, le saut vers START est exécuté seulement si A est 0 (FALSE) et B est 0 (FALSE). LD A AND B JMP**CN** START

Modificateur	Applicable sur les opérandes du type de données	Description
(	toutes	Le modificateur Parenthèse gauche (est utilisé pour repousser l'évaluation de l'opérande, jusqu'à ce que l'opérateur Parenthèse droite) apparaisse. Le nombre d'opérations Parenthèse droite doit être égal au nombre de modificateurs Parenthèse gauche. Il est possible d'imbriquer les parenthèses. **Exemple :** Dans l'exemple, E a pour valeur 1, si C et/ou D sont définis sur 1 et si A et B ont aussi la valeur 1. LD A AND B AND(C OR D) ST E L'exemple peut être programmé de la façon suivante : LD A AND B AND(LD C OR D) ST E

Opérateurs

Introduction

Un opérateur est un symbole pour :

- une opération arithmétique à exécuter,
- une opération logique à exécuter ou
- l'appel d'un bloc fonction élémentaire, d'un DFB ou d'un sous-programme.

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

Opérateurs de chargement et d'enregistrement

Opérateurs de chargement et d'enregistrement du langage IL :

Opérateur	Modificateur	Signification	Opérandes	Description
LD	N (uniquement pour les opérandes du type de données BOOL, BYTE, WORD ou DWORD)	Charge la valeur de l'opérande dans l'accumulateur	Valeur littérale, variable, adresse directe avec type de données quelconque	Avec LD, la valeur d'un opérande est chargée dans l'accumulateur. Les données de l'accumulateur s'adaptent automatiquement au type de données de l'opérande. Cela s'applique également aux types de données dérivés. **Exemple :** Dans l'exemple donné, la valeur de A est chargée dans l'accumulateur, ajoutée à la valeur de B, puis le résultat est enregistré dans E. **LD** A ADD B ST E
ST	N (uniquement pour les opérandes du type de données BOOL, BYTE, WORD ou DWORD)	Enregistre la valeur de l'accumulateur dans l'opérande.	Variable, adresse directe avec type de données au choix	Avec ST, la valeur actuelle de l'accumulateur est enregistrée dans l'opérande. Le type de données de l'opérande doit correspondre au type de données de l'accumulateur. **Exemple :** Dans l'exemple donné, la valeur de A est chargée dans l'accumulateur, ajoutée à la valeur de B, puis le résultat est enregistré dans E. LD A ADD B **ST** E Le fait que ST soit suivi d'un LD ou pas détermine si l'on continue à travailler avec le résultat antérieur. **Exemple :** Dans l'exemple donné, la valeur de A est chargée dans l'accumulateur, ajoutée à la valeur de B, puis le résultat est enregistré dans E. Ensuite la valeur de B est déduite de la valeur de E (contenu actuel de l'accumulateur) et le résultat est enregistré dans C. LD A ADD B **ST** E SUB 3 **ST** C

Opérateurs de paramétrage et de réinitialisation

Opérateurs de paramétrage et de réinitialisation du langage IL

Opérateur	Modificateur	Signification	Opérandes	Description
S	-	Définit l'opérande sur 1 si le contenu de l'accumulateur est 1.	Variable, adresse directe du type de données BOOL	S permet de définir l'opérande sur 1 lorsque le contenu de l'accumulateur actuel est un 1 booléen. **Exemple :** Dans cet exemple, la valeur de A est chargée dans l'accumulateur. Si le contenu de l'accumulateur (valeur de A) est 1, alors OUT est défini sur 1. LD A **S** OUT Cet opérande est habituellement utilisé avec l'opérateur de réinitialisation R. **Exemple :** L'exemple montre un retournement RS (réinitialisation dominante) commandé via les deux variables booléennes A et C. LD A **S** OUT LD C R OUT
R	-	Définit l'opérande sur 0 si le contenu de l'accumulateur est 1.	Variable, adresse directe du type de données BOOL	R permet de définir l'opérande sur 0 lorsque le contenu actuel de l'accumulateur est un 1 booléen. **Exemple :** Dans cet exemple, la valeur de A est chargée dans l'accumulateur. Si le contenu de l'accumulateur (valeur de A) est 1, alors OUT est défini sur 0. LD A **R** OUT Cet opérande est habituellement utilisé avec l'opérateur de paramétrage S. **Exemple :** L'exemple montre un retournement SR (paramétrage dominant) commandé via les deux variables booléennes A et C. LD A **R** OUT LD C S OUT

Opérateurs logiques

Opérateurs logiques du langage IL :

Opérateur	Modificateur	Signification	Opérandes	Description
AND	N, N(, (	ET logique	Valeur littérale, variable, adresse directe du type de données BOOL, BYTE, WORD ou DWORD	L'opérateur AND établit une liaison ET logique entre le contenu de l'accumulateur et l'opérande. Pour les types de données BYTE, WORD et DWORD, la liaison est effectuée par bit. **Exemple :** Dans l'exemple donné, D aura pour valeur 1, si A, B et C sont sur 1. LD A **AND** B **AND** C ST D
OR	N, N(, (	OU logique	Valeur littérale, variable, adresse directe du type de données BOOL, BYTE, WORD ou DWORD	L'opérateur OR établit une liaison OU logique entre le contenu de l'accumulateur et l'opérande. Pour les types de données BYTE, WORD et DWORD, la liaison est effectuée par bit. **Exemple :** Dans l'exemple donné, D aura pour valeur 1, si A ouB est sur 1 et si C est sur 1. LD A **OR** B **OR** C ST D

Opérateur	Modificateur	Signification	Opérandes	Description
XOR	N, N(, (	OU exclusif logique	Valeur littérale, variable, adresse directe du type de données BOOL, BYTE, WORD ou DWORD	L'opérateur XOR établit une liaison OU exclusif logique entre le contenu de l'accumulateur et l'opérande. 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. Pour les types de données BYTE, WORD et DWORD, la liaison est effectuée par bit. **Exemple :** Dans l'exemple, D est égal à 1 si A ouB est défini sur 1. Si A et B ont le même état (tous deux 0 ou 1), D est égal à 0. LD A **XOR** B ST D 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, F a pour valeur 1 si 1 ou 3 opérandes sont définis sur 1. F a pour valeur 0 si 0, 2 ou 4 opérandes sont définis sur 1. LD A **XOR** B **XOR** C **XOR** D **XOR** E ST F
NOT	-	Négation logique (complément)	Contenu d'accumulateur du type de données BOOL, BYTE, WORD ou DWORD	NOT permet d'inverser le contenu de l'accumulateur par bit. **Exemple :** Dans l'exemple donné, B aura pour valeur 1, si A est sur 0 et B a pour valeur 0, si A est sur 1. LD A **NOT** ST B

Opérateurs arithmétiques

Opérateurs arithmétiques du langage IL :

Opérateur	Modificateur	Signification	Opérandes	Description
ADD	(	Addition	Valeur littérale, variable, adresse directe du type de données INT, DINT, UINT, UDINT, REAL ou TIME	ADD permet d'ajouter la valeur de l'opérande à la valeur du contenu de l'accumulateur. **Exemple :** L'exemple correspond à la formule D = A + B + C LD A **ADD** B **ADD** C ST D
SUB	(	Soustraction	Valeur littérale, variable, adresse directe du type de données INT, DINT, UINT, UDINT, REAL ou TIME	SUB permet de retirer la valeur de l'opérande du contenu de l'accumulateur. **Exemple :** L'exemple correspond à la formule D = A - B - C LD A **SUB** B **SUB** C ST D
MUL	(	Multiplication	Valeur littérale, variable, adresse directe du type de données INT, DINT, UINT, UDINT ou REAL	MUL permet de multiplier le contenu de l'accumulateur par la valeur de l'opérande. **Exemple :** L'exemple correspond à la formule D = A * B * C LD A **MUL** B **MUL** C ST D **Remarque :** La fonction MULTIME de la bibliothèque obsolète est destinée aux multiplications du type de données Time.
DIV	(	Division	Valeur littérale, variable, adresse directe du type de données INT, DINT, UINT, UDINT ou REAL	DIV permet de diviser le contenu de l'accumulateur par la valeur de l'opérande. **Exemple :** L'exemple correspond à la formule D = A / B / C LD A **DIV** B **DIV** C ST D **Remarque :** La fonction DIVTIME de la bibliothèque obsolète est destinée aux divisions du type de données Time.

