# Structures de contrôle

LICENCE EN INGENIERIE INFORMATIQUE
2020 – 2021

**Dr Ousmane DIALLO**
**odiallo@univ-zig.sn**

# Chap4: Les structures de contrôle

## 4.1. Contrôler le déroulement d'un programme

- Jusqu'à présent, les instructions de nos algorithmes ou programmes s'exécutaient de manière séquentielle et linéaire sans aucune contrainte ni condition.
- Nous allons dans ce chapitre voir comment agir sur l'ordre ou la fréquence d'exécution des instructions d'un algorithme et conséquemment d'un programme.
- Ceci se fera par l'utilisation de ***structures de contrôle*** qui sont classées principalement en deux catégories:
  - Les **structures alternatives**:

    **Si...**

    **Si...Sinon**

    **Cas ... parmi**

  - Les **structures répétitives ou boucles**:

    **Pour...faire**

    **Tant que ... faire**

    **Répéter... jusqu'à**

# Chap4: Les structures de contrôle

## 4.1.1. Les structures alternatives

- Une structure alternative permet de faire le **choix** entre une, deux ou plusieurs **actions** suivant qu'une certaine **condition** est remplie ou non.
- Une telle structure algorithmique est encore appelée **sélection**.

### A. La structure Si

- Dans la suite, on considéra au niveau des syntaxes que le terme instruction désigne soit :
  - Une **expression** suivie de **point virgule**
  - Un **appel de fonction** suivi de **point virgule**
  - Une **structure de contrôle** (cas des structures imbriquées)
  - Un **bloc d'instructions** (regroupant plusieurs instructions entre **Debut** et **Fin**)

**Syntaxe algo:**

***SI*** *(condition)* ***ALORS***

*Instruction*

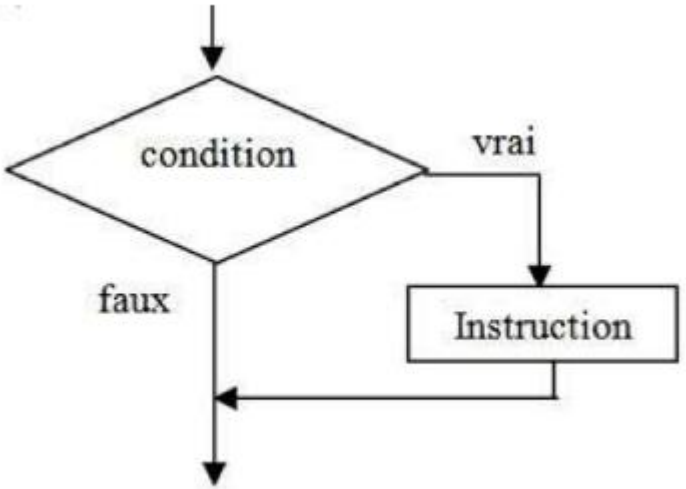
***FinSi***

# Chap4: Les structures de contrôle

## A. La structure Si

- ✓ Si la condition (***expression booléenne***) est vraie, l'instruction est exécutée puis le contrôle passe à l'instruction qui suit le **FinSi.**
- ✓ Si la condition est fausse, l'instruction n'est pas exécutée et le contrôle passe à l'instruction qui suit le **FinSi.**

