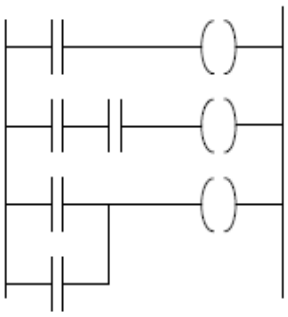
SEMAINE 4

LA PROGRAMMATION DES AUTOMATES

FICHE 19 : LES LANGAGES DE PROGRAMMATION

LE STRUCTURED TEXT (ST)

Ladder



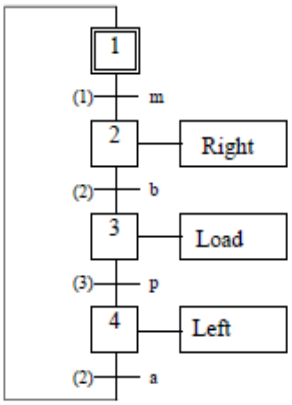
Structured Text (ST)

```
If %I1.0 THEN
  %Q2.1 := TRUE
ELSE
  %Q2.2 := FALSE
END_IF
```

Instruction List (IL)

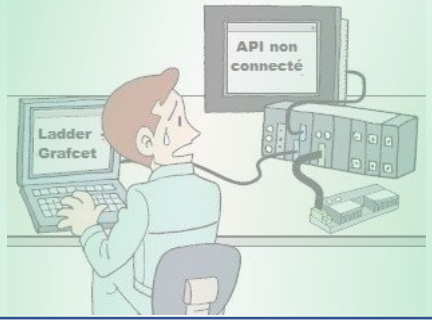
```
LD      %M12
AND     %I1.0
ANDN    %I1.1
OR      %M10
ST      %Q2.0
```

Grafcet



Automation & Sense

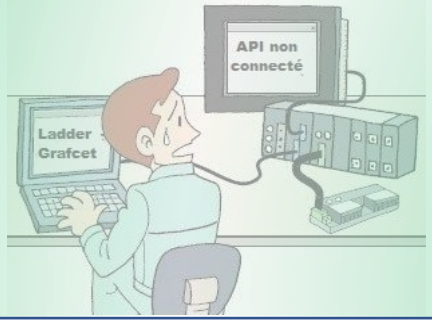
Novembre 2017 | www.automation-sense.com



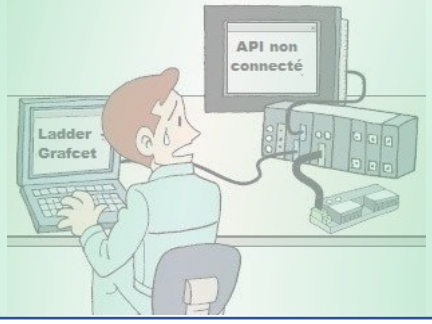
Objectifs :

Après la consultation de cette fiche, vous :

- Pourrez décrypter un programme conçu en langage ST
- Pourrez concevoir un programme en langage ST



SOMMAIRE



I) Généralités

La commission d'électrotechnique internationale (IEC) a défini cinq langages standards pour la programmation d'automates à savoir le Ladder, le Structured Text (ST), les blocs de fonctions FBD, l'instruction List(IL) et le Sequential Function Chart (SFC).

- Le Structured Text (ST) : c'est un langage haut niveau assez similaire aux langages comme le C ou le Pascal syntaxiquement. Il permet d'effectuer des fonctions complexes comme des fonctions PID, des fonctions mathématiques complexes etc...

- Le Ladder ou encore langage à contacts est de loin le langage de programmation le plus utilisé du fait de sa simplicité.Il est idéal pour les électriciens qui n'ont pas forcément de compétences en informatique et langage haut niveau.

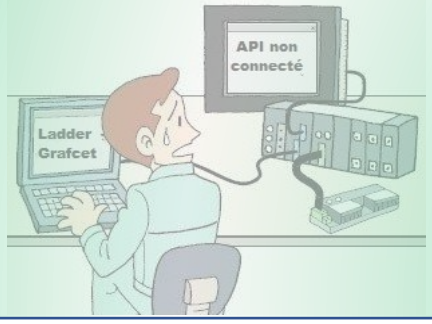
- Le langage IL ou Instruction List : C'est un langage bas niveau très similaire au langage assembleur syntaxiquement.

- Le langage FBD : il est composé de blocs de fonctions servant à implémenter des fonctions logiques. Dans ce langage, on utilise les principaux portes logiques (AND,OR,NOT etc..) qu'on a pu voir dans les fiches précédentes.

