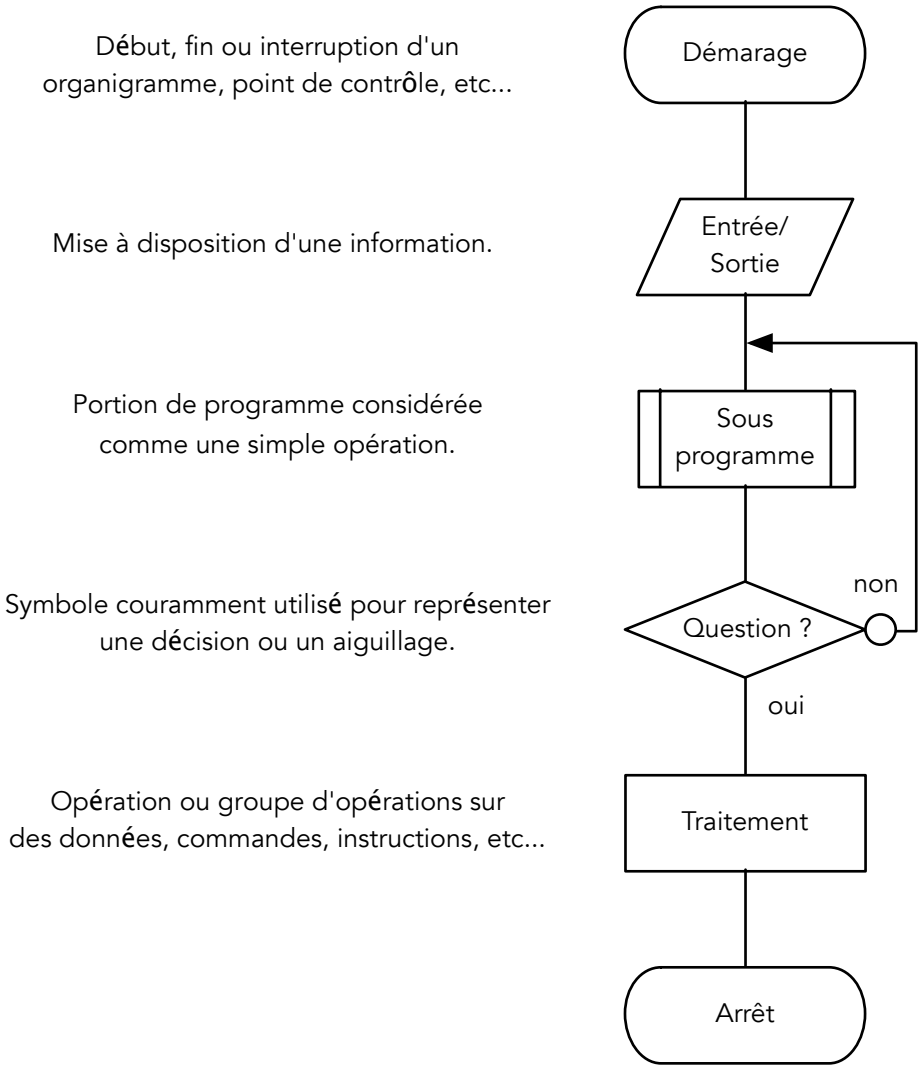
Langage ST et organigrammes

1. Organigrammes

1.1. Généralités

Un organigramme est une représentation schématique des liens et des relations fonctionnels, organisationnels et hiérarchiques qui existent entre les éléments et les individus d'une organisation formelle (association, entreprise, réseau, etc.), d'un programme, etc. Il met en évidence sa structure organisationnelle. Une norme ISO a été développée, elle porte le numéro ISO 5807. Elle décrit en détail les différents symboles à utiliser pour représenter un programme informatique de manière normalisée. (WIKIPÈDIA)

1.2. Eléments d'un organigramme



Remarques :

- Un organigramme se lit de haut en bas en suivant le trait qui lie les figures. Il faut matérialiser par une flèche une lecture dans un sens différent.
- Plusieurs actions peuvent s'écrire dans un seul rectangle de traitement. On peut également les inscrire dans des rectangles successifs pour bien marquer l'aspect séquentiel par exemple.
- Les indications oui/non de sortie d'aiguillage peuvent se mettre de diverses façons (oui à droite, à gauche…). Pour « gagner du temps » on peut trouver comme sur la figure ci-dessus, un rond qui est la marque du non. Ce rond peut se trouver dessiner sous ou à gauche du losange. Bien entendu, l'autre branche est la réponse oui.