Opérateur	Modificateur	Signification	Opérandes	Description

MOD	(	Division modulo	Valeur littérale, variable, adresse directe du type de données INT, DINT, UINT ou UDINT	MOD permet de diviser la valeur du premier opérande par la valeur du deuxième et de sortir le reste de la division (modulo) comme résultat.
				Exemple : Dans l'exemple donné,
				• C aura pour valeur 1, si A est égal à 7 et B à 2
				• C aura pour valeur 1, si A est égal à 7 et B à -2
				• C aura pour valeur -1, si A est égal à -7 et B à 2
				• C aura pour valeur -1, si A est égal à -7 et B à -2
				`LD A`
				`MOD B`
				`ST C`

Opérateurs de comparaison

Opérateurs de comparaison du langage IL :

Opérateur	Modificateur	Signification	Opérandes	Description

GT	(	Comparaison : >	Valeur littérale, variable, adresse directe du type de données BOOL, BYTE, WORD, DWORD, STRING, INT, DINT, UINT, UDINT, REAL, TIME, DATE, DT ou TOD	GT permet de comparer le contenu de l'accumulateur au contenu de l'opérande. Si le contenu de l'accumulateur est supérieur au contenu de l'opérande, le résultat est un 1 booléen. Si le contenu de l'accumulateur est inférieur ou égal au contenu de l'opérande, le résultat est un 0 booléen.
				Exemple : Dans l'exemple donné, la valeur de D est 1, si D est supérieur à 10. Dans le cas contraire, la valeur de D est 0.
				`LD A`
				`GT 10`
				`ST D`

GE	(	Comparaison : >=	Valeur littérale, variable, adresse directe du type de données BOOL, BYTE, WORD, DWORD, STRING, INT, DINT, UINT, UDINT, REAL, TIME, DATE, DT ou TOD	GE permet de comparer le contenu de l'accumulateur au contenu de l'opérande. Si le contenu de l'accumulateur est supérieur/égal au contenu de l'opérande, le résultat est un 1 booléen. Si le contenu de l'accumulateur est inférieur au contenu de l'opérande, le résultat est un 0 booléen.
				Exemple : Dans l'exemple donné, la valeur de D est 1, si D est supérieur ou égal à 10. Dans le cas contraire, la valeur de D est 0.
				`LD A`
				`GE 10`
				`ST D`

Opérateur	Modificateur	Signification	Opérandes	Description
EQ	(	Comparaison : =	Valeur littérale, variable, adresse directe du type de données BOOL, BYTE, WORD, DWORD, STRING, INT, DINT, UINT, UDINT, REAL, TIME, DATE, DT ou TOD	EQ permet de comparer le contenu de l'accumulateur au contenu de l'opérande. Si le contenu de l'accumulateur est égal à celui de l'opérande, le résultat est un 1 booléen. Si le contenu de l'accumulateur n'est pas égal à celui de l'opérande, le résultat est un 0 booléen. **Exemple :** Dans l'exemple donné, la valeur de D est 1, si A est égal à 10. Dans le cas contraire, la valeur de D est 0. LD A **EQ** 10 ST D
NE	(	Comparaison : <>	Valeur littérale, variable, adresse directe du type de données BOOL, BYTE, WORD, DWORD, STRING, INT, DINT, UINT, UDINT, REAL, TIME, DATE, DT ou TOD	NE permet de comparer le contenu de l'accumulateur au contenu de l'opérande. Si le contenu de l'accumulateur n'est pas égal à celui de l'opérande, le résultat est un 1 booléen. Si le contenu de l'accumulateur est égal à celui de l'opérande, le résultat est un 0 booléen. **Exemple :** Dans l'exemple donné, la valeur de D est 1, si A n'est pas égal à 10. Dans le cas contraire, la valeur de D est 0. LD A **NE** 10 ST D
LE	(	Comparaison : <=	Valeur littérale, variable, adresse directe du type de données BOOL, BYTE, WORD, DWORD, STRING, INT, DINT, UINT, UDINT, REAL, TIME, DATE, DT ou TOD	LE permet de comparer le contenu de l'accumulateur au contenu de l'opérande. Si le contenu de l'accumulateur est inférieur ou égal à celui de l'opérande, le résultat est un 1 booléen. Si le contenu de l'accumulateur est supérieur à celui de l'opérande, le résultat est un 0 booléen. **Exemple :** Dans l'exemple donné, la valeur de D est 1, si D est inférieur ou égal à 10. Dans le cas contraire, la valeur de D est 0. LD A **LE** 10 ST D

Opérateur	Modificateur	Signification	Opérandes	Description
LT	(	Comparaison : <	Valeur littérale, variable, adresse directe du type de données BOOL, BYTE, WORD, DWORD, STRING, INT, DINT, UINT, UDINT, REAL, TIME, DATE, DT ou TOD	LT permet de comparer le contenu de l'accumulateur au contenu de l'opérande. Si le contenu de l'accumulateur est inférieur à celui de l'opérande, le résultat est un 1 booléen. Si le contenu de l'accumulateur est supérieur ou égal à celui de l'opérande, le résultat est un 0 booléen. **Exemple :** Dans l'exemple donné, la valeur de D est 1, si D est inférieur à 10. Dans le cas contraire, la valeur de D est 0. LD A **LT** 10 ST D

Opérateurs d'appel

Opérateurs d'appel du langage IL :

Opérateur	Modificateur	Signification	Opérandes	Description
CAL	C, CN (uniquement si le contenu de l'accumulateur est du type de données BOOL)	Appel d'un bloc fonction, d'un DFB ou d'un sous-programme	Nom d'instance d'un bloc fonction, d'un DFB ou d'un sous-programme	CAL permet d'appeler un bloc fonction, un DFB ou un sous-programme avec ou sans conditions. Voir également *Appel de blocs fonction élémentaires et de blocs fonction dérivés, page 489* et *Appel de sous-programme, page 479*
NOM_DE_FONCTION	-	Exécution d'une fonction	Valeur littérale, variable, adresse directe (le type de données dépend de la fonction)	Le nom de fonction vous permet d'exécuter une fonction. Voir également *Appel de fonctions élémentaires, page 484*
NOM_DE_PROCEDURE	-	Exécution d'une procédure	Valeur littérale, variable, adresse directe (le type de données dépend de la procédure)	Le nom de procédure vous permet d'exécuter une procédure. Voir également *Procédures d'appel, page 500*

Opérateurs de structuration

Opérateurs de structuration du langage IL :

Opérateur	Modificateur	Signification	Opérandes	Description
JMP	C, CN (uniquement si le contenu de l'accumulateur est du type de données BOOL)	Saut vers l'étiquette	ETIQUETTE	JMP permet d'effectuer un saut avec ou sans conditions vers une étiquette. Voir également *Libellés et sauts, page 480*

RET	C, CN (uniquement si le contenu de l'accumulateur est du type de données BOOL)	Retour dans l'unité organisationnelle supérieure suivante du programme	-	Des opérateurs RETURN peuvent être utilisés dans des blocs de fonction dérivés DFB et des SR (sous-programmes).
				Des opérateurs RETURN ne peuvent pas être utilisés dans le programme principal.
				● Dans un DFB, un opérateur RETURN force le retour au programme qui a appelé le DFB.
				  ● Le reste de la section de DFB contenant l'opérateur 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, un opérateur RETURN force le retour au programme qui a appelé le SR.
				  ● Le reste du SR contenant l'opérateur RETURN n'est pas exécuté.
				Le programme qui a appelé le SR sera exécuté après retour du SR.

)	-	Traitement d'opérations placées en attente	-	La parenthèse droite) permet de lancer l'édition de l'opérateur en attente. Le nombre d'opérations Parenthèse droite doit être égal au nombre de modificateurs Parenthèse gauche. Il est possible d'imbriquer les parenthèses. **Exemple :** Dans l'exemple donné, E aura pour valeur 1, si C et/ou D est sur 1 et si A et B sont sur 1. LD A AND B AND(C OR D) ST E

Appel de sous-programme

Appeler un sous-programme

En IL, l'appel d'un sous-programme se compose de l'opérateur `CAL`, suivi du nom de la section de sous-programme, puis d'une liste de paramètres vide (facultative).

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 IL appelante.

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

par exemple:

```
ST A
CAL SubroutineName ()
LD B
```

ou

```
ST A
CAL SubroutineName
LD B
```

Les appels de sous-programmes 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.

Libellés et sauts

Présentation

Les libellés servent de cible à atteindre pour les sauts.

