# Algorithmique et Programmation

L1 MPI-PC

Dr Ousmane DIALLO

# Chap5: Les structures de contrôle

## 5.1. 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 dans quelle mesure il sera possible de contrôler le déroulement d'un algorithme et conséquemment celui d'un programme.
- Ceci se fera donc par l'utilisation de structures de contrôle. Nous en présenterons principalement deux catégories:
  - Les structures alternatives:

    **Si... (if...)**

    **Si...Sinon (if...else)**

    **Cas ... parmi (case...of)**

  - Les structures répétitives:

    **Pour...faire (for... do)**

    **Tant que ... faire (while...do)**

    **Répéter... jusqu'à (repeat...until)**

# Chap5: Les structures de contrôle

## 5.1.1. Les structures alternatives

- Une instruction 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. L'instruction 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 **instruction de contrôle** (cas des instructions imbriquées)
  - Un **bloc d'instructions** (regroupant plusieurs instructions, il commence par **Begin** et se termine par **End**; ou **Debut** et **Fin**; en Algo)

**Syntaxe algo:** **SI** <condition> **ALORS** Instruction

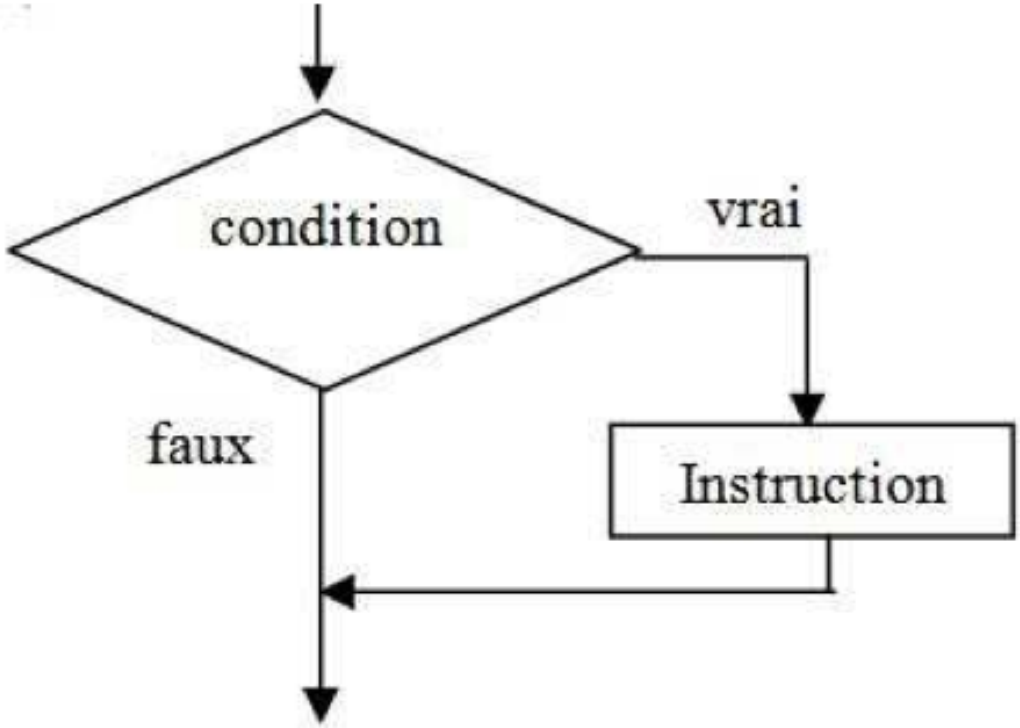
**Syntaxe Pascal:** **If** <condition> **Then** Instruction

- L'instruction **Si** permet de traiter l'**instruction** uniquement si la **condition** (l'expression booléenne) est vérifiée.