**Organigramme**



# Chap4: Les structures de contrôle

## B. La structure Si … Sinon

**Syntaxe algo:**

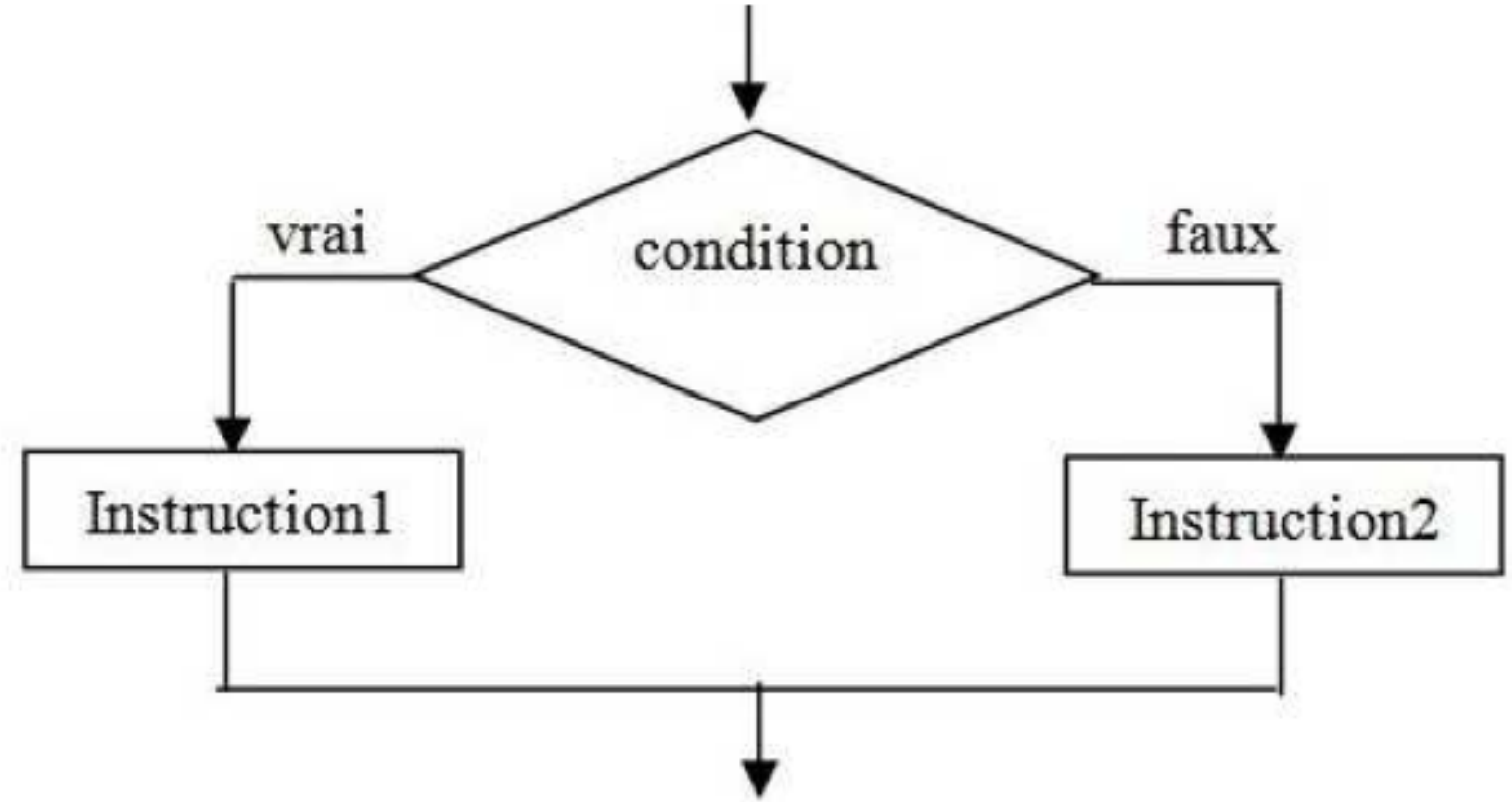
***SI*** *(condition)* ***ALORS***
*Instruction 1*
***Sinon***
*Instruction 2*
***FinSi***

- ✓ La condition est d'abord évaluée.
- ✓ Si elle est vraie, l'instruction1 est exécutée puis le contrôle passe à l'instruction qui suit le **FinSi**.
- ✓ Si elle est fausse, l'instruction2 est exécutée puis le contrôle passe à l'instruction qui suit le **FinSi**.
- ❑ Ici, l'exécution des instructions **instructions1** et **instructions2** est mutuellement exclusive ce qui signifie que seule une des deux instructions sera exécutée.