Propriétés des libellés :

Propriétés des étiquettes :

- les étiquettes doivent toujours être le premier élément d'une ligne.
- les étiquettes doivent être uniques dans toute la section, on ne fait pas de distinction ici entre majuscules et minuscules.
- la longueur maximale d'un étiquette est de 32 caractères.
- Les libellés doivent satisfaire aux conventions de désignation CEI.
- les étiquettes sont séparés par deux points : de la commande suivante.
- Les libellés doivent se trouver au début des "expressions" car uniquement des valeurs non-définies peuvent se trouver dans l'accumulateur.
  Exemple :

```
start: LD A
       AND B
       OR C
       ST D
       JMP start
```

Propriétés des sauts :

Propriétés des sauts

- L'opération `JMP` exécute un saut avec ou sans conditions vers un libellé.
- `JMP` peut être utilisé avec les modificateurs `C` et `CN` (uniquement quand le contenu d'accumulateur actuel est du type de données `BOOL`).
- Les sauts sont possibles au sein des sections de DFB et de programme.
- Les sauts ne sont possibles qu'au sein d'une même section.

Des destinations de saut possibles sont :

- la première instruction `LD` d'un appel d'EFB/de DFB avec affectation de paramètres d'entrée (voir `start2`),
- une instruction `LD` "normale" (voir `start1`),
- une instruction `CAL` ne travaillant pas avec l'affectation de paramètres d'entrée (voir `start3`),
- une instruction `JMP` (voir `start4`),
- la fin d'une liste d'instructions (voir `start5`).

Exemple

```
start2: LD A
        ST counter.CU
        LD B
        ST counter.R
        LD C
        ST counter.PV
        CAL counter
        JMPCN start4
start1: LD A
        AND B
        OR C
        ST D
        JMPC start3
        LD A
        ADD E
        JMP start5
start3: CAL counter (
        CU:=A
        R:=B
        PV:=C )
        JMP start1
        LD A
        OR B
        OR C
        ST D
start4: JMPC start1
        LD C
        OR B
start5: ST A
```

Commentaire

Description

Dans l'éditeur IL, les commentaires commencent par la chaîne de caractères `(*` et se terminent par la chaîne de caractères `*)`. Vous pouvez entrer un commentaire quelconque entre ces deux chaînes de caractères.

Selon la norme CEI 61131-3, il n'est pas possible d'imbriquer des commentaires. Toutefois, si des commentaires sont imbriqués, ils doivent être activés de manière explicite.

14.2 Appel de fonctions élémentaires, de blocs fonction élémentaires, de blocs fonction dérivés et de procédures

Objet de ce sous-chapitre

Appel de fonctions élémentaires, de blocs fonction élémentaires, de blocs fonction dérivés et de procédures dans le langage de programmation IL.

Contenu de ce sous-chapitre

Ce sous-chapitre contient les sujets suivants :

Appel de fonctions élémentaires

Utilisation des fonctions

Les fonctions élémentaires sont disponibles sous forme de bibliothèques. La logique des fonctions est créée dans le langage de programmation C et ne peut pas être modifiée dans l'éditeur IL.

Les fonctions n'ont pas d'état interne. Lorsque les valeurs d'entrée sont identiques, la valeur de sortie est la même à chaque exécution de la fonction. Par exemple, l'addition de deux valeurs donne toujours le même résultat. Pour certaines fonctions élémentaires, il est possible d'augmenter le nombre d'entrées.

Les fonctions élémentaires n'ont qu'une seule valeur de renvoi (sortie).

Paramètres

Pour importer des valeurs dans une fonction ou exporter des valeurs d'une fonction, on a besoin d'« entrées » et d'une « sortie ». Ces entrées et cette sortie sont appelées « paramètres formels ».

Les états actuels du processus sont transmis aux paramètres formels. Ce sont les paramètres réels.

On peut utiliser un paramètre réel d'entrée de fonction de type :
- variable,
- adresse,
- valeur littérale.

On peut utiliser un paramètre réel de sortie de fonction de type :
- variable,
- adresse.

Le type des données du paramètre réel doit correspondre au type des données du paramètre formel. La seule exception concerne les paramètres formels génériques dont le type de données est déterminé par le paramètre réel.

De plus, pour les paramètres formels génériques du type de données `ANY_BIT`, des paramètres réels du type de données `INT` ou `DINT` (pas `UINT` ni `UDINT`) peuvent être utilisés.

Il s'agit d'une extension de la norme CEI 61131-3, qui doit être activée de manière explicite.

Exemple :

Autorisé :

```
AND (AnyBitParam := IntVar1, AnyBitParam2 := IntVar2)
```

Non autorisé :

```
AND_WORD (WordParam1 := IntVar1, WordParam2 := IntVar2)
```

(Dans ce cas `AND_INT` doit être utilisé.)

```
AND_ARRAY_WORD (ArrayInt, ...)
```

(Dans ce cas, un changement de type explicite doit être effectué via `INT_ARR_TO_WORD_ARR (...)`.

Pour l'appel formel, il n'est en principe pas nécessaire d'affecter une valeur à tous les paramètres formels. Pour connaître les types de paramètre formel pour lesquels cela est cependant impératif, veuillez vous reporter au tableau.

Type de paramètre	EDT	STRING	ARRAY	ANY_ARRAY	IODDT	STRUCT	FB	ANY
Entrée	-	-	+	+	+	+	+	+
VAR_IN_OUT	+	+	+	+	+	+	/	+
Sortie	-	-	-	-	-	-	/	-
+ paramètre réel impératif								
- paramètre réel non impératif								
/ non applicable								

Si aucune valeur n'est affectée à un paramètre formel, la valeur initiale est utilisée lors de l'exécution de la fonction. Si aucune valeur initiale n'est définie, la valeur par défaut (0) est utilisée.

Remarques sur la programmation

Veuillez tenir compte des remarques qui suivent sur la programmation :

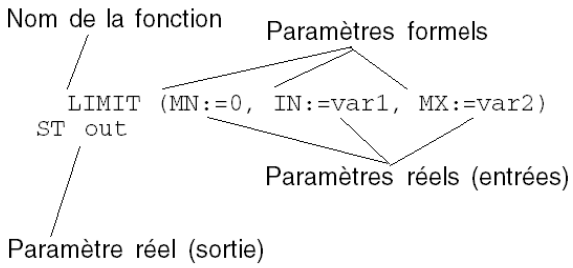
- Les fonctions ne sont exécutées que lorsque l'entrée `EN` = 1 ou lorsque l'entrée `EN` est désactivée (voir aussi EN and ENO *(voir page 488)*).
- Toutes les fonctions génériques sont surchargées. Cela signifie que les fonctions peuvent être appelées avec ou sans la saisie du type de données.
  Ex. :
  ```
  LD i1
  ADD i2
  ST i3
  ```
  est identique à
  ```
  LD i1
  ADD_INT i2
  ST i3
  ```
- Contrairement au langage ST, les fonctions ne peuvent pas être imbriquées dans le langage IL.
- Il existe deux façons d'appeler une fonction :
  - appel formel (appel d'une fonction avec les noms des paramètres formels),
  - appel informel (appel d'une fonction sans les noms des paramètres formels).

Appel formel

Avec ce type d'appel (avec les noms des paramètres formels), les fonctions sont appelées via une suite d'instructions qui comprend le nom de la fonction suivi d'une liste entre parenthèses des affectations de valeur (paramètres réels) aux paramètres formels. L'ordre d'énumération des paramètres formels **n'est pas important**. La liste des paramètres réels peut être éditée directement après une virgule. Après l'exécution de la fonction, le résultat est chargé dans l'accumulateur et peut être sauvegardé via `ST`.

Il est possible d'utiliser `EN` et `ENO` avec ce type d'appel.

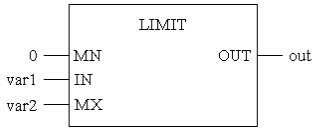
Appel d'une fonction avec les noms des paramètres formels :



ou

```
LIMIT ( MN:=0, IN:=var1, MX:=var2 ) ST out
```

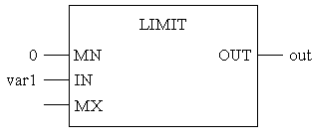
Appel de la même fonction dans FBD :



Lors d'un appel formel, il n'est pas nécessaire d'affecter une valeur à tous les paramètres formels (voir également Parameter *(voir page 484)*).

```
LIMIT (MN:=0, IN:=var1) ST out
```

Appel de la même fonction dans FBD :

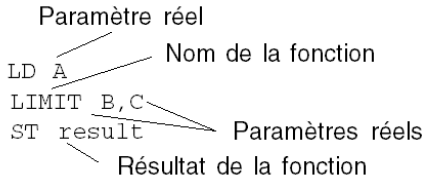


Appel informel

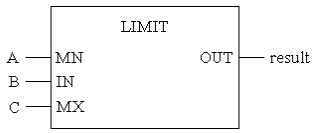
Avec ce type d'appel (sans les noms des paramètres formels), les fonctions sont appelées via une suite d'instructions qui comprend le chargement du premier paramètre réel dans l'accumulateur suivi du nom de la fonction, lui-même suivi d'une liste optionnelle des paramètres réels. L'ordre d'énumération des paramètres réels est **important**. La liste des paramètres réels ne peut pas être éditée. Après l'exécution de la fonction, le résultat est chargé dans l'accumulateur et peut être sauvegardé via `ST`.