# Chap5: Les structures de contrôle

## A. L'instruction Si

### Organigramme



# Chap5: Les structures de contrôle

## B. L'instruction Si … Sinon

**Syntaxe algo:** **SI** <condition> **ALORS** Instruction 1
**Sinon** Instruction 2

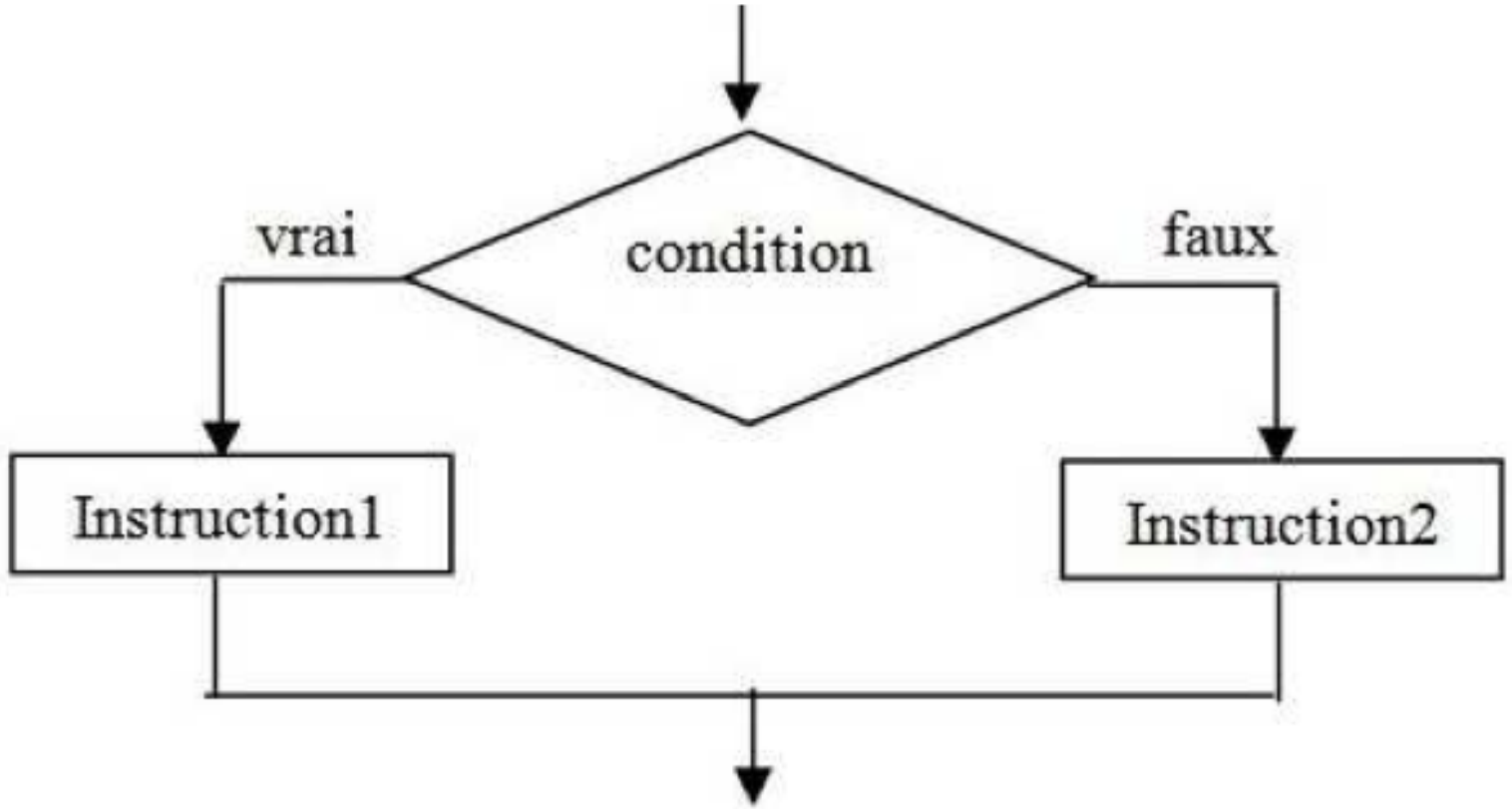
**Syntaxe Pascal:** **If** <condition> **Then** Instruction 1
**Else** Instruction 2

