1ère partie : modélisation

2ème partie : microcontrôleurs

3ème partie : API

Langage Structured Text – ST

1 – INTRODUCTION

2 – LA STRUCTURE DU PROGRAMME

3 – LES STRUCTURES DE CONTRÔLE

4 – LES FONCTIONS

1 – Introduction

2 – LA STRUCTURE DU PROGRAMME

3 – LES STRUCTURES DE CONTRÔLE

4 – LES FONCTIONS

Introduction

- Le langage Structured Text est un langage de haut niveau
  - Niveau équivalent au C
  - Basé sur Pascal
- Il définit toutes les instructions de type boucle ou test nécessaires à la structuration d'un programme

1 – INTRODUCTION

2 – La structure du programme

3 – LES STRUCTURES DE CONTRÔLE

4 – LES FONCTIONS

La structure du programme

- Les lignes d'instruction se terminent par un « ; », comme en C
- Les commentaires ont la forme suivante :

```
(* Ceci est
un commentaire
multiligne *)
```

- L'affectation utilise le symbole « := »

```
%Q1 := %I2 + %M3;
```

- Les éléments de structure (boucle, test) se présentent sous la forme suivante :

```
MOTCLE
    contenu
END_MOTCLE;
```

Les opérateurs de calcul

- Les opérateurs mathématiques de calcul sont disponibles :
  - `+ : Addition`
  - `- : Soustraction`
  - `* : Multiplication`
  - `/ : Division`
- Les opérateurs logiques pour les calculs sur les bits :
  - `NOT : Non`
  - `AND : Et`
  - `OR  : Ou`
  - `XOR : Ou exclusif`

Les opérateurs de comparaison

- `<` : Strictement inférieur
- `>` : Strictement supérieur
- `<=` : Inférieur ou égal
- `>=` : Supérieur ou égal
- `=` : Égal
- `<>` : Différent

Les opérateurs d'action

- Des opérateurs d'action permettent d'agir directement sur les variables :
  - `SET   : Mise à 1`
  - `RESET : Mise à 0`
  - `INC   : Incrémenter`
  - `DEC   : Décrémenter`

Étiquettes et sauts

- *Pour information uniquement – non recommandé*
- Il est possible de faire des sauts comme en assembleur
  - Pour cela, on définit un *label*
    - %L suivi d'une valeur
  - Puis on peut *sauter* à ce label

```
(* code *)
%L1 :
(* code *)
%L2 :
(* code *)
JUMP %L1;
(* code *)
JUMP %L2;
```