`EN` et `ENO` ne peuvent **pas** être utilisés avec ce type d'appel.

Appel d'une fonction avec les noms des paramètres formels :



Appel de la même fonction dans FBD :



NOTE : veuillez noter que pour l'appel informel, la liste des paramètres réels ne doit **pas** être indiquée entre parenthèses. La norme CEI 61133-3 exige dans ce cas d'enlever les parenthèses, afin d'indiquer que le premier paramètre réel ne fait pas partie de la liste.

Appel informel **non valide** d'une fonction :

```
LD A
LIMIT (B,C)
```

Si la valeur à traiter (premier paramètre réel) se trouve déjà dans l'accumulateur, l'instruction de chargement n'est plus nécessaire.

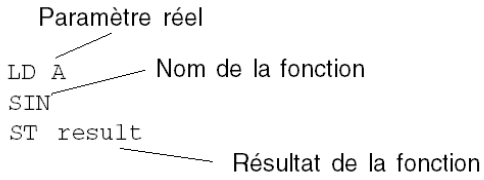
```
LIMIT B,C ST result
```

Si le résultat doit être directement utilisé, l'instruction de sauvegarde n'est plus nécessaire :

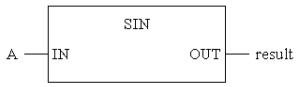
```
LD A LIMIT_REAL B,C MUL E
```

Si la fonction à exécuter n'a qu'une seule entrée, le nom de la fonction n'est pas suivi d'une liste de paramètres réels.

Appel d'une fonction avec un paramètre réel :



Appel de la même fonction dans FBD :



EN et ENO

Pour toutes les fonctions, une entrée EN et une sortie ENO peuvent être configurées.

Si la valeur d'EN est égale à « 0 », lorsque la fonction est appelée, les algorithmes définis par cette dernière ne sont pas exécutés et ENO est mis à « 0 ».

Si la valeur d'EN est égale à « 1 », lorsque la fonction est appelée, les algorithmes définis par la fonction sont exécutés. Une fois les algorithmes exécutés, la valeur de la sortie ENO est réglée sur « 1 ». Si une erreur se produit durant l'exécution de ces algorithmes, ENO est mis à « 0 ».

Si aucune valeur n'est attribuée à la broche EN à l'appel du FFB, l'algorithme défini par ce dernier est exécuté (comme lorsque EN a la valeur « 1 »).

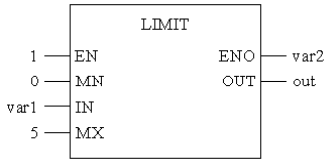
Si ENO est mis à « 0 » (en raison de EN = 0 ou d'une erreur d'exécution), la sortie de la fonction est mise à « 0 ».

Le comportement de sortie de la fonction ne dépend pas de l'appel de la fonction sans EN/ENO ou avec EN=1.

Si EN/ENO doivent être utilisés, l'appel de la fonction doit être exécuté comme un appel formel.

```
LIMIT (EN:=1, MN:=0, IN:=var1, MX:=5, ENO=>var2) ST out
```

Appel de la même fonction dans FBD :



Appel de blocs fonction élémentaires et de blocs fonction dérivés

Bloc fonction élémentaire

Les blocs fonction élémentaires ont des états internes. Pour des valeurs égales aux entrées, la valeur à la sortie peut être différente à chaque exécution du bloc fonction. Pour un compteur, par exemple, la valeur de la sortie est incrémentée.

Les blocs fonction peuvent comprendre plusieurs valeurs de renvoi (sorties).

Bloc fonction dérivé

Les blocs fonction dérivés (DFB) ont les mêmes caractéristiques que les blocs fonction élémentaires. Ils sont cependant créés par l'utilisateur dans les langages FBD, LD, IL et/ou ST.

Paramètres

Pour importer des valeurs dans un bloc fonction ou les exporter d'un bloc fonction, des entrées et des sorties sont nécessaires. Ces entrées et ces sorties sont appelées « paramètres formels ».

Les états actuels du processus sont transmis aux paramètres formels. Ce sont les « paramètres réels ».

Pour les entrées de bloc fonction, on peut utiliser un paramètre réel de type :
- variable,
- adresse,
- valeur littérale.

Pour les sorties de bloc fonction, on peut utiliser un paramètre réel de type :
- variable,
- adresse.

Le type des données du paramètre réel doit correspondre au type des données du paramètre formel. La seule exception concerne les paramètres formels génériques dont le type de données est déterminé par le paramètre réel.

Exception :

Pour les paramètres formels génériques de type de données `ANY_BIT`, des paramètres réels de type de données `INT` ou `DINT` (pas `UINT` ni `UDINT`) peuvent être utilisés.

Il s'agit d'une extension de la norme CEI 61131-3, qui doit être activée de manière explicite.

Exemple :

Autorisé :

```
AND (AnyBitParam := IntVar1, AnyBitParam2 := IntVar2)
```

Non autorisé :

```
AND_WORD (WordParam1 := IntVar1, WordParam2 := IntVar2)
```

(Dans ce cas `AND_INT` doit être utilisé.)

```
AND_ARRAY_WORD (ArrayInt, ...)
```

(Dans ce cas, un changement de type explicite doit être effectué via `INT_ARR_TO_WORD_ARR (...)`.

Il n'est en principe pas nécessaire d'affecter une valeur à tous les paramètres formels. Pour connaître les types de paramètre formel pour lesquels cela est cependant impératif, veuillez vous reporter au tableau.

Type de paramètre	EDT	STRING	ARRAY	ANY_ARRAY	IODDT	STRUCT	FB	ANY
EFB : Entrée	-	+	+	+	/	+	/	+
EFB : VAR_IN_OUT	+	+	+	+	+	+	/	+
EFB : Sortie	-	-	+	+	+	-	/	+
DFB : Entrée	-	+	+	+	/	+	/	+
DFB : VAR_IN_OUT	+	+	+	+	+	+	/	+
DFB : Sortie	-	-	+	/	/	-	/	+
+ paramètre réel impératif								
- paramètre réel non impératif								
/ non applicable								

Si aucune valeur n'est affectée à un paramètre formel, la valeur initiale est utilisée pendant l'exécution du bloc fonction. Si aucune valeur initiale n'est définie, la valeur par défaut (0) est utilisée.

Si aucune valeur n'est affectée à un paramètre formel et que le bloc fonction/DFB ait été instancié à plusieurs reprises, les instances appelées par la suite travaillent avec l'ancienne valeur.

Variables publiques

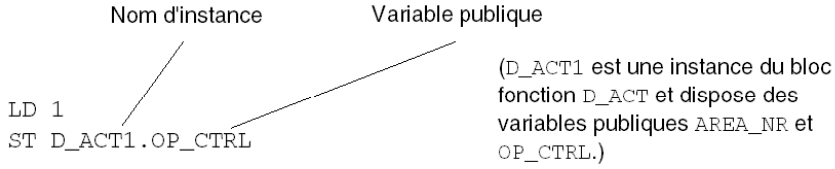
Certains blocs fonction disposent non seulement d'entrées et de sorties, mais également de variables publiques.

Ces variables permettent de transmettre des valeurs statistiques (valeurs non influencées par le procédé) au bloc fonction. Elles sont donc utilisées lors du paramétrage du bloc fonction.

Les variables publiques sont une extension de la norme CEI 61131-3.

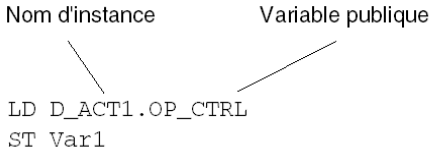
Des valeurs sont affectées aux variables publiques via leur valeur initiale ou par instructions de chargement et d'enregistrement.