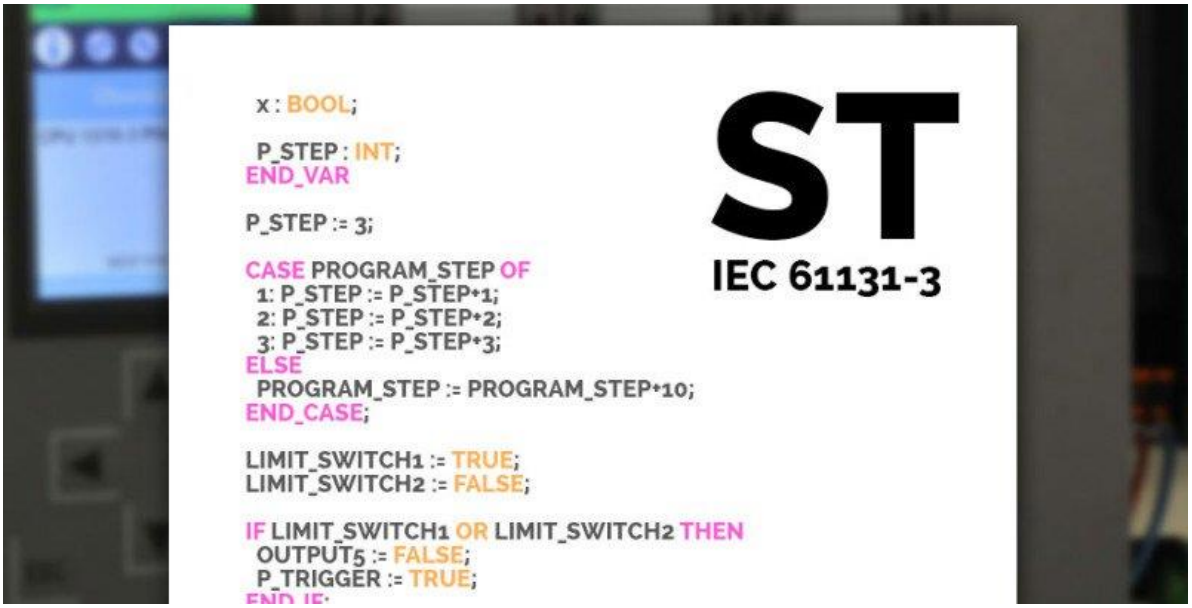
- Le langage SFC (Sequential Function Chart) : c'est un langage graphique qui permet de décrire l'évolution séquentielle d'un système automatisé. Il est assez similaire au grafcet.

Lors de la conception de projet en automatisme, il est assez fréquent d'utiliser plusieurs langages au sein d'un même projet. Par exemple si on travaille avec le langage SFC sur TIA Portal, on pourra écrire les réceptivités d'un grafcet en langage Ladder. En fonction du langage avec lequel on est le plus familier, on peut choisir d'utiliser tel langage à la place d'un autre.

Dans cette fiche, on traitera du langage ST ou Structured Text.



II) C'est quoi le Structured Text ?

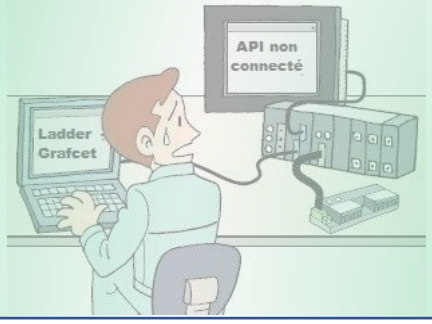


Le texte structuré ou Structured Text ou encore littéral structuré est un langage de programmation haut niveau défini par la **PLCopen** dans la norme **CEI 61131-3**. Ce langage de programmation est basé sur du texte, comparé au Ladder qui est graphique.

Si vous êtes déjà familier avec les langages de programmation comme le C, le Python ou le Java, vous maitriserez facilement le langage Structured Text : leur syntaxe est quasiment la même. Par exemple au niveau du texte structuré, on retrouvera les notions de variables, de conditions, de boucles etc…

Si vous n'avez jamais vu un langage de programmation haut niveau, le texte structuré peut être une excellente introduction à ces types de langages.

Pour commencer, lisez le programme ci-dessous et essayez de le décrypter. Cela vous permettra déjà de voir la syntaxe du langage de manière globale.

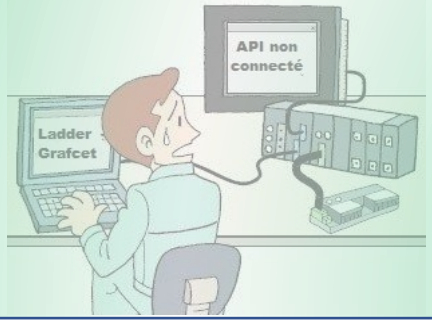


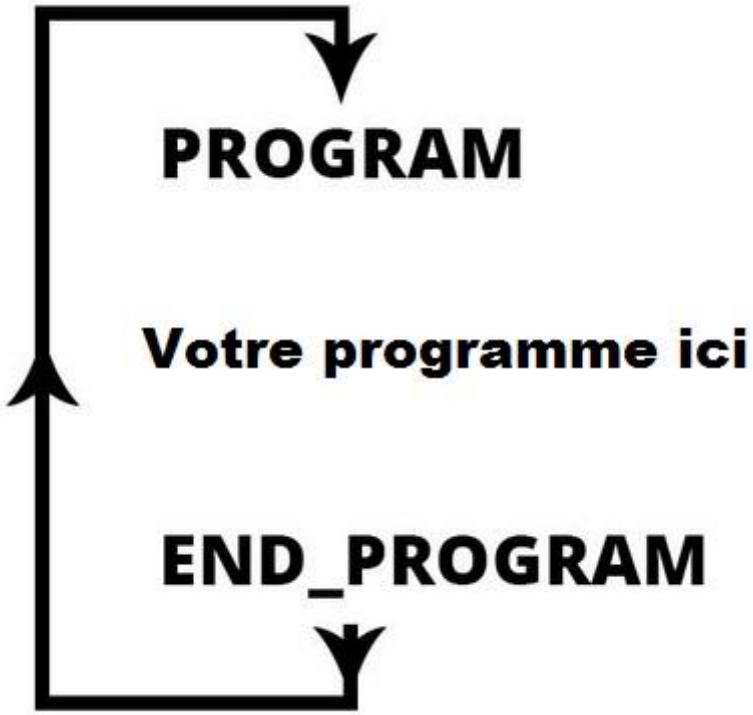

```
PROGRAM stexample
  VAR
    x : BOOL;
  END_VAR
  x := TRUE;
  REPEAT
    x := FALSE;
  UNTIL x := FALSE;
  END_REPEAT;
END_PROGRAM;
```