2. Langage ST

2.1. Présentation

La norme IEC 61131-3 définit un langage de programmation textuel structuré, c'est le « Structured Text » (ST). C'est un langage performant et adapté aux systèmes d'automatisation. C'est le langage textuel qui nous permettra de programmer les organigrammes.

Il est parfaitement adapté pour la définition des blocs fonctionnels complexes ou non qui seront ensuite employés dans les autres langages.

La structure générale d'une instruction est la suivante :

```
<Variable> := <opérande> <opérateur> <opérande> ... ;
```

Chaque instruction se termine par le caractère « ; ».

2.2. Commentaires

Le commentaire facilite l'interprétation d'une phrase à laquelle il est affecté. Le commentaire peut être intégré n'importe où dans le code et le nombre de commentaires n'est pas limité. Un commentaire est encadré de part et d'autre par les caractères (* et *).

```
(* Ceci est un commentaire ;-) *)
```

2.3. Types de variables classiques

Nom	Notation	Taille (bit)	Etendue
Booléen	BOOL	1	[0(false),1(true)]
Entier court	SINT	8	[-128 ; 127]
Entier	INT	16	[-32 768 ; 32 767]
Entier double	DINT	32	[-2 147 483 648 ; 2 147 483 647]
Entier court non signé	USINT	8	[0 ; 255]
Entier non signé	UINT	16	[0 ; 65 535]
Entier double non signé	UDINT	32	[0 ; 4 294 967 295]
Nombre flottant	REAL	32	$[-2^{128} ; 2^{128}]$
Nombre flottant long	LREAL	64	$[-2^{1024} ; 2^{1024}]$
Entier sur 8 bits (en hexa)	BYTE	8	[16#00 ; 16#FF]
Entier sur 16 bits (en hexa)	WORD	16	[16#0000 ; 16#FFFF]
Entier sur 32 bits (en hexa)	DWORD	32	[16#0000 ; 16#FFFFFFFF]
Temps	TIME(*)		T#0,00s à T#21 474 836,47s
TIME_OF_DAY, DATE_AND_TIME, DATE,			DATE_AND_TIME#2001-04-02- 10:15:29.00 DATE#2001-04-02
Chaine de caractères	STRING	8 bits / ASCII	1 à 255 caractères ASCII

Remarque :

Il existe des fonctions comme INT_TO_REAL qui permettent la conversion des variables.

2.4. Opérateurs

Opérateurs	Description	Opérateur	Description
Fonctions logiques			
NOT	NON	OR	OU
AND	ET	XOR	OU Exclusif
Opérateurs arithmétiques			
+	Addition	-	Soustraction
mod	Reste de la division entière	/	Division
		*	Multiplication
Opérateurs de comparaison			
=	Égal à	<>	Différent de
>	Supérieur à	>=	Supérieur ou égal à
<	Inférieur à	<=	Inférieur ou égal à

2.5. Tableaux

Il est possible de définir des tableaux de tous types de variables classiques, ainsi que leurs valeurs initiales.

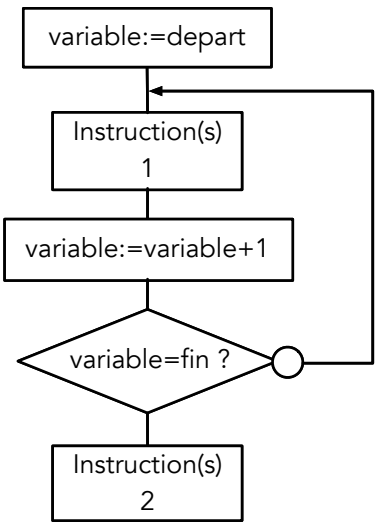
Exemples :

```
VAR
        X ARRAY [1..5] OF INT := 1,2,3,4,5; (* Tableau de 5 entiers *)
        Y ARRAY [1..2,3..4] OF INT := 1,3(7); (* pour 1,7,7,7 *)
        Z ARRAY [1..2,2..3,3..4] OF INT := 2(0),4(4),2,3; (* On arrête là *)
END_VAR
(* Exemple de code *)
X[1]:=10;
Y[1,2]:= X[2];
Z[2,3,4]:=5;
```