Exemple :



Les valeurs des variables publiques sont ensuite lues à partir du nom d'instance du bloc fonction et du nom de la variable publique.

Exemple :



Variables privées

Certains blocs fonction disposent non seulement d'entrées, de sorties et de variables publiques, mais également de variables privées.

A l'instar des variables publiques, ces variables permettent de transmettre des valeurs statistiques (valeurs non influencées par le procédé) au bloc fonction.

Le programme utilisateur ne peut pas accéder à ces variables. Seule la table d'animation en a la capacité.

NOTE : les DFB imbriqués sont déclarés comme des variables privées du DFB parent. Ainsi, leurs variables ne sont pas accessibles via la programmation, mais via la table d'animation.

Les variables privées sont une extension de la norme CEI 61131-3.

Remarques sur la programmation

Veuillez tenir compte des remarques qui suivent sur la programmation :

- Les fonctions ne sont exécutées que lorsque l'entrée EN = 1 ou lorsque l'entrée EN est désactivée (voir aussi EN and ENO *(voir page 497)*).
- L'affectation de variables aux sorties de type ANY ou ARRAY doit être effectuée via l'opérateur => (voir aussi Formal Form of CAL with a List of the Input Parameters *(voir page 492)*).
  Une affectation en dehors du cadre de l'appel de bloc fonction n'est pas possible. L'instruction

```
My_Var := My_SAH.OUT
```
n'est pas valide, la sortie `OUT` du bloc fonction `SAH` étant de type `ANY`.
L'instruction
```
Cal My_SAH (OUT=>My_Var)
```
est en revanche **valide**.

- Des conditions particulières s'appliquent lors de l'utilisation de variables VAR_IN_OUT *(voir page 498)*.
- L'utilisation des blocs fonction comprend deux parties :
  - la déclaration *(voir page 492)*
  - l'appel du bloc fonction
- Il existe quatre façons d'appeler un bloc fonction :
  - forme formelle de CAL avec liste des paramètres d'entrée *(voir page 492)* (appel avec les noms des paramètres formels)
    Des variables peuvent ainsi être affectées aux sorties via l'opérateur `=>`.
  - forme informelle de CAL avec liste des paramètres d'entrée *(voir page 494)* (appel sans les noms des paramètres formels)
  - CAL et chargement/sauvegarde *(voir page 495)* des paramètres d'entrée
  - usage des opérateurs d'entrée *(voir page 495)*
- Les instances de bloc fonction/DFB peuvent être appelées plusieurs fois, à l'exception des instances d'EFB de communication qui ne peuvent être appelées qu'une seule fois (voir Multiple Call of a Function Block Instance *(voir page 497)*).

Déclaration

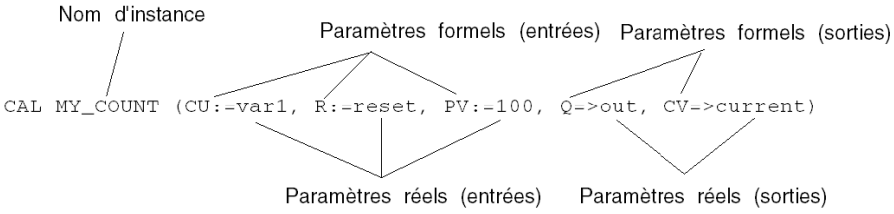
Avant d'être appelé, un bloc fonction doit être déclaré dans l'éditeur de variables.

Forme formelle de CAL avec liste des paramètres d'entrée

Avec cette forme d'appel (avec les noms des paramètres formels), les blocs fonction sont appelés via une instruction qui se compose de l'instruction `CAL`, suivie du nom d'instance du bloc fonction et d'une liste entre parenthèses des affectations de paramètres réels aux paramètres formels. L'affectation des paramètres formels des entrées s'effectue via l'affectation `:=` et l'affectation des paramètres formels des sorties via l'affectation `=>`. L'ordre d'énumération des paramètres formels d'entrées et de sorties **n'est pas important**. La liste des paramètres réels peut être éditée directement après une virgule.

Il est possible d'utiliser `EN` et `ENO` avec ce type d'appel.

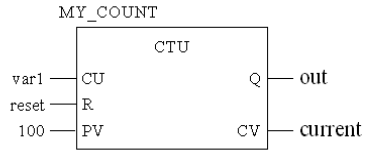
Appel d'un bloc fonction dans la forme formelle de CAL avec liste des paramètres d'entrée :



ou

```
CAL MY_COUNT (CU:=var1, R:=reset, PV:=100, Q=>out,
CV=>current)
```

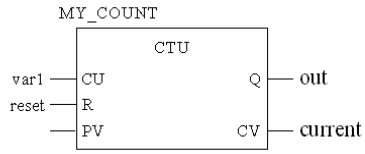
Appel du même bloc fonction dans FBD :



Il n'est pas nécessaire d'affecter une valeur à tous les paramètres formels (voir aussi Parameter *(voir page 489)*).

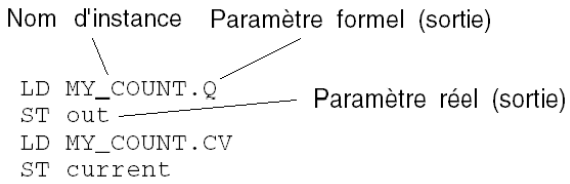
```
CAL MY_COUNT (CU:=var1, R:=reset, Q=>out, CV=>current)
```

Appel du même bloc fonction dans FBD :



Le déplacement de la valeur d'une sortie de bloc fonction peut également avoir lieu via le chargement de la sortie du bloc fonction (nom d'instance du bloc fonction séparé par un point du paramètre formel) suivi d'une sauvegarde.

Chargement et sauvegarde des sorties de bloc fonction :

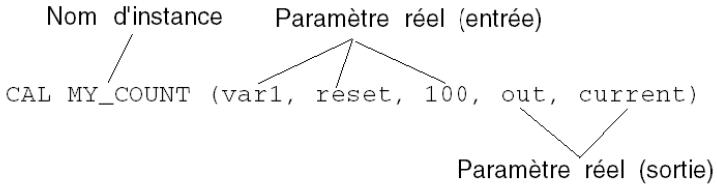


Forme informelle de CAL avec liste des paramètres d'entrée

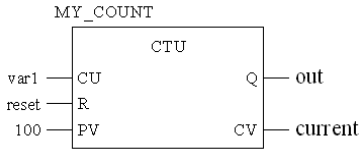
Avec cette forme d'appel (sans les noms des paramètres formels), les blocs fonction sont appelés via une instruction qui se compose de l'instruction `CAL`, suivie du nom d'instance du bloc fonction et d'une liste entre parenthèses des paramètres réels des entrées et sorties. L'ordre d'énumération des paramètres réels dans l'appel d'un bloc fonction **est important**. La liste des paramètres réels ne peut pas être éditée.

`EN` et `ENO` ne peuvent **pas** être utilisés avec ce type d'appel.

Appel d'un bloc fonction dans la forme informelle de CAL avec liste des paramètres d'entrée :



Appel du même bloc fonction dans FBD :



Même lors d'un appel informel, il n'est pas nécessaire d'affecter une valeur à tous les paramètres formels (voir également Parameter *(voir page 489)*).

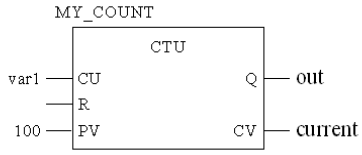
Il s'agit d'une extension de la norme CEI 61131-3, qui doit être activée de manière explicite.

Un champ de paramètre vide permet d'omettre un paramètre.

Appel avec un champ de paramètre vide :

```
CAL MY_COUNT (var1, , 100, out, current)
```

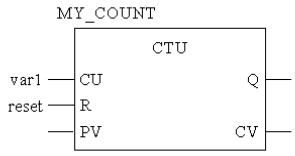
Appel du même bloc fonction dans FBD :



Si les paramètres formels sont omis à la fin, aucun champ de paramètre vide ne doit être utilisé.

```
MY_COUNT (var1, reset)
```

Appel du même bloc fonction dans FBD :



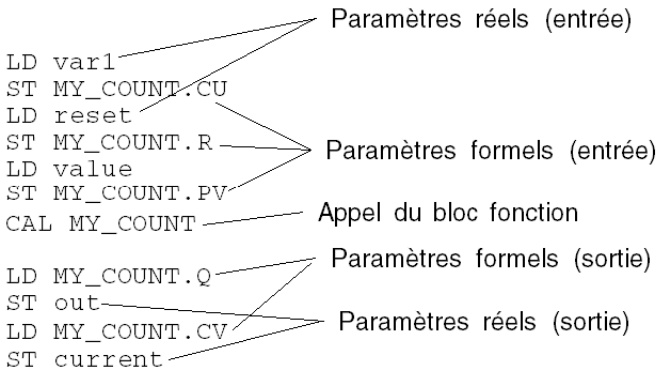
CAL et chargement/sauvegarde des paramètres d'entrée

Vous pouvez appeler les blocs fonction à l'aide d'une liste d'instructions composée du chargement des paramètres réels suivi de leur sauvegarde dans les paramètres formels, suivie de l'instruction `CAL`. L'ordre de chargement et de sauvegarde des paramètres **n'est pas important**.