En effet, la première chose que vous devrez apprendre est la syntaxe du texte structuré. Une fois que vous comprendrez la structure du langage, vous comprendrez comment concevoir un programme.

A partir de l'exemple ci-dessus, vous pouvez voir que le programme commence avec le mot clé **PROGRAM** et se termine avec **END_PROGRAM**. Tout le reste constitue le corps du programme. Ces deux mots clés permettent de délimiter le programme. Pour l'instant, ne soyez pas confus, nous verrons ces mots-clés plus tard.

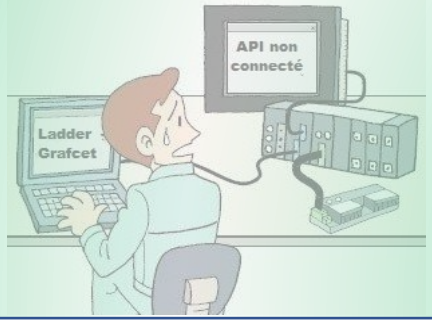
Tout ce qu'on peut retenir pour l'instant c'est qu'un programme conçu en texte structuré commence avec le mot clé **PROGRAM** et que une fois que l'automate aura exécuté tout le programme et arrivera au mot clé **END_PROGRAM**, le cycle de l'automate recommencera, et le programme sera exécuter de nouveau comme symbolisé sur l'image ci-dessous.





Le schéma ci-dessus symbolise la structuration d'un programme conçu en texte structuré. Comme pour les autres langages, l'automate exécutera le programme de manière infinie tant qu'il est en mode **RUN**. Si vous êtes habitué à la programmation des microcontrôleurs, c'est comme une boucle infinie.

NB : Une chose à ajouter ici est que, lorsque vous programmez en texte structuré, vous n'aurez souvent pas besoin d'utiliser les mots clés **PROGRAMM / END_PROGRAM**. Ils seront ajoutés automatiquement par le logiciel de programmation de l'API, vous aurez juste à ajouter votre programme. Aussi, en texte structuré comme pour les autres langages les instructions sont exécutées ligne par ligne.



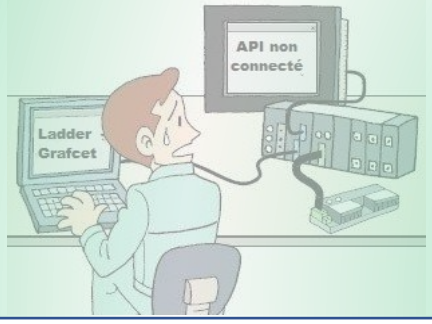
III) La syntaxe du texte structuré

La syntaxe d'un langage de programmation est la définition de la façon dont il est écrit. Comme vous pouvez le voir dans l'exemple précédent, le texte structuré est plein de **deux points**, **points virgules** et **autres symboles**. Tous ces symboles ont une signification et sont utilisés pour représenter quelque chose. Certains d'entre eux sont des opérateurs, d'autres des symboles caractérisant la fin d'une instruction.

Tous les détails de la syntaxe du langage Structured Text seront expliqués tout au long de cette fiche. Il y'a quelques règles générales de syntaxe à connaitre. Vous n'êtes pas dans l'obligation de mémoriser toutes les règles de syntaxe pour le moment, vous allez vous familiarisé avec le Structured Text au fur et à mesure que vous pratiquerez. Ci-dessous, deux règles capitales à respecter :

Règle 1 : Toutes les instructions du langage ST se terminent par un point-virgule
Règle 2 : Le Texte structuré n'est pas sensible à la casse

Ce qui est vraiment important de comprendre ici est que, lorsque vous écrivez un programme automate en texte structuré, votre ordinateur va traduire le programme dans un «langage » que l'automate peut comprendre. En effet, lorsque vous téléchargez le programme dans l'automate, le logiciel de programmation que vous utilisez se chargera avant de compiler votre programme. Cela signifie qu'il traduira votre code en une sorte de code machine qui peut être exécuté par l'automate. Le compilateur utilise la syntaxe du langage de programmation et effectue la traduction au fur et à mesure. Par exemple: A chaque fois que le compilateur voit un point-virgule, il saura que c'est la fin de l'instruction en cours. Le compilateur va donc tout lire jusqu'à ce qu'il atteigne un point-virgule, puis exécutera l'instruction en question.



a) Les commentaires en langage ST

Dans les langages de programmation textuels, vous avez la possibilité d'inclure des commentaires ou annotations qui ne seront pas exécutés par le processeur. Les commentaires sont très utiles au programmeur et en tant que débutant, vous devriez toujours commenter votre code. Ainsi, il vous sera plus facile de le comprendre plus tard.