# Chap4: Les structures de contrôle

## B. La structure Si … Sinon

### Organigramme



# Chap4: Les structures de contrôle

**Exemple1:** *Signe d'un nombre*

```
Algorithm Signe_Nombre;
Variables
    n: réel ;
Début
    Ecrire("entrez un nombre")
    Lire(n)
    Si (n > 0) Alors        (*dans le cas où la condition n>0 est vraie*)
                Ecrire("valeur positive")
            Sinon  (*dans le cas où la condition n>0 est fausse*)
                Ecrire ("valeur négative ou nulle")
    FinSi

Fin.
```

# Chap4: Les structures de contrôle

**Application:** *Minimum entre deux nombres*

*Ecrire un algorithme qui cherche et affiche à l'écran le minimum entre deux nombres*

```
Algorithme minimum;
Variables
    nb1, nb2: réel ;
Début
    Ecrire("Entrez deux nombres: ")
    Lire(nb1, nb2)
    Si (nb1<nb2)  Alors
                Ecrire("Le minimum des deux nombres est: ", nb1)
                Sinon
                Ecrire("Le minimum des deux nombres est: ", nb2)
     FinSi
Fin.
```

# Chap4: Les structures de contrôle

## C. Structures imbriquées

- Il faut bien percevoir que la formulation de la **structure Si** n'est pas toujours sans ambiguïté. En effet, on peut avoir une **imbrication** au niveau des structures de contrôle.

**Exemple:**

**Si** (cond1) **Alors Si** (cond2) **Alors Si** (cond3) **Alors** instruction1 **Sinon** instruction2 **Sinon** instruction3 **Sinon** instruction4;

- Afin d'éviter des ambiguïtés de ce genre lors de la lecture d'un programme, il est vivement recommandé de bien mettre un **Alors** et un **Sinon** de la même structure alternative au même niveau vertical afin de ne pas les mélanger entre eux.
- On parle d'**indentation** ou de **paragraphage.**

```
Si (cond1) Alors
                Si (cond2) Alors
                                Si (cond3) Alors instruction1
                                                Sinon instruction2
                                 FinSi
                               Sinon instruction3
                 FinSi
             Sinon instruction4 ;
FinSi
```

# Chap4: Les structures de contrôle

## C. Structures imbriquées

- En général, une telle ambiguïté syntaxique est écartée définitivement soit en utilisant les parenthèses symboliques **Debut** et **Fin**, soit en respectant la règle suivante:

**Règle:**

*«La partie **Sinon** se rapporte toujours au mot réservé **Si** précédent le plus proche pour lequel il n'existe pas de partie **Sinon**. »*

- Dans une construction de structures alternatives imbriquées il doit y avoir autant de mots **Alors** que de mots **Si**.

# Chap4: Les structures de contrôle

**Exemple:** *Equation du premier degré avec C=0*

*Ecrire un programme qui résoud une équation du premier degré Ax+b=0.*

```
Algorithme Premier_Degre;
Variables
    A, B : réel;
Debut
   Ecrire('Entrez les coefficients A et B : ');
   Lire(A,B);
   Si (A=0) Alors
                  Si (B=0) Alors
                                 Ecrire('Tout réel est solution !')
                             Sinon
                                 Ecrire('Impossible !')
                   FinSi
              Sinon
                  Ecrire('La solution est : ',-B/A);
    FinSi
End.
```

# Chap4: Les structures de contrôle

## C. Structures imbriquées

### Remarque

- La structure **Si** peut contenir des clauses **Sinon Si** suivant le schéma suivant :

**Si** (condition1) **Alors**
Instruction1
**Sinon Si** (condition2) **Alors**
Instruction2
…
**Sinon Si** (conditionN) **Alors**
InstructionN
**Sinon**
Instruction
**FinSi**

## C. Structures imbriquées

### Remarque

- Les conditions sont évaluées dans l'ordre d'apparition. Dès qu'une condition est vraie, le traitement associé est exécuté.
- L'instruction suivante à exécuter sera alors celle qui suit le **FinSi**.
- Si aucune condition n'est vérifiée, alors le traitement associée au **Sinon**, s'il existe, est exécuté.