2.6. Structures de contrôle

2.6.1. FOR

```
for variable:=depart to fin do
        {Instruction(s) 1}
end_for;
        {Instruction(s) 2}
```

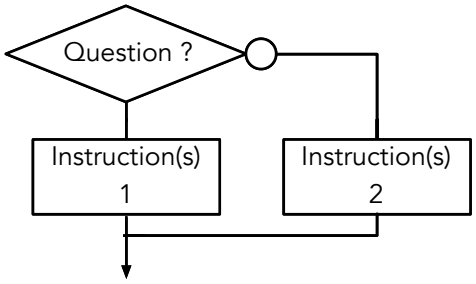


Exemple :

```
k:=0;
for i:=0 to 10 do
        k:=1-k;
end_for;
k:=0;
```

2.6.2. IF

```
if Question then
        {Instruction(s) 1}
else
        {Instruction(s) 2}
end_if;
```

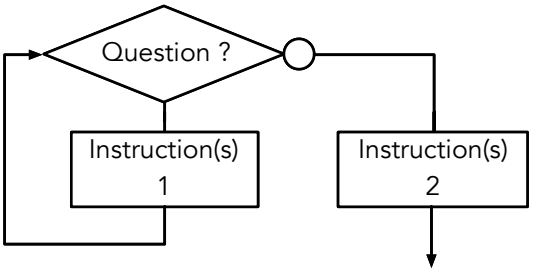


Exemple :

```
if i=3 then
        j:=true;
else
        j:=false;
end_if;
```

2.6.3. WHILE

```
while Question do
        {Instruction(s) 1}
end_while;
{Instruction(s) 2}
```

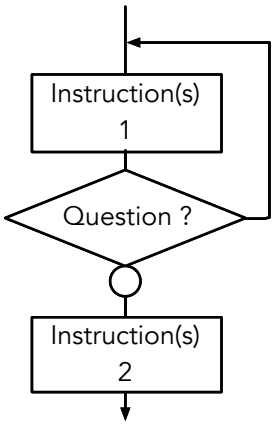


Exemple :

```
k:=false;
i:=0;
while i<10 do
        i:=i+1;
end_while;
k:=true;
```

2.6.4. REPEAT

```
repeat
        {Instruction(s) 1}
until Question
end_repeat
{Instruction(s) 2}
```

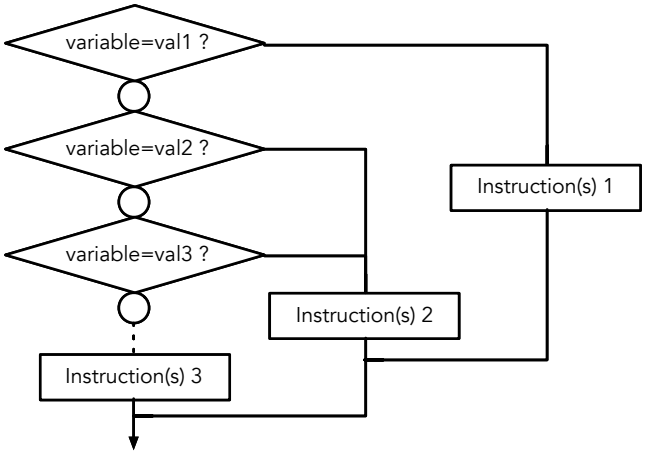


Exemple :

```
k:=false;
i:=0;
repeat
        i:=i+1;
        until i>10
end_repeat;
k:=true;
```

2.6.5. CASE

```
case variable of
        val1:{Instruction(s) 1}
        val2,val3:{Instruction(s) 2}
        val4...
else {Instruction(s) 3}
end_case;
```



Exemple :

```
case i of
        1 : k:=1;
        2,3 : k:=3;
        else k:=10;
end_case;
```

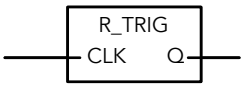
2.7. Fonctions

Comme tout langage structuré, le langage ST permet de créer des fonctions personnalisées. Dans un premier temps, cependant, il est bon de vérifier que la fonction désirée n'est déjà pas disponible dans une des bibliothèques de l'automate. La structure de la déclaration d'une fonction est la suivante :

```
function_block nomdelafonction
        var_input
                {Déclaration de variable(s)}
        end_var
        var_output
                {Déclaration de variable(s)}
        end_var
        var
                {Déclaration de variable(s)}
        end_var
        var_in_out
                {Déclaration de variable(s)}
        end_var
        {Instruction(s)}
end_function_block
```

Exemple :