En texte structuré, vous pouvez ajouter un commentaire à chaque ligne ou ajouter plusieurs lignes de commentaires à la fois.

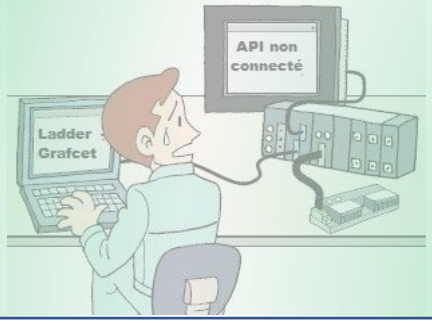
Commentaire sur 1 ligne

```
// comment
```

Commentaire sur plusieurs lignes

```
/* start comment
...
end comment */
```

```
(* start comment
...
end comment *)
```



b) Les instructions en texte structuré

Une instruction est un bout de code qui permet de dire à l'automate ce qu'il doit faire. Dans l'exemple ci-dessous, l'instruction correspond à une déclaration de variable.

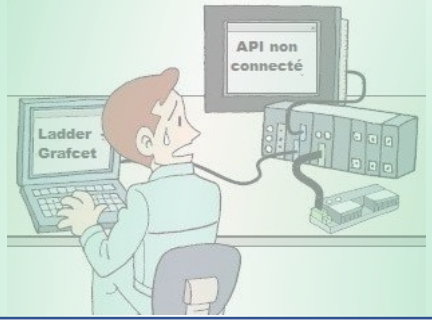
```
X : BOOL;
```

Le compilateur va lire cela comme une instruction, parce que quand il atteint le point-virgule, il sait que c'est la fin de l'instruction. Rappelez-vous des règles énoncées avant : **les instructions sont séparées par des points-virgules**. Dans l'instruction ci-dessus, vous dites à l'API de créer une variable appelée **X** et cette variable est de type **BOOL**.

c) Les variables en Structured Text

Revenons maintenant sur les mots-clés que l'on a mentionné auparavant. Comme vous pouvez le voir dans l'exemple de tout à l'heure, la variable **X** est défini entre deux autres mots clés **VAR** et **END_VAR**. Les mots clés **PROGRAMM / END_PROGRAM** et **VAR / END_VAR** délimitent une certaine zone dans le programme. Le mot clé **PROGRAMM** est le début du corps du programme, et le mot clé **VAR** débute l'endroit où on définit les variables (l'endroit où sont déclarés les variables).

Que ce soit **VAR,PROGRAMM,END_VAR** etc.. , ils sont appelés mots-clés, parce qu'ils sont des mots réservés. Vous ne pouvez pas utiliser ces mots pour autre chose lorsque vous programmez en texte structuré. Par exemple le nom de votre programme ne peut pas être **PROGRAMM** ou même **programm** (Le ST n'est pas sensible à la casse).



Une variable est un endroit où vous pouvez stocker des données. Selon le type de données que vous voulez stocker, il existe plusieurs types de variables disponibles. Par exemple, si vous avez une variable où vous souhaitez stocker un bit avec des valeurs vraies ou fausses, vous pouvez déclarer la variable comme un type booléen. Lorsque vous déclarez une variable dans la section **VAR/END_VAR**, vous commencerez par donner un nom à votre variable.

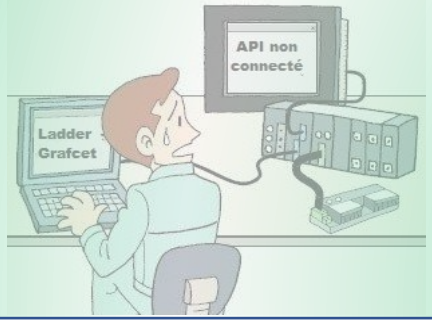
```
X : BOOL;
```

Par exemple, la déclaration ci-dessus va créer une variable appelée **X**, avec un type de données **BOOL**. Soyez conscient que lorsque vous programmez avec certains logiciels comme STEP 7 de Siemens ou RSLogix de Rockwell vous ne pourrez pas utiliser les mots clés **VAR / END_VAR** pour déclarer des variables. Sur Step 7, les variables sont souvent appelés tags ou symboles, si vous programmez en texte structuré, vous déclarez votre variable au niveau de la table des variables.

IV) Les Types de données en texte structuré

Selon la marque de l'automate que vous utilisez, vous aurez différents types de données disponibles. Pour les automates Siemens, vous avez les types de données définis au niveau de STEP 7 et qui sont similaires à ceux du standard de la CEI 61131-3. Mais vous aurez également d'autres types de données uniquement utilisés au niveau des automates SIEMENS comme par exemple le type **S5TIME**.

Tous les types de données standards sont définis par l'organisation PLCOpen et ils font partie des langages de programmation des APIs. Chaque logiciel de programmation d'API peut inclure ses propres types de données en plus des standards. Dans la norme IEC les types de données sont divisés en deux catégories: les types de données élémentaires et les types de données dérivés.

