Ministère de l'enseignement supérieur et de la recherche scientifique
Université Abderrahmane MIRA de Bejaia

Faculté de Technologie
Département de Technologie

Structure des ordinateurs et applications

Présentation

✓ Dr. DJAFRI Ghani
✓ ghani.djafri@univ-bejaia.dz

À propos du cours

- **Crédits**: 02
- **Coefficients**:02

Structures de contrôle répétitives

structures de contrôle répétitives

- Les **instructions itératives** (ou **structures de contrôle répétitives**) sont des instructions qui permettent de **répéter plusieurs fois un bloc de code** en fonction d'une condition. il existe trois types principaux de boucles :
  - Boucle Pour (For);
  - Boucle Tant-que (While);
  - Boucle Répéter (Repeat).

Boucle Pour (For)

Boucle Pour (For)

La structure de contrôle répétitive ***pour*** (mot-clé for en Pascal) permet d'exécuter un bloc d'instructions selon un compteur défini à l'avance.

La syntaxe de la boucle pour

Algorithme	**Pascal**

Pour <indice> ← <vi> à <vf>
faire
<instruction(s)> ;
finPour;
	for <indice> := <vi> to <vf> do
	begin
	<instruction(s)> ;
	End ;

<indice> : variable entière utilisée comme compteur de la boucle;
<vi> : valeur initiale de l'indice (début du comptage);
<vf> : valeur finale de l'indice (fin du comptage).

Caractéristiques principales

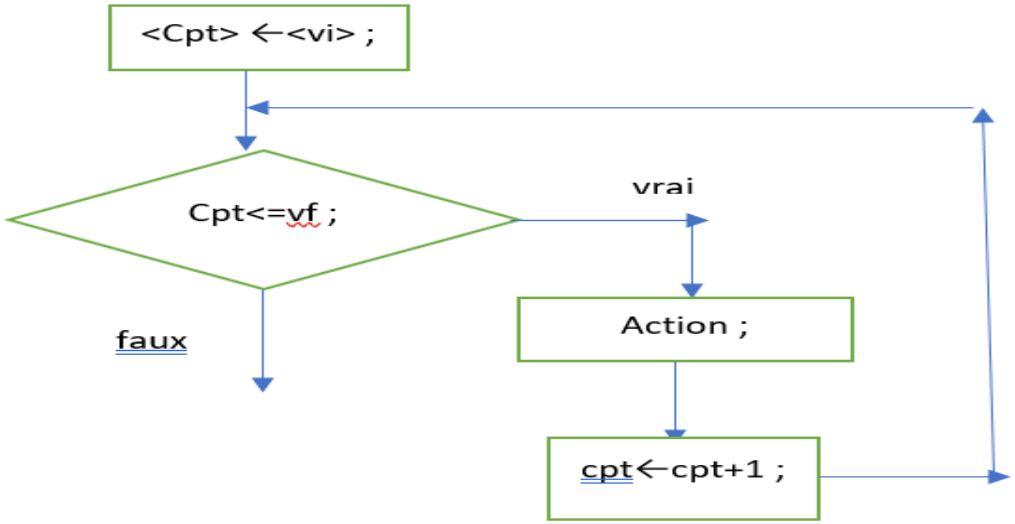
- L’indice est entier;
- La valeur de l’indice augmente d’un pas de 1 à chaque tour de boucle;
- La boucle est utilisée lorsque le nombre d’itérations est connu à l’avance;
- L’indice va d’une valeur initiale vers une valeur finale;
- La boucle ‘**pour’** contient **un** bloc d’instructions (les instructions à répéter). Si le bloc contient une seule instruction, le **begin** et **end** sont facultatifs.
- Il ne faut jamais mettre de **point-virgule** après le mot clé **do**. (erreur logique);

Déroulement de la boucle POUR

Dans la boucle **POUR**, on exécute le bloc <Acitons> (<vf> - <vi> + 1) fois. Ceci dans le cas où <vf> est supérieur ou égale à <vi>. Dans le cas contraire, le bloc d'actions ne sera jamais exécuté.