- L'instruction **Si…Sinon** permet de traiter l'**instruction1** si la condition est vérifiée et l'**instruction2** si la **condition** n'est pas vérifiée.
- Ici, l'exécution des instructions **instructions1** et **instructions2** est mutuellement exclusive ce qui signifie que seule une des deux instructions sera exécutée.
- **ATTENTION: PAS DE POINT VIRGULE AVANT UN ELSE !!!**

# Chap5: Les structures de contrôle

## B. L'instruction Si … Sinon

### Organigramme



# Chap5: Les structures de contrôle

**Exemple:** *Calcul d'une racine carrée (avec SI … ALORS)*

```
Algorithm SI_ALORS;
Variables
      x: réel ; (*opérande*)
      r: réel ; (*résultat de la racine carrée*)
Début
      Ecrire ('Saisir le nombre x');
      Lire (x);
      Si (x>0)    Alors
                  Début
                  r := racine (x);
                  Ecrire (r);
                  Fin
Fin.
```

```
program SI_ALORS;
Var
      x: real ; (*opérande*)
      r: real ; (*résultat de la racine carrée*)
Begin
      Write ('Saisir le nombre x');
      Read(x);
      If (x>0)    Then
                  Begin
                  r := SQRT (x);
                  Write (r);
                  End;
End.
```

# Chap5: Les structures de contrôle

**Exemple:** *Calcul d'une racine carrée (avec SI … ALORS … SINON)*

```
Algorithm SI_ALORS_SINON;
Variables
      x: réel ; (*opérande*)
      r: réel ; (*résultat de la racine carrée*)
Début
      Ecrire ('Saisir le nombre x');
      Lire (x);
      Si (x<0)    Alors
                    Ecrire ('x est négatif')
                  Sinon
                    Début
                        r := racine (x);
                        Ecrire (r);
                    Fin ;
Fin.
```

```
program SI_ALORS_SINON;
Var
      x: real ; (*opérande*)
      r: real ; (*résultat de la racine carrée*)
Begin
      Write ('Saisir le nombre x');
      Read (x);
      If (x<0)    Then
                    Write ('x est négatif')
                  Else
                    Begin
                        r := SQRT (x);
                        Write (r);
                    End;
End.
```

# Chap5: Les structures de contrôle

## C. Instructions imbriquées

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

**Exemple:**

**IF** (cond1) **THEN IF** (cond2) **THEN IF** (cond3) **THEN** instruction1 **ELSE** instruction2 **ELSE** instruction3 **ELSE** instruction4;

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

```
IF (cond1) THEN
        IF (cond2) THEN
                IF (cond3) THEN instruction1
                        ELSE instruction2
                ELSE instruction3
        ELSE instruction4 ;
```

## C. Instructions imbriquées

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

**Règle:**

*«La partie **ELSE** se rapporte toujours au mot réserve **IF** précédent le plus proche pour lequel il n'existe pas de partie **ELSE**. »*

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

# Chap5: Les structures de contrôle

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

*Ecrire un programme qui résous une équation du premier degré Ax+b=0 qui lit les valeurs de A et B entrées par l'utilisateur.*

```
Program Premier_Degre;
 Var
    A, B : real;
Begin
   write('Entrez les coefficients A et B : ');
   readln(A,B);
if (A=0) then { Èvaluation de la condition}
            if (B=0) then
                        writeln('Tout réel est solution !')
                        else
                          writeln('Impossible !')
             else writeln('La solution est : ',-B/A:10:3);
End.
```

# Chap5: Les structures de contrôle

## D. La sélection multiple: Case … of

- Elle est utilisée pour tester une solution parmi N. 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 **IF…THEN… ELSE…** Imbriqués, il est préférable de choisir une sélection multiple. Ainsi au lieu d'écrire:

```
IF reponse=1 THEN
        Instructions de lecture…
        ELSE
          IF reponse=2 THEN
                Instructions d'écriture…
                ELSE
                  IF reponse=3 THEN
                        instructions de calcul…
```