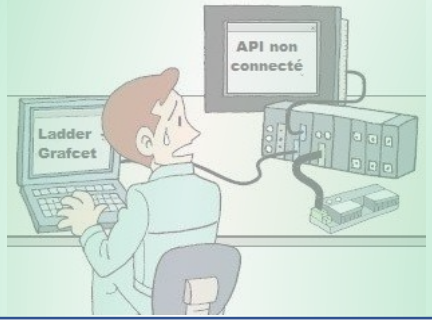


Les types « Integers » ou entiers :

Type	Format	Plage
SINT	Short Integer	-128 … 127
INT	Integer	-32768 … 32767
DINT	Double Integer	-2^31 … 2^31-1
LINT	Long Integer	-2^63 … 2^63-1
USINT	Unsigned Short Integer	0 … 255
UINT	Unsigned Integer	0 … 2^16-1
LDINT	Long Double Integer	0 … 2^32-1
ULINT	Unsigned Long Integer	0 … 2^64-1

Les types float :

Type	Format	Plage
REAL	Real Numbers	±10^±38
LREAL	Long Real Numbers	±10^±308



Les types dérivés

- Structure
- Enumération
- Tableau

Les types de données dérivés sont des types de données personnalisés que l'utilisateur peut créer.

V) Les opérateurs en Structured Text

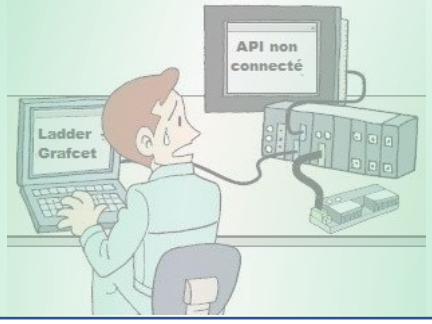
Les opérateurs arithmétiques

- **+** (addition)
- **–** (soustraction)
- ***** (multiplication)
- ****** (exposant)
- **/** (division)
- **MOD** (modulo)

Les opérateurs de comparaison

- **=** (égal)
- **<** (inférieur)
- **<=** (inférieur ou égal)
- **>** (supérieur)
- **>=** (supérieur ou égal)
- **<>** (différent)

```
TEMPERATURE := 93.9;
TEMPERATURE >= 100.0
```



Les opérateurs logiques

- **&** ou **AND**
- **OR**
- **XOR**
- **NOT**

VI) L'instruction d'affectation

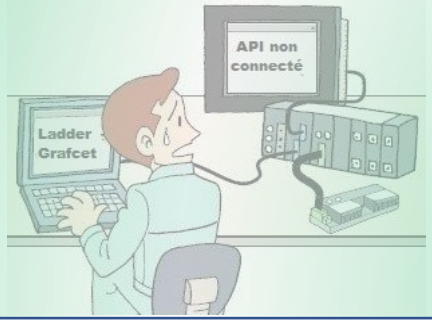
Dans l'image ci-dessous l'instruction permet de mettre la variable B dans A. Le symbole **: =** est appelé opérateur d'affectation.

```
A := B;
```

Une erreur courante consiste à utiliser l'opérateur d'égalité (**=**) au lieu de l'opérateur d'affectation (**: =**). Mais même si elles se ressemblent fortement, il y'a une énorme différence entre les deux. Par exemple si on a **if (A = B)** , cela permet d'évaluer si la valeur de la variable A est égale à B.

VII) Les conditions avec IF/ELSE

```
IF [boolean expression] THEN
  <statement>;
ELSIF [boolean expression] THEN
  <statement>;
ELSE
  <statement>;
END_IF ;
```

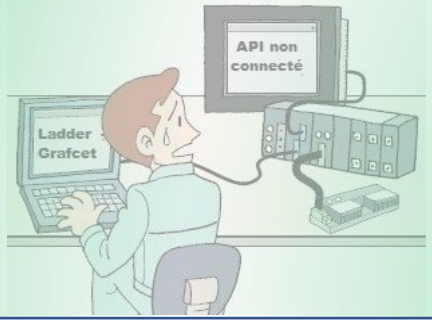


Dans l’image ci-dessous, la ligne 3 ne sera exécuté que si A est égal à 0.

```
A := 0;
IF A = 0 THEN
  B := 0;
END_IF ;
```

Si l'expression booléenne de la ligne 1 est FAUX, les instructions immédiatement en dessous ne seront pas exécutées. Au lieu de cela le compilateur va vérifier l'expression booléenne après le mot clé **ELSIF**.

```
IF [boolean expression] THEN
  <statement>;
ELSIF [boolean expression] THEN
  <statement>;
ELSE
  <statement>;
END_IF ;
```



VIII) Les conditions avec CASE/END_CASE

La deuxième façon de faire des conditions en texte structuré est l'utilisation du mot clé **CASE.** En effet si on a plusieurs conditions à la fois, on utilisera par souci de clarté le mot clé **CASE** au lieu de **IF.**

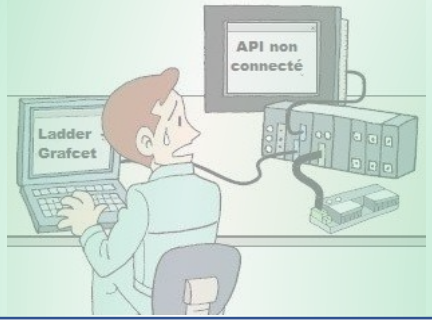
```
CASE [numeric expression] OF
  result1: <statement>;
  resultN: <statemtent>;
ELSE
  <statement>;
END_CASE;
```

```
PROGRAM_STEP := 3;
CASE PROGRAM_STEP OF
  1: PROGRAM_STEP := PROGRAM_STEP+1;
  2: PROGRAM_STEP := PROGRAM_STEP+2;
  3: PROGRAM_STEP := PROGRAM_STEP+3;
ELSE
  PROGRAM_STEP := PROGRAM_STEP+10;
END_CASE;
```