Seules les instructions de chargement et de sauvegarde pour le bloc fonction en cours de paramétrage doivent figurer entre la première instruction de chargement du paramètre réel et l'appel du bloc fonction. Aucune autre instruction n'est autorisée à cet endroit.

Il n'est pas nécessaire d'affecter une valeur à tous les paramètres formels (voir aussi Parameter *(voir page 489)*).

CAL avec chargement/sauvegarde des paramètres d'entrée :



Usage des opérateurs d'entrée

Vous pouvez appeler les blocs fonction à l'aide d'une liste d'instructions composée du chargement des paramètres réels suivi de leur sauvegarde dans les paramètres formels, suivie d'un opérateur d'entrée. L'ordre de chargement et de sauvegarde des paramètres **n'est pas important**.

Seules les instructions de chargement et de sauvegarde pour le bloc fonction en cours de paramétrage doivent figurer entre la première instruction de chargement du paramètre réel et l'opérateur d'entrée du bloc fonction. Aucune autre instruction n'est autorisée à cet endroit.

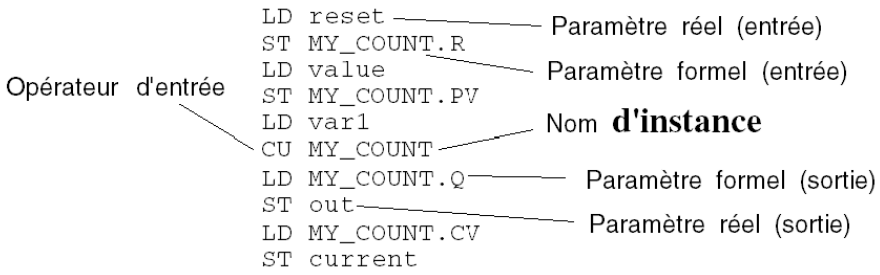
EN et ENO ne peuvent **pas** être utilisés avec ce type d'appel.

Il n'est pas nécessaire d'affecter une valeur à tous les paramètres formels (voir aussi Parameter *(voir page 489)*).

Reportez-vous au tableau pour connaître les opérateurs d'entrée disponibles pour les différents blocs fonction. Aucun autre opérateur d'entrée n'est disponible.

Opérateur d'entrée	Type FB
S1, R	SR
S, R1	RS
CLK	R_TRIG
CLK	F_TRIG
CU, R, PV	CTU_INT, CTU_DINT, CTU_UINT, CTU_UDINT
CD, LD, PV	CTD_INT, CTD_DINT, CTD_UINT, CTD_UDINT
CU, CD, R, LD, PV	CTUD_INT, CTUD_DINT, CTUD_UINT, CTUD_UDINT
IN, PT	TP
IN, PT	TON
IN, PT	TOF

Usage des opérateurs d'entrée :



Appel d'un bloc fonction sans entrées

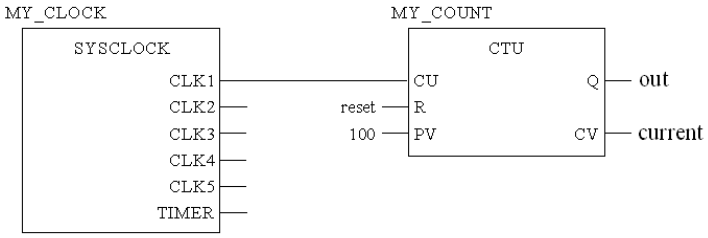
Même si le bloc fonction ne dispose pas d'entrées ou si les entrées ne sont pas à paramétrer, vous devez appeler le bloc fonction avant que ses sorties puissent être utilisées. Faute de quoi, le système transmettra les valeurs initiales des sorties, c'est-à-dire « 0 ».

Ex. :

Appel de blocs fonction dans IL :

```
CAL MY_CLOCK ()CAL MY_COUNT (CU:=MY_CLOCK.CLK1, R:=reset,
PV:=100) LD MY_COUNT.Q ST out LD MY_COUNT.CV ST current
```

Appel du même bloc fonction dans FBD :



Appel multiple d'une instance de bloc fonction

Les instances de bloc fonction/DFB peuvent être appelées plusieurs fois, à l'exception des instances d'EFB de communication qui ne peuvent être appelées qu'une seule fois.

L'appel multiple de la même instance de DFB/bloc fonction est par exemple utile dans les cas suivants :

- si le bloc fonction/DFB ne comporte aucune valeur interne ou si celle-ci n'est plus nécessaire pour un traitement ultérieur.
  Dans ce cas, l'appel multiple de la même instance de DFB/bloc fonction permet d'économiser de l'espace mémoire, car le code du bloc fonction/DFB n'est alors chargé qu'une seule fois.
  Le bloc fonction/DFB est pour ainsi dire traité comme une « fonction ».
- si le bloc fonction/DFB comprend une valeur interne, qui doit influencer différents endroits du programme (la valeur d'un compteur, par exemple, doit être augmentée dans différentes parties du programme).
  Dans ce cas, l'appel multiple de la même instance de bloc fonction/DFB permet d'économiser la mémoire des résultats intermédiaires pour un traitement ultérieur à un autre endroit du programme.

EN et ENO

Pour tous les blocs fonction/DFB, une entrée EN et une sortie ENO peuvent être configurées.

Au cas où la valeur d'EN est égale à « 0 », lorsque le bloc fonction/DFB est appelé, les algorithmes définis par ce dernier ne sont pas exécutés et ENO est mis à « 0 ».

Au cas où la valeur d'EN est égale à « 1 », lorsque le bloc fonction/DFB est appelé, les algorithmes définis par ce dernier sont exécutés. Une fois les algorithmes exécutés, la valeur de la sortie ENO est réglée sur « 1 ». En cas d'erreur lors de l'exécution de ces algorithmes, la sortie ENO est réglée sur « 0 ».

Si aucune valeur n'est attribuée à la broche EN à l'appel du FFB, l'algorithme défini par ce dernier est exécuté (comme lorsque EN a la valeur « 1 »).

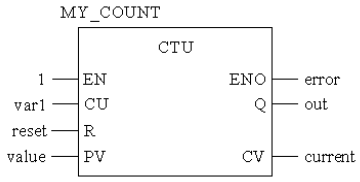
Si ENO est mis à « 0 » (du fait de EN = 0 ou d'une erreur d'exécution), les sorties du bloc fonction/DFB conservent l'état qu'elles avaient au dernier cycle exécuté correctement.

Le comportement aux sorties des blocs fonction/DFB est le même, que les blocs fonction/DFB aient été appelés sans EN/ENO ou avec EN = 1.

Si EN/ENO doivent être utilisés, l'appel du bloc fonction doit être exécuté sous forme d'appel formel. L'affectation d'une variable à ENO doit être effectuée avec l'opérateur =>.

```
CAL MY_COUNT (EN:=1, CU:=var1, R:=reset, PV:=value,
ENO=>error, Q=>out, CV=>current) ;
```

Appel du même bloc fonction dans FBD :



Variable VAR_IN_OUT

Très souvent, on utilise des blocs fonction pour lire une variable au niveau de l'entrée (variables d'entrée), traiter celle-ci, et sortir à nouveau les valeurs modifiées de la même variable (variables de sortie). Ce cas particulier d'une variable d'entrée/de sortie est également appelé variable VAR_IN_OUT.