Le déroulement de la boucle POUR est exprimé comme suit :

1. La variable entière <cpt> (le compteur) prends la valeur initiale <vi> ;
2. On compare la valeur de <cpt> à celle de <vf> ; si <cpt> est supérieur à <vf> on sort de la boucle ;
3. Si <cpt> est inférieur ou égale à <vf> on exécute le bloc <Action(s)> ;
4. La boucle POUR incrémente automatiquement le compteur <cpt>, c'est-à-dire elle lui ajoute un (<cpt> ← <cpt> + 1);



Exercice –Écrire un programme en Pascal qui multiplie une variable x par 2, 5 fois et affiche le résultat final.

Boucle Tant-que (While)

La boucle tantque **(while)** en langage Pascal) utilise une expression logique ou booléenne comme condition d'accès à la boucle : si la condition est vérifiée (elle donne un résultat vrai : TRUE) donc on entre dans la boucle, sinon on la quitte.

La syntaxe de la boucle tantque

Algorithme	Pascal

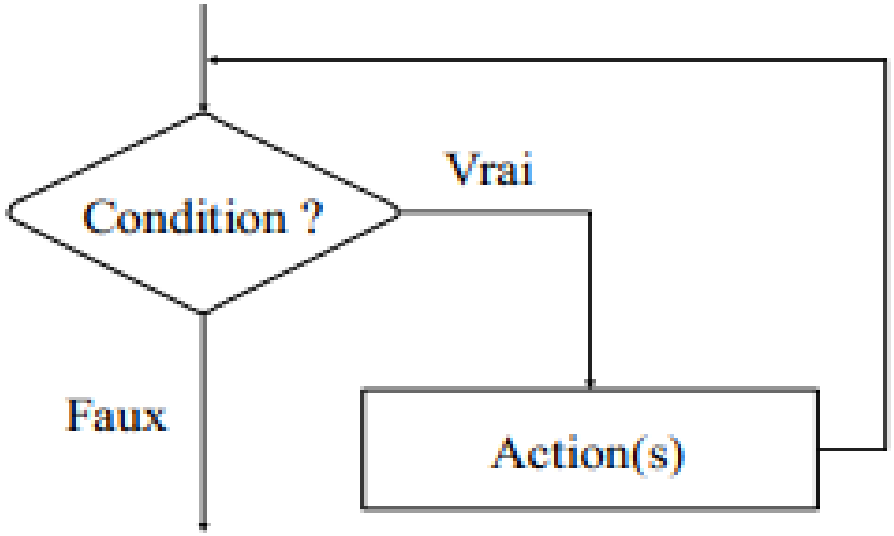
tant-que <condition> faire
<instruction(s)> ;
fin tant-que;
	while <condition> do
	begin
	<instruction(s)>;
	end;

<condition> : expression logique qui peut être vraie ou fausse.

Déroulement de la boucle Tanque

- On exécute le bloc d’instructions Tant que la condition est vraie. Une fois la condition est fausse, on arrête la boucle, et on continue l’exécution de l’instruction qui vient après fin Tant que (après **end**).
- Comme la boucle **for**, il faut jamais mettre de point-virgule après **do**.
- Il est possible de remplacer toute boucle "pour" par une boucle "tantque", cependant, l’inverse n'est pas toujours réalisable.

l'organigramme de la structure itérative Tant-que



Le déroulement de la boucle est comme suit :
On évalue la condition :

- si la condition est fausse on sort de la boucle ;
- Si la condition est vraie, on exécute le bloc <Action(s)> ;

Exercice –Écrire un programme qui multiplie une variable x par 2(c’est-à-dire x := x * 2) tant que x est inférieur à 20.