IX) Les itérations en langage ST

La boucle For

```
FOR count := initial_value TO final_value BY increment DO
  <statement>;
END_FOR;
```



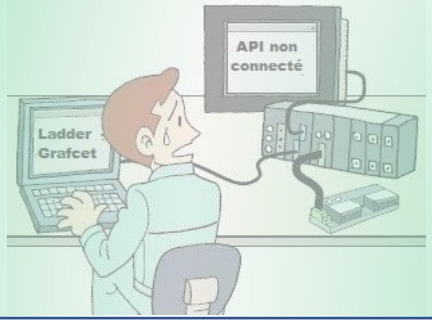
La boucle While

```
WHILE [boolean expression] DO
  <statement>;
END_WHILE;
```

La boucle Repeat

Il fonctionne à l'inverse de la boucle while. Cette boucle cessera de se répéter quand l'expression booléenne est TRUE.

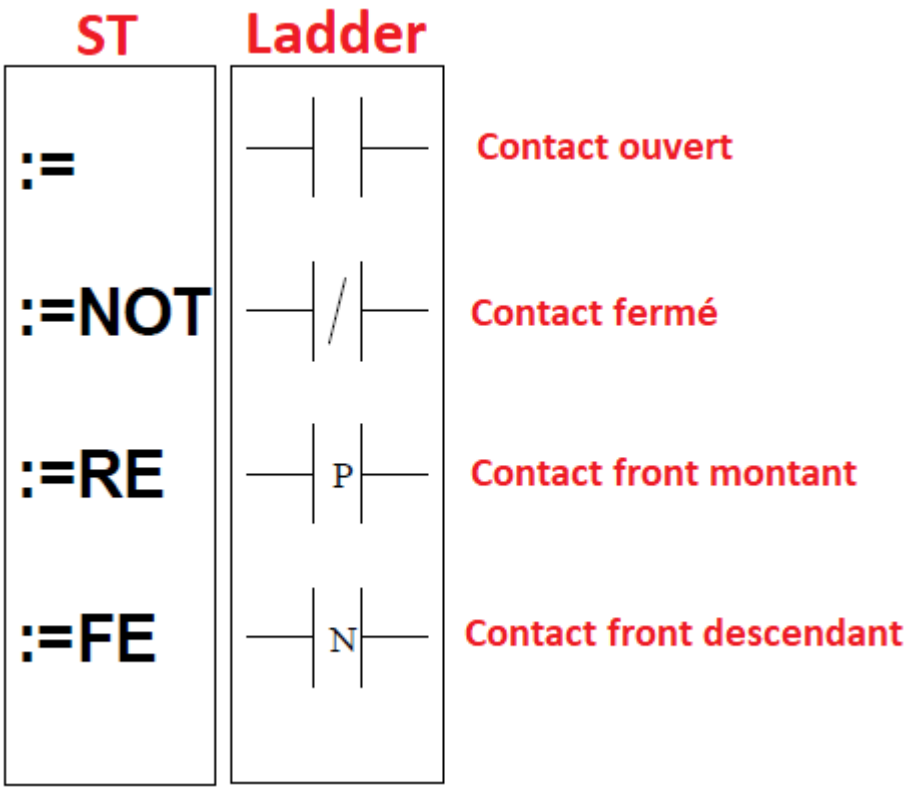
```
REPEAT
  <statement>;
UNTIL [boolean expression]
END_REPEAT;
```



X) Equivalence Ladder/Structured Text

Dans cette partie nous allons voir les équivalences des principaux symboles Ladder en langage Structured Text.

a) Les contacts

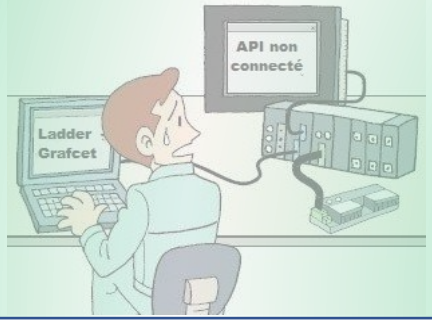


Exemples :

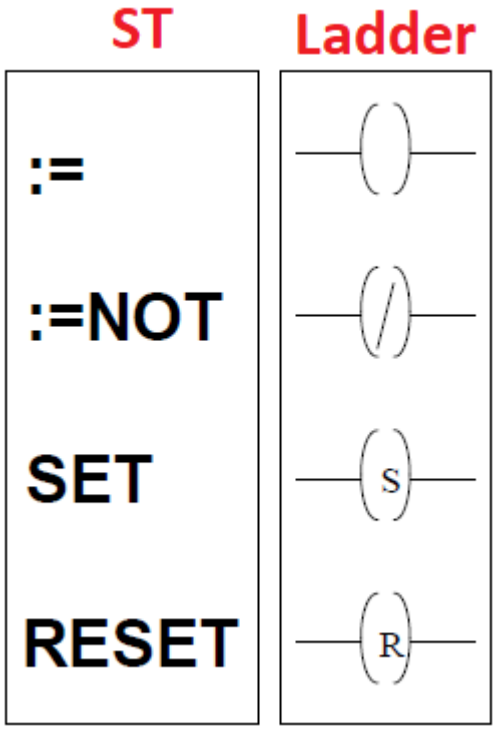
%M0 := %I0.2.0;