1 – INTRODUCTION

2 – LA STRUCTURE DU PROGRAMME

3 – Les structures de contrôle

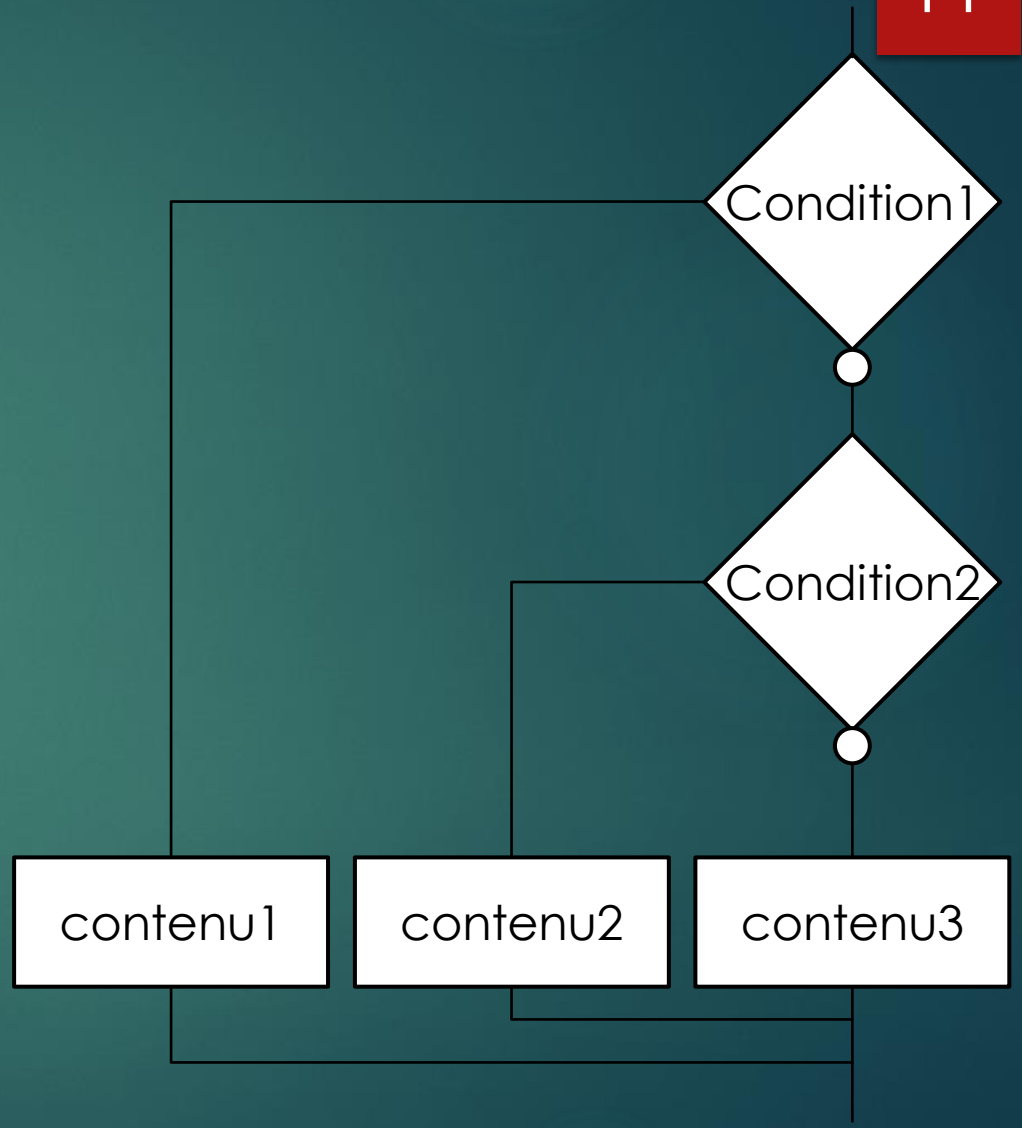
4 – LES FONCTIONS

Les tests

- Un test se présente sous la forme suivante :

```
IF condition1 THEN
    contenu1
ELSIF condition2 THEN
    contenu2
ELSE
    contenu3
END_IF;
```

- Avec *condition* une valeur booléenne



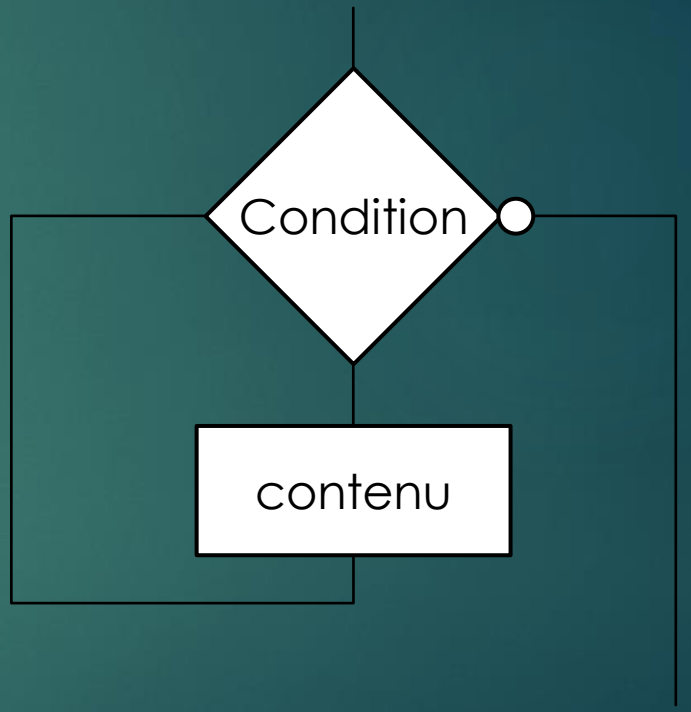
Choix parmi plusieurs

- Le case (équivalent du switch) permet de faire une sélection parmi plusieurs valeurs possible d'une variable

```
CASE variable OF
    1:
       // Cas 1
    2..4:
       // Cas 2 à 4
   ELSE
       // Correspond au default
END_CASE;
```

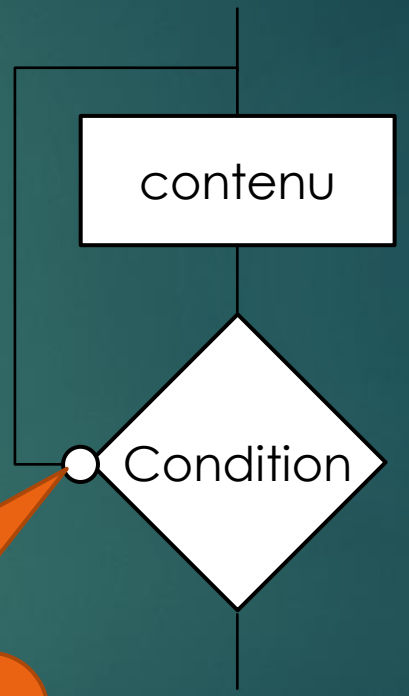
Les boucles : tant que


```
WHILE condition DO
    contenu
END_WHILE;
```