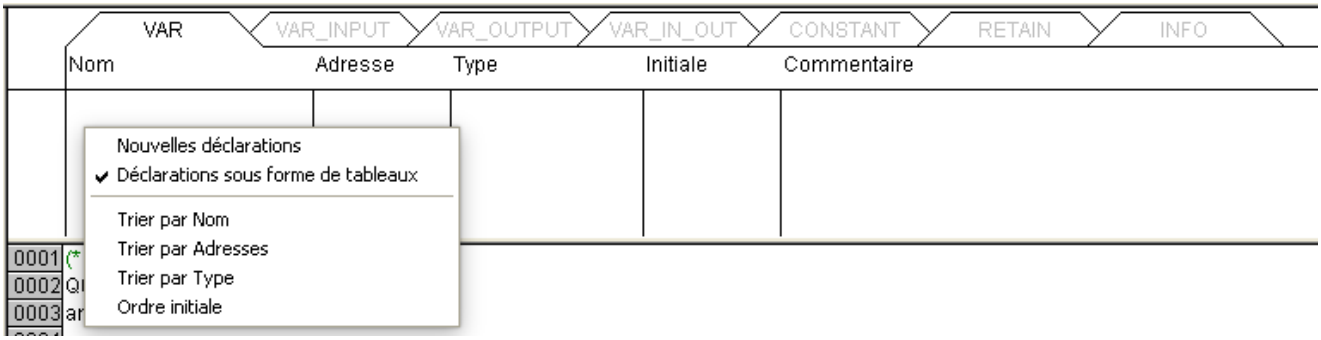
```
function_block r_trig
        var_input
                clk : bool;
        end_var
        var_output
                q : bool;
        end_var
        var
                m : bool := false;
        end_var
        q := clk and not m;
        m := clk;
end_function_block
```



3. CODESYS

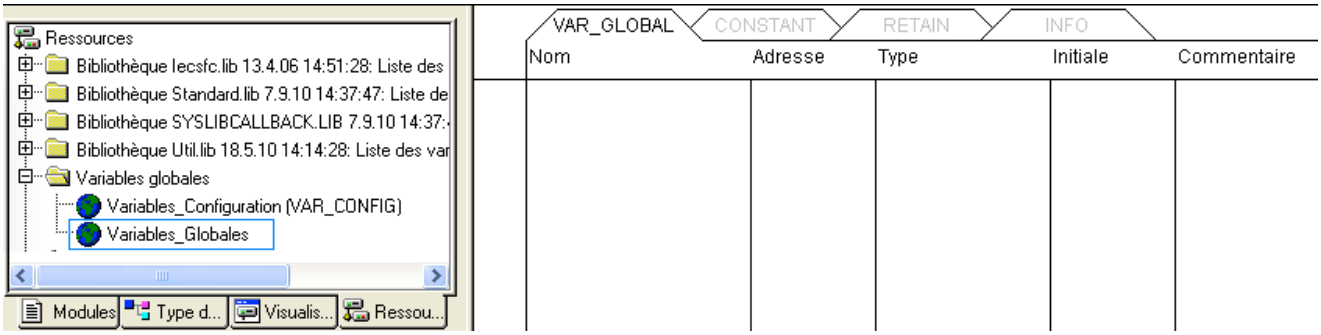
3.1. Visibilité des variables

Avant de commencer, vérifiez que vos variables s'affichent sous forme de tableau, c'est plus lisible et convivial.



Les variables que vous définissez dans ce tableau ne sont visibles que dans le module concerné.

Pour des variables visibles dans tous les modules, il faut les déclarer dans l'onglet Ressources, répertoire Variables Globales.



Attention : Toujours définir le type des variables en MAJUSCULE.

3.2. Les blocs fonctionnels

Avec Codesys on peut créer facilement des modules fonctionnels, munis d'entrée(s), de sortie(s) et de variable(s) interne(s). Il suffit de rajouter un module ST, de définir les variables d'entrée, les variables de sortie ainsi que les variables internes. Ce bloc pourra alors être utilisé dans le programme principale, ou tout autre bloc.

3.3. Astuce : Créer un bit de désactivation

Pour créer un bit de désactivation d'un bloc fonctionnel, utilisable dans le programme principal, il suffit de :

- Définir une variable d'entrée de type BOOL, que l'on peut nommer STOP ;
- Ecrire à la première ligne du programme (LD) le code suivant :
  `IF(STOP) THEN RETURN; END_IF`
- Il apparait alors une entrée STOP qui désactive la fonction (ici AUTO).