%M0 :=NOT %I0.2.0;

%M0 :=RE (%I0.2.0);



b) Les bobines



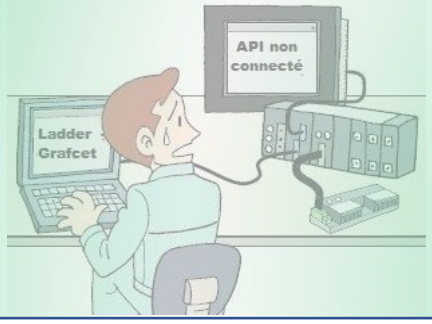
Exemples :

%MW100 := 123;

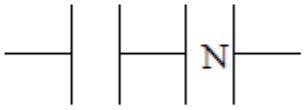
%Q0.4.0 :=NOT %M1;

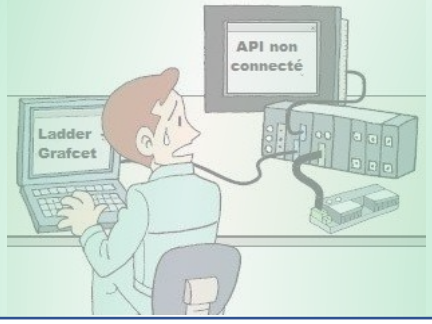
%M0 := TRUE;

SET (%Q0.4.0);



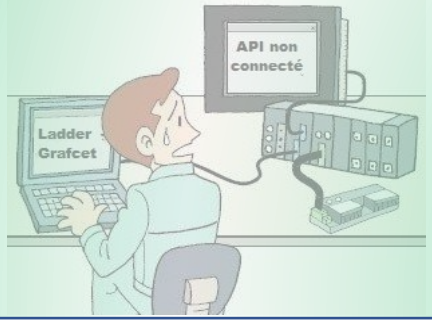
c) Le « ET logique »

ST	Ladder
AND	
AND(NOT...)	
AND(RE...)	
AND(FE...)	



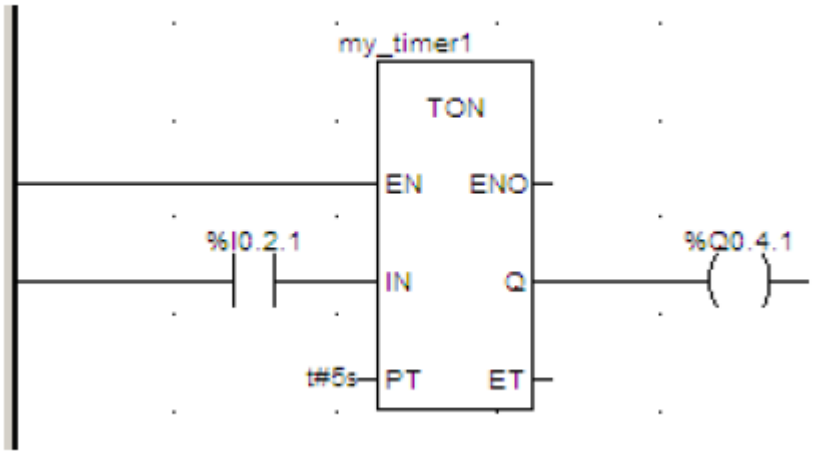
d) Le « OU logique »

ST	Ladder
OR	
OR(NOT...)	/
OR(RE...)	P
OR(FE...)	N



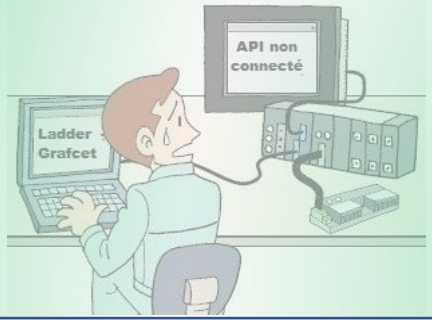
e) Le timer

Ladder



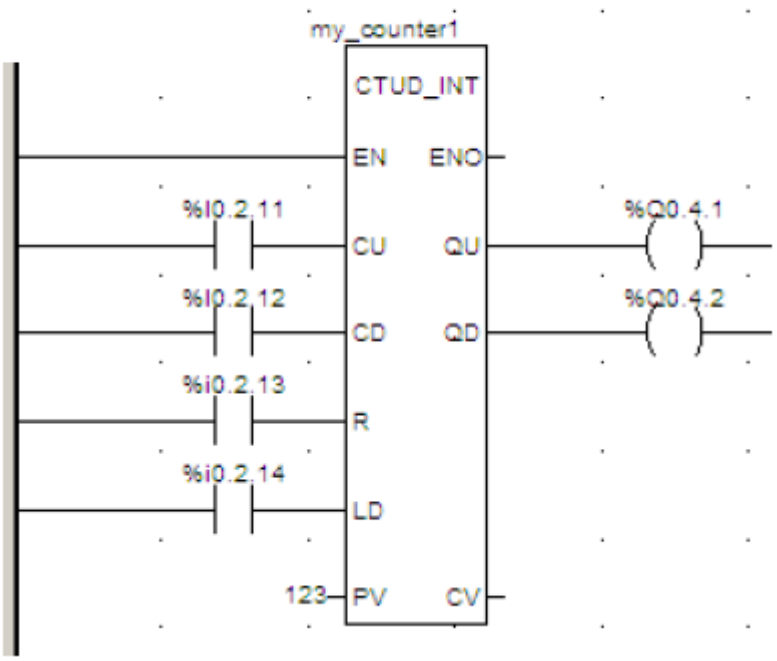
Structured Text

```
my_timer1 (IN := %I0.2.1 (*BOOL*),
           PT := t#5s (*TIME*),
           Q => %Q0.4.1 (*BOOL*),
           ET => my_var (*TIME*));
```