Il serait préférable d'écrire:

```
CASE reponse OF
        1 : Instructions de lecture…
        2 : Instructions d'écriture…
        3 : instructions de calcul…
End;
```

**Syntaxe algo:**

```
CAS variable PARMI
        constante1 : suite d'instructions1
        constante2 : suite d'instructions2
        intervalle1 : suite d'instructions3
        …
       SINON suite instructions par défaut
FIN;
```

**Syntaxe Pascal:**

```
CASE variable OF
        constante1 : suite d'instructions1
        constante2 : suite d'instructions2
        intervalle1 : suite d'instructions3
        …
       ELSE suite instructions par défaut
END;
```

# Chap5: Les structures de contrôle

## D. La sélection multiple: Case … of

### À retenir:

- Comme avec l'instruction **IF**, 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 **CASE** 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.

# Chap5: Les structures de contrôle

## **Exemple:** *Simuler une calculatrice*

```
Program calculette;
var A,B : real;
     RESULTAT : real;
     TOUCHE : char;
Begin
   write('entrez une opération ');
   write('(taper un nombre, un opérateur puis un nombre): ');
    readln(A,TOUCHE,B);
    case TOUCHE of
          '+' : RESULTAT:= A+B;
          '-' : RESULTAT:= A-B;
          '*' : RESULTAT:= A*B;
          '/' : RESULTAT:= A/B;
     end;
     writeln(A,TOUCHE, B,' = ', RESULTAT);
end.
```

# Chap5: Les structures de contrôle

## 5.1.2. Les structures répétitives

- Une structure répétitive permet d'exécuter une séquence d'instructions un certain nombre de fois jusqu'à ce qu'une condition déterminée soit remplie ou non. La répétition se fait par l'intermédiaire de **boucles** et d'**itérations**.
  - Une **boucle** consiste à parcourir une partie d'un programme un certain nombre de fois.
  - Une **Itération** est la répétition d'un même traitement plusieurs fois.
- *La même séquence d'instructions est réitérée plusieurs fois au cours d'une même exécution.*
- 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.

# Chap5: Les structures de contrôle

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

- Dans le cas d'une boucle à bornes définies, nous connaissons le nombre d'itérations 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

**FAIRE** <Séquence d'Instructions>

**Syntaxe Pascal:** **FOR compteur:=** borne_min **TO** borne_max

**DO** <Séquence d'Instructions>**;**

**FOR compteur:=** borne_max **downto** borne_min

**DO** <Séquence d'Instruction>**;**

- La variable **compteur** doit être de type scalaire (entier, énuméré, intervalle ou caractère) elle ne peut pas être réelle. si borne_min > borne_max le **FOR** de la première syntaxe n'est pas exécuté.

# Chap5: Les structures de contrôle

## Exemples:

```
program boucle_for;
Var        i:integer;
begin
   for i:=1 to 5 do
         writeln('le carré de ', i, ' est :', sqr(i));
writeln;

writeln('FIN. A la prochaine…');
end.
```

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

```
For X1 := C1 to C2 do
     Begin
          …
          For X2 := D1 to D2 do
               Begin
                  …
               End;
          …
End;
```

```
PROGRAM table_multiplication;
 VAR
    i, j : integer;
BEGIN
    for i := 1 to 10 do
         begin
            for j := 1 to 10 do
            write (i*j : 3);
            writeln;
          end;
END.
```