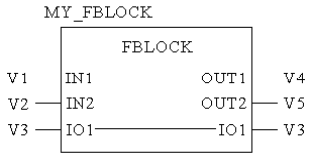
Il convient de noter les particularités suivantes lors de l'utilisation de blocs fonction/DFB avec des variables VAR_IN_OUT :

- une variable doit être affectée à toutes les entrées VAR_IN_OUT.
- il est interdit d'affecter des valeurs littérales ou des constantes aux entrées VAR_IN_OUT.
- **aucune** valeur ne doit être affectée aux sorties VAR_IN_OUT.
- les variables VAR_IN_OUT ne peuvent **pas** être utilisées en dehors de l'appel du bloc fonction.

Appel d'un bloc fonction avec une variable VAR_IN_OUT dans IL :

```
CAL MY_FBLOCK(IN1:=V1, IN2:=V2, IO1:=V3, OUT1=>V4, OUT2=>V5)
```

Appel du même bloc fonction dans FBD :



Les variables VAR_IN_OUT ne peuvent **pas** être utilisées en dehors de l'appel du bloc fonction.

Les appels de blocs fonction suivants sont par conséquent **non valides** :

Appel **non valide**, exemple 1 :

`LD V1`	Chargement de la variable `V1` dans l'accumulateur
`CAL InOutFB`	Appel d'un bloc fonction avec un paramètre `VAR_IN_OUT`. L'accumulateur est alors chargé avec référence à un paramètre `VAR_IN_OUT`.
`AND V2`	Liaison ET du contenu de l'accumulateur avec les variables `V2`. **Erreur :** l'opération ne peut pas être exécutée car il n'est pas possible d'accéder au paramètre `VAR_IN_OUT` (contenu de l'accumulateur) en dehors de l'appel d'un bloc fonction.

Appel **non valide**, exemple 2 :

`LD V1`	Chargement de la variable `V1` dans l'accumulateur
`AND InOutFB.inout`	Liaison ET du contenu de l'accumulateur avec référence à un paramètre `VAR_IN_OUT`. **Erreur :** l'opération ne peut pas être exécutée car il n'est pas possible d'accéder au paramètre `VAR_IN_OUT` en dehors de l'appel d'un bloc fonction.

Les appels de blocs fonction suivants sont en revanche **valides** :

Appel **valide**, exemple 1 :

`CAL InOutFB (IN1:=V1,inout:=V2`	Appel d'un bloc fonction avec un paramètre `VAR_IN_OUT` et affectation des paramètres réels au sein de l'appel de bloc fonction.

Appel **valide**, exemple 2 :

`LD V1`	Chargement de la variable `V1` dans l'accumulateur
`ST InOutFB.IN1`	Affectation du contenu de l'accumulateur au paramètre `IN1` du bloc fonction `IN1`.
`CAL InOutFB(inout:=V2)`	Appel du bloc fonction avec affectation du paramètre réel (`V2`) au paramètre `VAR_IN_OUT`.

Procédures d'appel

Procédure

Les procédures sont disponibles sous forme de bibliothèques. La logique des procédures est établie en langage de programmation C et ne peut pas être modifiée dans l'éditeur IL.

Comme les fonctions, les procédures n'ont pas d'états internes. Lorsque les valeurs d'entrée sont identiques, la valeur de sortie est la même à chaque exécution de la procédure. Par exemple, l'addition de deux valeurs donne toujours le même résultat.

Contrairement aux fonctions, les procédures ne livrent aucune valeur de renvoi et prennent en charge les variables `VAR_IN_OUT`.

Les procédures sont un complément de la norme CEI 61131-3 et doivent être activées de manière explicite.

Paramètres

Pour importer des valeurs dans une procédure ou exporter des valeurs d'une procédure, on a besoin d'« entrées » et de « sorties ». Ces entrées et ces sorties sont appelées « paramètres formels ».

Les états actuels du processus sont transmis aux paramètres formels. Ce sont les paramètres réels.

On peut utiliser comme paramètre réel des entrées de procédure :
- variable,
- adresse,
- valeur littérale.

On peut utiliser comme paramètre réel des sorties de procédure :
- variable,
- adresse.

Le type de données du paramètre réel doit correspondre au type de données du paramètre formel. La seule exception concerne les paramètres formels génériques dont le type de données est déterminé par le paramètre réel.

De plus, pour les paramètres formels génériques du type de données `ANY_BIT`, des paramètres réels du type de données `INT` ou `DINT` (pas `UINT` ni `UDINT`) peuvent être utilisés.

Il s'agit d'une extension de la norme CEI 61131-3, qui doit être activée de manière explicite.

Exemple :

Autorisé :

```
AND (AnyBitParam := IntVar1, AnyBitParam2 := IntVar2)
```

Non autorisé :

```
AND_WORD (WordParam1 := IntVar1, WordParam2 := IntVar2)
```

(Dans ce cas `AND_INT` doit être utilisé.)

```
AND_ARRAY_WORD (ArrayInt, ...)
```

(Dans ce cas, un changement de type explicite doit être effectué via `INT_ARR_TO_WORD_ARR (...)`.

Pour l'appel formel, il n'est en principe pas nécessaire d'affecter une valeur à tous les paramètres formels. Pour connaître les types de paramètre formel pour lesquels cela est cependant impératif, veuillez vous reporter au tableau.

Type de paramètre	EDT	STRING	ARRAY	ANY_ARRAY	IODDT	STRUCT	FB	ANY
Entrée	-	-	+	+	+	+	+	+
VAR_IN_OUT	+	+	+	+	+	+	/	+
Sortie	-	-	-	-	-	-	/	+
+ paramètre réel impératif								
- paramètre réel non impératif								
/ non applicable								

Si aucune valeur n'est affectée à un paramètre formel, la valeur initiale est utilisée pendant l'exécution du bloc fonction. Si aucune valeur initiale n'est définie, la valeur par défaut (0) est utilisée.

Remarques sur la programmation

Veuillez tenir compte des remarques qui suivent sur la programmation :

- Les procédures ne sont exécutées que lorsque l'entrée `EN` = 1 ou lorsque l'entrée `EN` est désactivée (voir aussi EN and ENO *(voir page 504)*).
- Des conditions particulières s'appliquent lors de l'utilisation de variables VAR_IN_OUT *(voir page 505)*.
- Il existe deux façons d'appeler une procédure :
  - appel formel (appel d'une fonction avec les noms des paramètres formels) Des variables peuvent alors être affectées aux sorties via l'opérateur `=>` (appel d'un bloc fonction sous forme abrégée).
  - appel informel (appel d'une fonction sans les noms des paramètres formels)

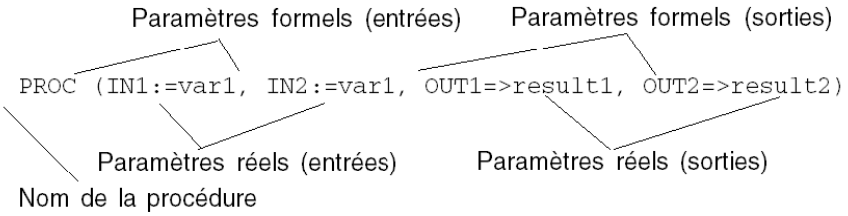
Appel formel

Avec cette forme d'appel (avec les noms des paramètres formels), les procédures sont appelées via une suite d'instructions composée d'une instruction optionnelle `CAL` suivie du nom de la procédure et d'une liste entre parenthèses des affectations de paramètres réels aux paramètres formels. L'affectation des paramètres formels des entrées s'effectue via l'affectation `:=` et l'affectation des paramètres formels des sorties via l'affectation `=>`. L'ordre d'énumération des paramètres formels d'entrées et de sorties **n'est pas important**.

La liste des paramètres réels peut être éditée directement après une virgule.

Il est possible d'utiliser `EN` et `ENO` avec ce type d'appel.