Les boucles : faire tant que

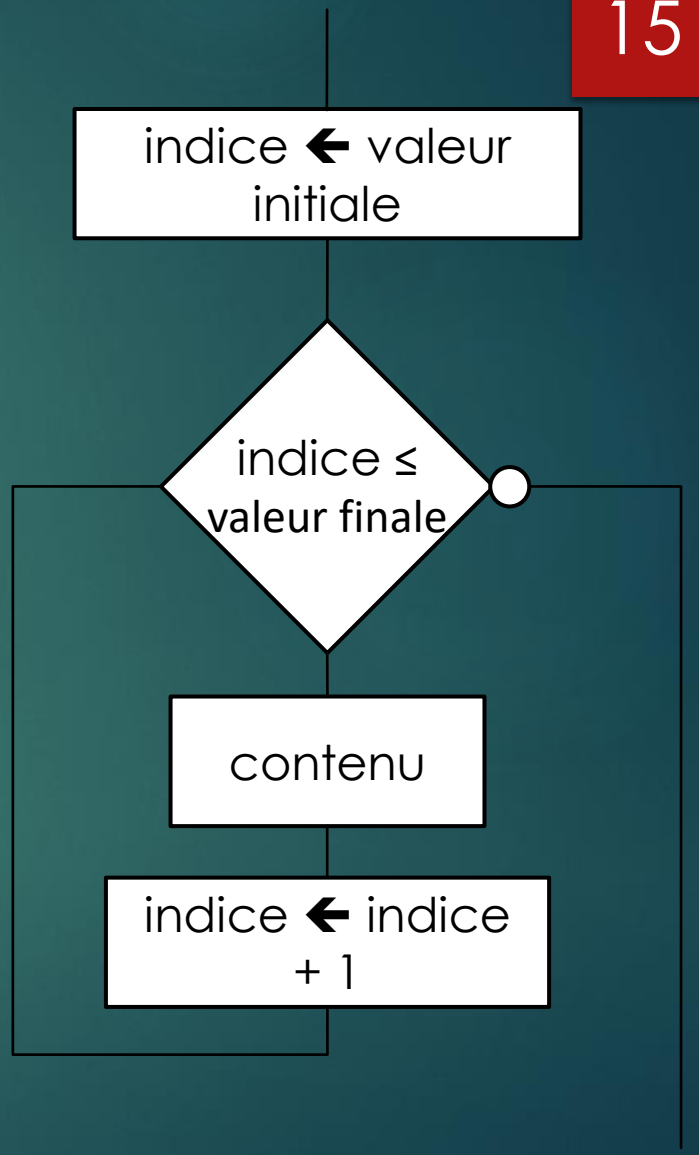

```
REPEAT
    contenu
UNTIL condition END_REPEAT;
```



Notez bien la position du « non » : on continue tant que la condition est *fausse*
La condition correspond au *maintient* dans la boucle

Les boucles : pour


```
FOR indice := valeur initiale TO valeur finale DO
    contenu
END_FOR;
```



4 – Les fonctions

Structure d'une fonction

- Deux types de fonctions
  - `FUNCTION_BLOCK`
  - `FUNCTION`
  - Correspondent aux deux types vus dans le standard

Structure d'une fonction

- La déclaration d'une fonction se présente comme ci-contre
- On note que l'on peut avoir plusieurs variables de sortie
  - Correspondent aux paramètres de sorties (de type pointeur) en C

Attention : dans TIA portal, la partie déclaration du programme est remplacée par les déclarations en en-tête

Variables d'entrée

Variables de sortie

Variables locales

```
FUNCTION nom fonction:

VAR_INPUT
     variable 1 : type;
     variable 2 : type;
END_VAR;

VAR_OUTPUT
     variable 3 : type;
     variable 4 : type;
END_VAR;

VAR
     variable 5 : type;
END_VAR;

contenu

END_FUNCTION;
```

Structure d'une fonction

- Le retour d'une fonction se fait comme en C, avec le mot-clé RETURN
- Attention, la valeur de retour est un concept différent des variables de sortie !
  - Le retour n'apparaît pas sur la vue externe de la fonction : il n'est possible de l'utiliser que dans un autre bloc ST (voir slide suivant)

```
FUNCTION nom fonction:

…
contenu
…

RETURN variable;

END_FUNCTION;
```

Appels de fonctions

- On appelle une fonction avec son nom et des parenthèses pour les variables
- Toutefois, contrairement au C, les variables ne se passent pas par position mais par nom
- On différentie les variables d'entrée des variables de sortie
  - Les variables de sortie doivent bien évidemment être associés à des variables et non à des valeurs brutes !

```
nom_fonction(paramIn1  := valeur1,
             paramIn2  := valeur2,
             paramOut1 => variable1,
             paramOut2 => variable2
            );
```

Appels de fonctions

- Si la fonction a un retour, on peut le récupérer dans une variable

```
variable3 = nom_fonction(paramIn1  := valeur1,
                         paramIn2  := valeur2,
                         paramOut1 => variable1,
                         paramOut2 => variable2
                        );
```