# Chap5: 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>

**Syntaxe Pascal:** **WHILE** <condition> **DO**
<Séquence d'Instructions>

**Remarques**

- Arrêt si **condition** est fausse

⇒Pas de boucle si faux au départ

- Incrémentation gérée par le programmeur lui-même

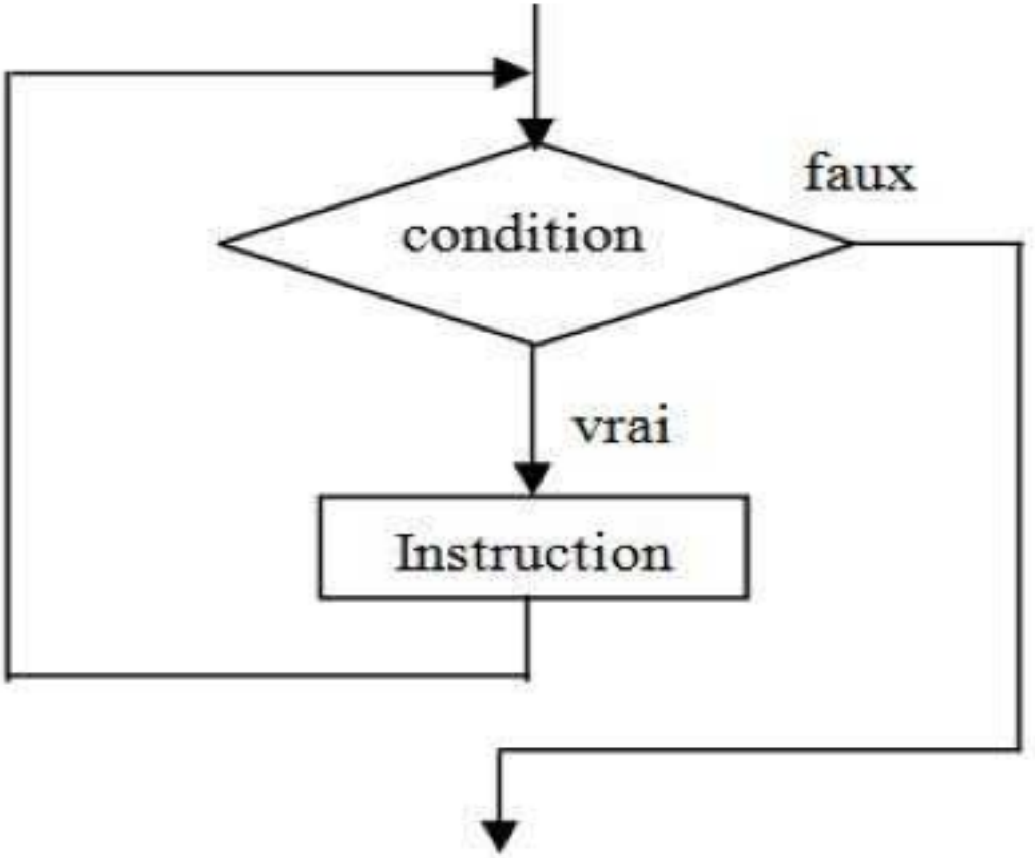
⇒Pas d'augmentation automatique d'une variable (contrairement à la boucle FOR)

- Les variables de l'expression **condition** doivent être initialisées avant le **while**, pour qu'au premier passage **condition** puisse être évaluée.

# Chap5: Les structures de contrôle

## Boucle TANT QUE … FAIRE

## Organigramme



## Exemple:

```
program boucle_while;
var
   i:integer;
begin
    i:=1;
    while i <= 5 do
    begin
      writeln('le carré de ', i, ' est :', sqr(i));
       i:=i+1; {incrémentation gérée par le programmeur}
     end;
       writeln;
        writeln('FIN. A la prochaine…');
end.
```