Appel d'une procédure avec les noms des paramètres formels :



ou

```
CAL PROC (IN1:=var1, IN2:=var1, OUT1=>result1,OUT2=>result2)
```

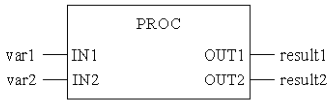
ou

```
PROC (IN1:=var1, IN2:=var1, OUT1=>result1, OUT2=>result2)
```

ou

```
CAL PROC (IN1:=var1, IN2:=var1, OUT1=>result1, OUT2=>result2)
```

Appel de la même procédure dans FBD :



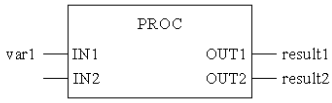
Lors d'un appel formel, il n'est pas nécessaire d'affecter une valeur à tous les paramètres formels (voir également Parameter *(voir page 500)*).

```
PROC (IN1:=var1, OUT1=>result1, OUT2=>result2)
```

ou

```
CAL PROC (IN1:=var1, OUT1=>result1, OUT2=>result2)
```

Appel de la même procédure dans FBD :

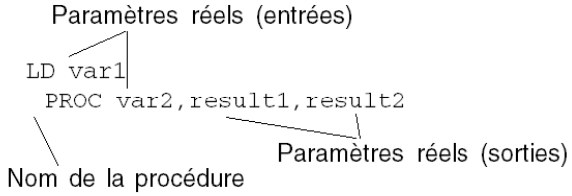


Appel informel sans instruction CAL

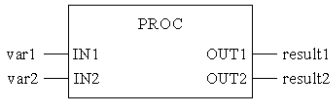
Avec cette forme d'appel (sans les noms des paramètres formels), les procédures sont appelées via une suite d'instructions qui comprend le chargement du premier paramètre réel dans l'accumulateur suivi du nom de la procédure, lui-même suivi d'une liste optionnelle des paramètres réels des entrées et sorties. L'ordre d'énumération des paramètres réels est **important**. La liste des paramètres réels ne peut pas être éditée.

EN et ENO ne peuvent **pas** être utilisés avec ce type d'appel.

Appel d'une procédure avec les noms des paramètres formels :



Appel de la même procédure dans FBD :



NOTE : veuillez noter que pour l'appel informel, la liste des paramètres réels ne doit **pas** être indiquée entre parenthèses. La norme CEI 61133-3 exige dans ce cas d'enlever les parenthèses, afin d'indiquer que le premier paramètre réel ne fait pas partie de la liste.

Appel informel **non valide** d'une procédure :

```
LD A
LIMIT (B,C)
```

Si la valeur à traiter (premier paramètre réel) se trouve déjà dans l'accumulateur, l'instruction de chargement n'est plus nécessaire.

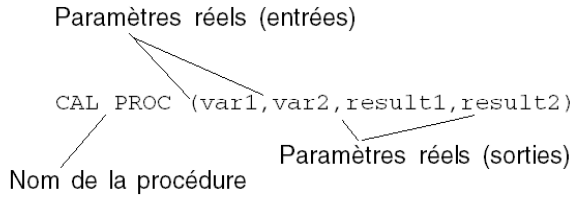
```
EXAMP1 var2,result1,result2
```

Appel informel avec instruction CAL

Avec cette forme d'appel, les procédures sont appelées par une suite d'instructions composée de l'instruction CAL suivie du nom de la procédure, suivi lui-même de la liste entre parenthèses des paramètres réels des entrées et sorties. L'ordre d'énumération des paramètres réels est **important**. La liste des paramètres réels ne peut pas être éditée.

EN et ENO ne peuvent **pas** être utilisés avec ce type d'appel.

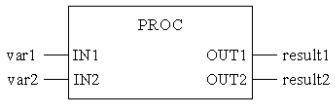
Appel d'une procédure avec les noms de paramètres formels et l'instruction CAL :



ou

```
CAL PROC (var1, var2, result1, result2)
```

Appel de la même procédure dans FBD :



NOTE : contrairement à l'appel informel sans instruction CAL, dans le cadre de l'appel informel avec instruction CAL, la valeur à traiter (le premier paramètre réel) n'est pas chargée explicitement dans l'accumulateur, mais fait partie de la liste des paramètres réels. Par conséquent, lors d'appels informels à l'aide d'une instruction CAL, la liste des paramètres réels doit être indiquée entre parenthèses.

EN et ENO

Pour toutes les procédures, une entrée EN et une sortie ENO peuvent être configurées.

Si la valeur d'EN est égale à « 0 », lorsque la procédure est appelée, les algorithmes définis par cette dernière ne sont pas exécutés et ENO est mis sur « 0 ».

Si la valeur d'EN est égale à « 1 », lorsque la procédure est appelée, les algorithmes définis par la procédure sont exécutés. Une fois les algorithmes exécutés, la valeur de la sortie ENO est réglée sur « 1 ». En cas d'erreur lors de l'exécution de ces algorithmes, la sortie ENO est réglée sur « 0 ».

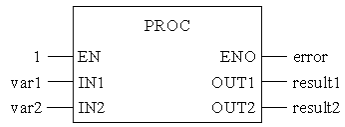
Si aucune valeur n'est attribuée à la broche EN à l'appel du FFB, l'algorithme défini par ce dernier est exécuté (comme lorsque EN a la valeur « 1 »).

Si ENO est mis sur « 0 » (en raison de EN=0 ou d'une erreur d'exécution), les sorties de la procédure sont mises sur « 0 ».

Si EN/ENO doivent être utilisés, l'appel de la procédure doit être exécuté comme un appel formel. L'affectation d'une variable à ENO doit être effectuée avec l'opérateur =>.

```
PROC (EN:=1, IN1:=var1, IN2:=var2, ENO=>error,
OUT1=>result1, OUT2=>result2) ;
```

Appel de la même procédure dans FBD :



Variable VAR_IN_OUT

Très souvent, on utilise des procédures pour lire une variable au niveau de l'entrée (variables d'entrée), traiter celle-ci, et sortir à nouveau les valeurs modifiées de la même variable (variables de sortie). Ce cas particulier d'une variable d'entrée/de sortie est également appelé variable `VAR_IN_OUT`.

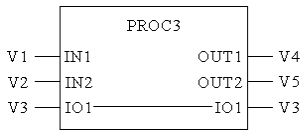
Il convient de noter les particularités suivantes dans le cas de l'utilisation de procédures avec des variables `VAR_IN_OUT` :

- une variable doit être affectée à toutes les entrées `VAR_IN_OUT`.
- il est interdit d'affecter des valeurs littérales ou des constantes aux entrées `VAR_IN_OUT`.
- **aucune** valeur ne doit être affectée aux sorties `VAR_IN_OUT`.
- les variables `VAR_IN_OUT` ne peuvent **pas** être utilisées en dehors de l'appel de procédure.

Appel d'une procédure avec une variable `VAR_IN_OUT` dans IL :

```
PROC3 (IN1:=V1, IN2:=V2, IO1:=V3, OUT1=>V4, OUT2=>V5) ;
```

Appel de la même procédure dans FBD :



Les variables `VAR_IN_OUT` ne peuvent **pas** être utilisées en dehors de l'appel de procédure.

Les appels de procédure suivants sont par conséquent **invalides** :

Appel **non valide**, exemple 1 :

`LD V1`	Chargement de la variable `V1` dans l'accumulateur
`CAL InOutProc`	Appel d'une procédure avec un paramètre `VAR_IN_OUT`. L'accumulateur est alors chargé avec référence à un paramètre `VAR_IN_OUT`.
`AND V2`	Liaison ET du contenu de l'accumulateur avec la variable `V2`. **Erreur :** l'opération ne peut pas être exécutée car il n'est pas possible d'accéder au paramètre `VAR_IN_OUT` (contenu de l'accumulateur) en dehors de l'appel de procédure.

Appel **non valide**, exemple 2 :

`LD V1`	Chargement de la variable `V1` dans l'accumulateur
`AND InOutProc.inout`	Liaison ET du contenu de l'accumulateur avec référence à un paramètre `VAR_IN_OUT`. **Erreur :** l'opération ne peut pas être exécutée car il n'est pas possible d'accéder au paramètre `VAR_IN_OUT` en dehors de l'appel de procédure.

Appel **non valide**, exemple 3 :

`LD V1`	Chargement de la variable `V1` dans l'accumulateur
`InOutFB V2`	Appel de la procédure avec affectation du paramètre réel (`V2`) au paramètre `VAR_IN_OUT`. **Erreur :** l'opération ne peut pas être exécutée car, pour ce type d'appel de procédure, le paramètre `VAR_IN_OUT` continuerait d'être utilisable dans l'accumulateur.

Les appels de procédure suivants sont en revanche **valides** :

Appel **valide**, exemple 1 :

`CAL InOutProc (IN1:=V1,inout:=V2)`	Appel d'une procédure avec un paramètre `VAR_IN_OUT` et affectation formelle des paramètres réels au sein de l'appel de procédure.

Appel **valide**, exemple 2 :

`InOutProc (IN1:=V1,inout:=V2)`	Appel d'une procédure avec un paramètre `VAR_IN_OUT` et affectation formelle des paramètres réels au sein de l'appel de procédure.

Appel **valide**, exemple 3 :

`CAL InOutProc (V1,V2)`	Appel d'une procédure avec un paramètre `VAR_IN_OUT` et affectation informelle des paramètres réels au sein de l'appel de procédure.