# Chap4: Les structures de contrôle

## D. La sélection multiple: Cas … parmi ou Selon … Faire

- Elle permet de choisir l'instruction à effectuer en fonction de la valeur d'une variable ou d'une expression
- Par exemple, au cas où un menu est proposé à l'utilisateur **(1. pour lire, 2. pour écrire, 3. pour calculer, 4. pour sortir, etc.)**, il est important de **tester** si l'utilisateur a tapé 1, 2, 3 ou 4.
- Au lieu d'utiliser plusieurs **Si… Alors… Sinon…** Imbriqués, il est préférable de choisir une sélection multiple.

Ainsi au lieu d'écrire:

```
Si reponse=1 Alors
        Instructions de lecture…
        Sinon
          Si reponse=2 Alors
                Instructions d'écriture…
                Sinon
                  Si reponse=3 Alors
                        instructions de calcul…
```

Il serait préférable d'écrire:

```
Selon reponse Faire
        1 : Instructions de lecture…
        2 : Instructions d'écriture…
        3 : instructions de calcul…
FinSelon
```

# Chap4: Les structures de contrôle

## D. La sélection multiple: Cas … parmi ou Selon … Faire

**Syntaxe**

**CAS** variable **PARMI**

constante1 : suite d'instructions1
constante2 : suite d'instructions2
intervalle1 : suite d'instructions3
…
**SINON**
suite instructions par défaut
**FIN;**

**Selon** variable **Faire**

constante1 : suite d'instructions1
constante2 : suite d'instructions2
intervalle1 : suite d'instructions3
…
**SINON**
suite instructions par défaut
**FinSelon**

# Chap4: Les structures de contrôle

## D. La sélection multiple: Cas … parmi ou Selon … Faire

- La variable (ou expression) est évaluée, puis sa valeur est successivement comparée à chacune des valeurs.
- Dès qu'il y a correspondance, les comparaisons sont arrêtées et le traitement associé est exécuté.
- Si aucune valeur ne correspond à la valeur de l'expression, alors le traitement associé au **Sinon**, s'il existe, est exécuté.

# Chap4: Les structures de contrôle

## D. La sélection multiple: Cas … parmi ou Selon … Faire

### À retenir:

- Comme avec l'instruction **Si**, l'exécution de chaque branche est **mutuellement exclusive**.
- La variable « variable » est appelée **sélecteur** et doit être d'un **type scalaire** (caractère ou entier).
- Les constantes **CAS** doivent être toutes différentes et du même type que le sélecteur. Elles sont interprétées comme des **étiquettes**.
- Seules les égalités sont possibles au niveau du test ***(Pas de comparaisons de type <,>,<=,>= ou <>.*** On peut néanmoins utiliser des **intervalles**.
- On peut donner une liste de constantes, ou des intervalles de constantes.
- **Attention**, chaque valeur possible ne doit être représentée qu'une fois au plus (sinon il y a erreur à la compilation).
  - Par exemple, on ne peut pas faire des intervalles se chevauchant, comme **3..6 et 5..10**, les cas 5 et 6 étant représentés 2 fois.

# Chap4: Les structures de contrôle

**Exemple:**

*Programme qui affiche le mois en toute lettre selon son numéro.*

*Le numéro du mois est mémorisé dans la variable mois.*

```
Algorithme GererMois;
variables mois : entier;
Debut
        Ecrire("Donner le numéro du mois")
        Lire(mois)
        Selon (mois) Faire
        1 : Ecrire("Janvier")
        2 : Ecrire("Février")
        3 : Ecrire("Mars")
        4 : Ecrire("Avril")
        …
        11: Ecrire("Novembre")
        12: Ecrire("Décembre")
        Sinon Ecrire("Un numéro de mois doit être compris entre 1 et 12")
        FinSelon
```