# Chap5: Les structures de contrôle

## B. Boucle à bornes non définies

**Boucle REPETER … JUSQU'À**

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

<Séquence d'Instructions>

**JUSQU'À** <condition>

**Syntaxe Pascal:** **REPEAT**

<Séquence d'Instructions>

**UNTIL** <condition>

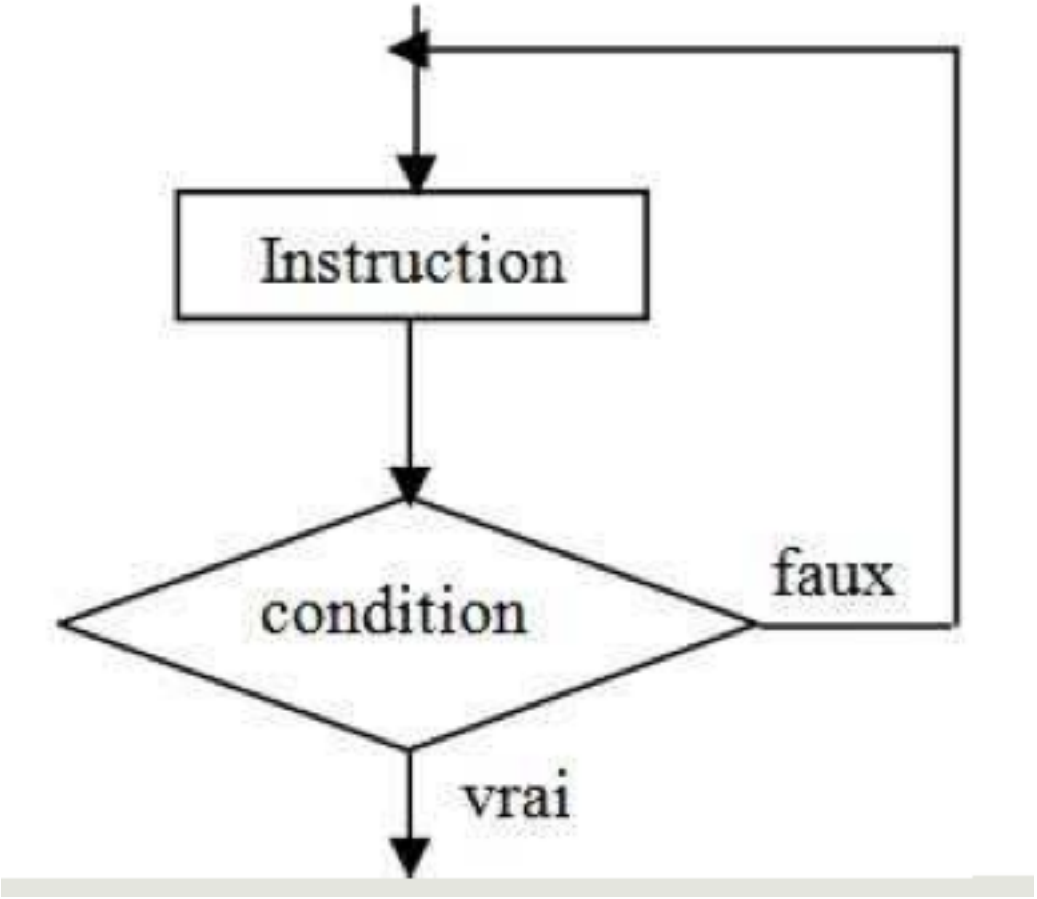
**Remarques**

- La boucle s'effectue tant que l'expression est fausse, arrêt quand l'expression est vraie. C'est le contraire de la boucle **WHILE**.
- Contrairement au **WHILE**, il y a au moins un passage (1 boucle), même si l'expression est vraie ;
- De même que pour le **WHILE**, c'est le programmeur qui gère l'incrémentation.

# Chap5: Les structures de contrôle

## Boucle REPETER … JUSQU'À

## Organigramme



## Exemple:

```
program boucle_repeat;
var
  i:integer;
begin
i:=1;
   repeat
        writeln('le carré de ', i, ' est :', sqr(i));
         i:=i+1; { incrémentation gérée par le programmeur }
   until i>5;
   writeln;
   writeln('FIN. A la prochaine…');
end.
```

# Chap5: 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>**

$\Leftrightarrow$

**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)
```

# Chap5: 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

# 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;
    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;
   Jusque (i>b);
   Ecrire('Le produit est',P);
Fin.
```