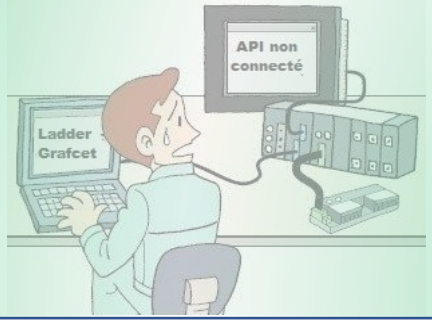
f) Le compteur/décompteur

Ladder



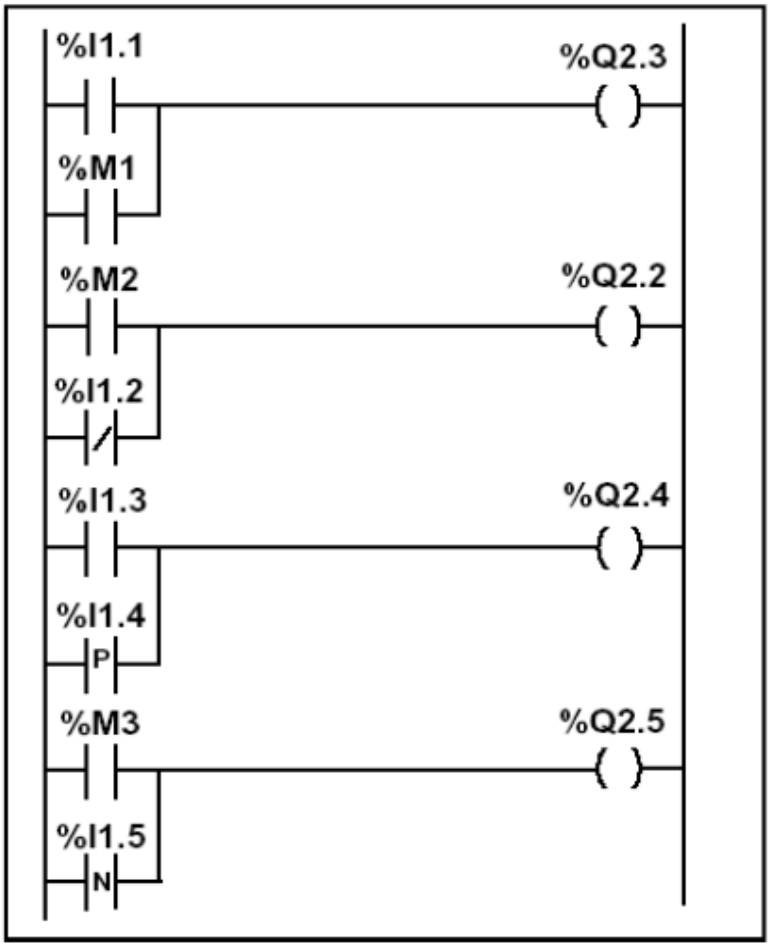
Structured Text

```
my_counter1 (CU := %I0.2.11 (*BOOL*),
             CD := %I0.2.12 (*BOOL*),
             R  := %I0.2.13 (*BOOL*),
             LD := %I0.2.14 (*BOOL*),
             PV := 123 (*INT*),
             QU => %Q0.4.1 (*BOOL*),
             QD => %Q0.4.2 (*BOOL*),
             CV => %MW100 (*INT*)) ;
```



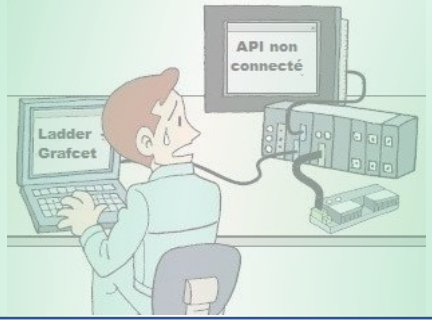
g) Traduction Ladder → Langage ST

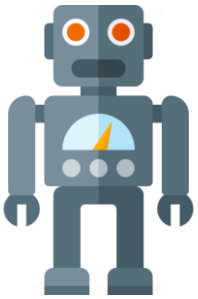
Ci-dessous l'équivalence en langage Structured Text du Ladder ci-dessous :



Structured text

```
%Q2.3:=%I1.1 OR %M1;
%Q2.2:=%M2 OR (NOT%I1.2);
%Q2.4:=%I1.3 OR (RE%I1.4);
%Q2.5:=%M3 OR (FE%I1.5);
```





Dans cette fiche, vous avez pu découvrir le langage Structured Text ainsi que sa syntaxe.

Vous pourrez désormais concevoir avec un peu de pratique des programmes en langage ST sans trop de difficultés.