# Chap4: Les structures de contrôle

## **Application:** *Simuler une calculatrice*

```
Algorithme calculette;
variables A,B : réel;
     RESULTAT : réel;
     TOUCHE : caractère;
Debut
   Ecrire('entrez une opération ');
   Ecrire('(taper un nombre, un opérateur puis un nombre): ');
    Lire(A,TOUCHE,B);
    Selon TOUCHE Faire
          '+' : RESULTAT ← A+B;
          '-' : RESULTAT ← A-B;
          '*' : RESULTAT ← A*B;
          '/' : RESULTAT ← A/B;
      Sinon Ecrire('Operateur inexistent!');
     FinSelon;
     Ecrire(A, TOUCHE, B,' = ', RESULTAT);
end.
```

# Chap4: Les structures de contrôle

## 4.1.2. Les structures répétitives

- Les structures répétitives, appelées aussi **boucles**, permettent de répéter un traitement (c'est à dire une instruction simple ou composée) autant de fois qu'il est nécessaire
- On distingue les boucles à bornes définies (**POUR...FAIRE**) et les boucles à bornes non définies (**TANT QUE...FAIRE** et **RÉPÉTER...JUSQU'À)**.
- Toute structure répétitive est composée de trois éléments:
  - d'une **initialisation** d'un **compteur** ;
  - d'une **condition d'arrêt ou de continuité**;
  - d'un **bloc d'instructions**.
- Toute modification d'un quelconque de ces trois éléments nécessite un contrôle de cohérence des deux autres.

# Chap4: Les structures de contrôle

## A. Boucle à bornes définies (POUR...FAIRE)

- Avec cette boucle, nous connaissons le nombre de répétitions à effectuer, grâce aux valeurs des bornes minimale et maximale fournies dans la définition de la boucle.
- Un indice de boucle varie alors de la valeur minimale (initiale) jusqu'à la valeur maximale (finale).

**Syntaxe algo:** **POUR** *compteur* = *borne_min* **À** *borne_max*

**par pas de** *increment* **FAIRE** <*Séquence d'Instructions*>

**FinPour**

- La variable *compteur* doit être de type scalaire (entier, énuméré, intervalle ou caractère) elle ne peut pas être réelle.
- Elle est initialisée à la valeur *borne_min*.
- Le compteur augmente (**implicitement**) de la valeur *increment* à chaque exécution du traitement.
- Lorsque la variable *compteur* vaut la valeur *borne_max*, l'instruction est exécutée une dernière fois puis le programme sort de la boucle.
- Par défaut, l'incrément est de 1

# Chap4: Les structures de contrôle

**Exemples:**

```
Algorithme boucle_for;
Variables        i:integer;
Debut
   Pour i =1 A 5 Faire
        Ecrire('le carré de ', i, ' est :', i*i);
   FinPour

Ecrire('FIN. A la prochaine…');
Fin.
```

**Il est possible d'imbriquer plusieurs boucles Pour:**

```
Algorithme table_multiplication;
Variables
    i, j : entier;
Debut
    Pour i = 1 à 10 Faire
        debut
            Pour j = 1 à 10 Faire
            Ecrire(i*j);
            EcrireSautLigne;
            FinPour
         Fin;
Fin.
```

# Chap4: Les structures de contrôle

**Exemples:**

```
Algorithme boucle_for;
Variables        i:integer;
Debut
   Pour i =1 A 5 Faire
        Ecrire('le carré de ', i, ' est :', i*i);
   FinPour

Ecrire('FIN. A la prochaine…');
Fin.
```

**Il est possible d'imbriquer plusieurs boucles Pour:**

```
Pour X1 = 1 A 10 Faire
    Debut
        …
        Pour X2 = 1 A 15 Faire
            Debut
              …
            Fin;
         FinPour
        …
FinPour
```

```
Algorithme table_multiplication;
Variables
    i, j : entier;
Debut
    Pour i = 1 à 10 Faire
        debut
            Pour j = 1 à 10 Faire
            Ecrire(i*j);
            EcrireSautLigne;
            FinPour
         Fin;
Fin.
```

# Chap4: Les structures de contrôle

## B. Boucle à bornes non définies

- Lorsque les bornes ne sont pas connues, il existe deux autres types de boucles:

- **Boucle TANT QUE ... FAIRE …**

**Syntaxe algo:** **TANT QUE** (condition) **FAIRE**

<Séquence d'Instructions>

**FinTQ**

**Remarques**

- La boucle Tant que permet de répéter un traitement tant qu'une **condition** est vraie.
- Arrêt si **condition** est fausse

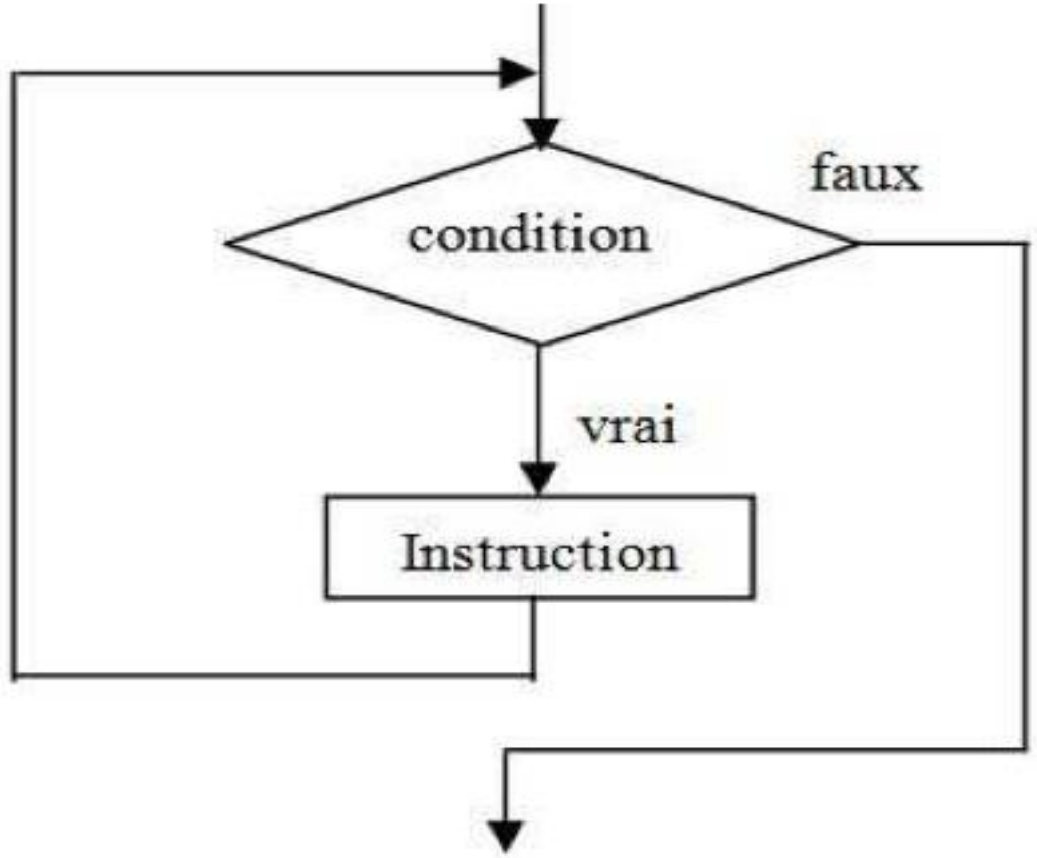
  ⇒Si la condition est fausse au départ, alors le traitement ne sera pas exécuté
- Incrémentation gérée par le programmeur lui-même

  ⇒Pas d'augmentation automatique d'une variable (contrairement à la boucle FOR)
- Les variables de l'expression **condition** doivent être initialisées avant le **Tant Que**, pour qu'au premier passage **condition** puisse être évaluée.

# Chap4: Les structures de contrôle

## Boucle TANT QUE … FAIRE

## Organigramme



# Chap4: Les structures de contrôle

## Exemple:

```
program boucle_TantQue;
variables
   i:entier;
Debut
    i←1;
    Tant Que (i <= 5) Faire
        Ecrire('le carré de ', i, ' est :', i*i);
        EcrireSautLigne;
       i ← i+1; (*incrémentation gérée par le programmeur*)
    FinTq;
       EcrireSautLigne;
        Ecrire('FIN. A la prochaine…');
Fin.
```

```
program boucle_TantQue;
variables
   i:entier;
Debut
    i←1;
    Tant Que (i <= 5) Faire
    Debut
        Ecrire('le carré de ', i, ' est :', i*i);
        EcrireSautLigne;
       i ← i+1; (*incrémentation gérée par le programmeur*)
      Fin;
       EcrireSautLigne;
        Ecrire('FIN. A la prochaine…');
Fin.
```

# Chap5: Les structures de contrôle

## B. Boucle à bornes non définies

### Boucle REPETER … JUSQU'À

**Syntaxe algo:** **REPETER**

<Séquence d'Instructions>

**JUSQU'À** (condition)

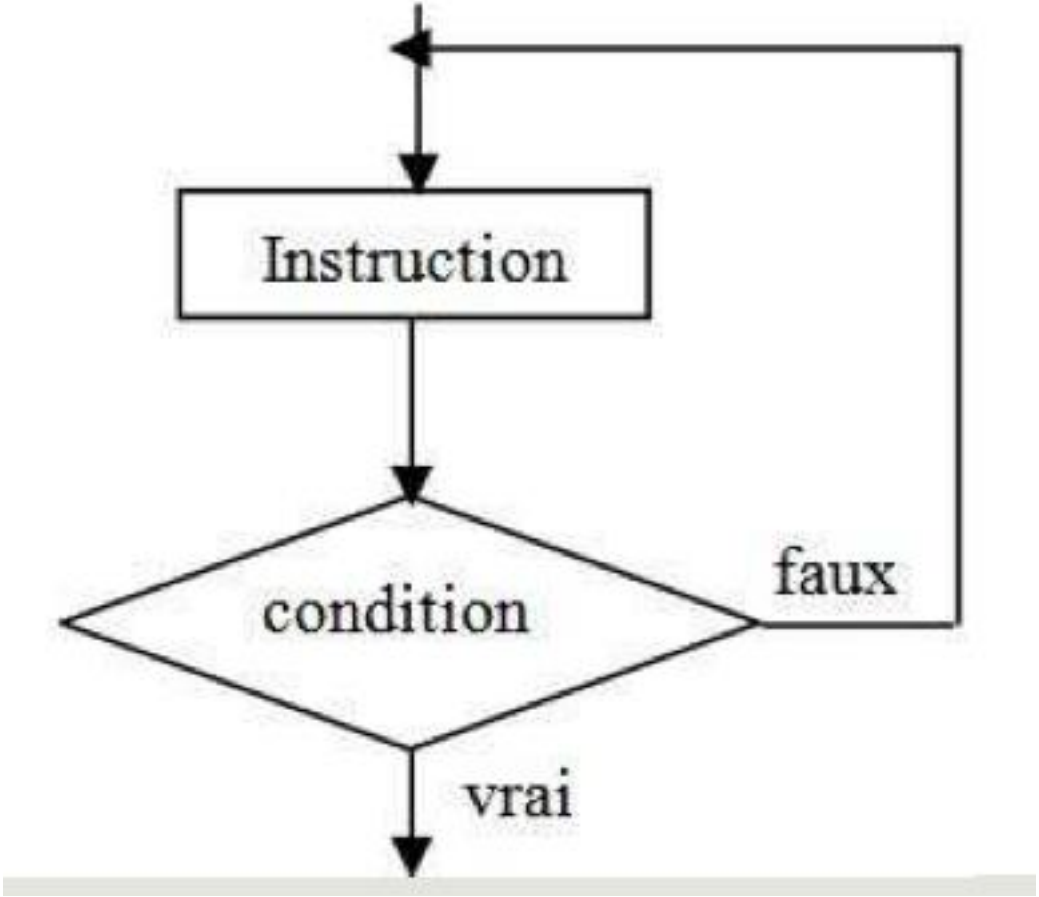
### Remarques

- Cette boucle sert à répéter une instruction ou un bloc d'instruction jusqu'à ce qu'une condition soit remplie.
  - La boucle s'effectue donc tant que l'expression est fausse, arrêt quand l'expression est vraie. C'est le contraire de la boucle **Tant Que**.
- Contrairement au **Tant Que**, il y a au moins un passage (1 boucle), même si l'expression est vraie ;
- De même que pour le **Tant Que**, c'est le programmeur qui gère l'incrémentation.

# Chap4: Les structures de contrôle

## Boucle REPETER … JUSQU'À

## Organigramme



# Chap4: Les structures de contrôle

## Exemple:

```
Algorithme boucle_repeat;
variable
   i: entier;
Debut
i ←1;
   Repeter
         Ecrire('le carré de ', i, ' est :', i*i);
          EcrireSautLigne;
          i ← i+1; { incrémentation gérée par le programmeur }
   Jusqu'à (i>5);
   EcrireSautLigne;
   Ecrire('FIN. A la prochaine…');
Fin.
```

# Chap4: Les structures de contrôle

**Remarques**

- **Faire attention** aux **conditions initiales**, aux **conditions d'arrêt** et à l'**incrémentation** sinon la boucle risque d'être infinie.
- Les deux boucles peuvent être choisies indifféremment. Cependant, l'une est le contraire de l'autre, au niveau de la **condition d'arrêt**:

**TANT QUE condition1 FAIRE <Bloc d'instructions>**

**⇐⇒**

**REPETER <Bloc d'instructions> JUSQU'A non (condition1)**

- Tant que condition1 est vraie, faire bloc d'instructions…
- Répéter bloc d'instructions, jusqu'à ce que condition1 ne soit plus vraie

- Dans ce cas, la condition d'arrêt de la boucle **TANT QUE** est l'opposée de la condition d'arrêt de la boucle **REPETER**.

**Exemple:**

```
TANT QUE (i ≠ 10) FAIRE
    i←i+1 {on fait varier i jusqu'à 10}

            Est équivalent à :

REPETER
     i ←i+1
JUSQU'À (i=10)
```

# Chap4: Les structures de contrôle

**Remarque**

- Les boucles **REPETER** et **TANT QUE** peuvent être utilisées même si les bornes sont définies. Dans ce cas, **il est bien entendu préférable d'utiliser une boucle POUR**.

**Exemple comparatif**

- Dans cet exemple, un même algorithme sera traité avec les trois boucles étudiées

**Algorithme:** Reconstruire l'opération de multiplication, en effectuant des sommes successives. Soit à effectuer le produit des entiers naturels a et b (distincts de 0).

**Données:** a multiplicande, b multiplicateur

**Résultat:** P produit

**Méthode:** ajouter b fois le multiplicande

# Chap5: Les structures de contrôle

## Exemple comparatif:

```
Algorithme Avec_Pour;
Variables a,b,i,P:entier;
Debut
    Ecrire('Donner a et b');
    Lire(a,b);
    P←0;
    Pour i ←1 à b Faire
        P:=P+a;
    FinPour
    Ecrire('Le produit est',P);
Fin.
```

```
Algorithme Avec_TANTQUE;
variables a,b,i,P:entier;
Debut
    Ecrire('Donner a et b');
    Lire(a,b);
    P ← 0;
    i ←1;
    TantQue (i<=b) Faire
      Debut
        P← P+a;
        i ←i+1;
      Fin;
   Ecrire('Le produit est',P);
Fin.
```

```
Algorithme Avec_REPETER;
variables a,b,i,P:entier;
Debut
   Ecrire('Donner a et b');
   Lire(a,b);
   P← 0;
   i ← 1;
   Repeter
      P← P+a;
      i ←i+1;
   Jusqu'à (i>b);
   Ecrire('Le produit est',P);
Fin.
```

# Fin
# Structures de contrôle

LICENCE EN INGENIERIE INFORMATIQUE
2020 – 2021

**Dr Ousmane DIALLO**
**odiallo@univ-zig.sn**