Boucle Répéter (Repeat)

Boucle répéter…jusqu'à: (repeat….until) en langage Pascal) utilise une expression logique ou booléenne comme condition de sortie de la boucle : si la condition est vérifiée (elle donneun résultat vrai : TRUE) on sort de la boucle, sinon on y accède (on répète l'exécution du bloc).

La syntaxe de la boucle répéter

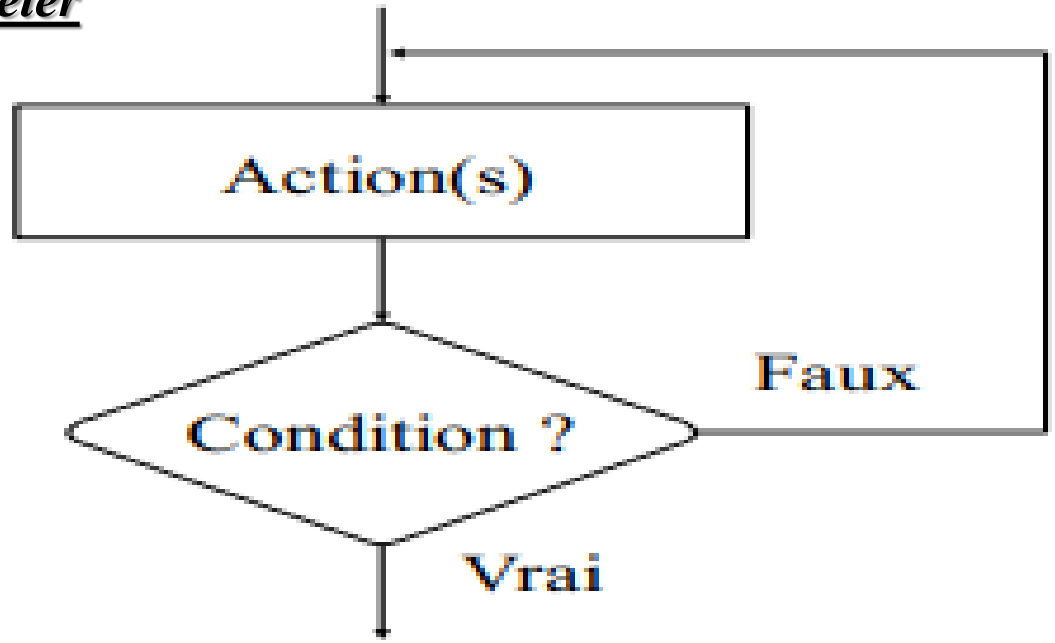
Algorithme	Pascal
répéter <instruction(s)> ; Jusqu'à <condition> ;	repeat <instruction(s)>; until <condition> ;

<condition> : expression logique qui peut être vraie ou fausse.

Déroulement de la boucle répéter

- On exécute le bloc d'instructions jusqu'à avoir la condition correcte. Une fois la condition est vérifiée, on arrête la boucle, et on continue l'exécution de l'instruction qui vient après **jusqu'à** (après **until**).
- Dans la boucle **repeat** on utilise pas **begin** et **end** pour délimiter le bloc d'instructions (le bloc est déjà délimité par **repeat** et **until**)

l'organigramme de la structure itérative répéter



On répète l'exécution du bloc <Action(s)> jusqu'à avoir la condition correcte.

Le déroulement est comment suit :

1. On exécute le bloc <Action(s)> ;
2. On évalue la condition : si la condition est vérifiée (elle est vraie) on sort de la boucle (on continue la suite de l'algorithme);
3. Si la condition n'est pas vérifiée (elle est fausse) on revient à 1.

Exercice –Écrire un programme en Pascal qui multiplie une variable x par 3 (c'est-à-dire x := x * 3) jusqu'à ce que x devienne supérieur ou égal à 20.

La différence entre la boucle répéter et la boucle tantque est :

- La condition de répéter et toujours l’inverse de la condition tantque : pour répéter c’est lacondition de sortie de la boucle, et pour tantque c’est la condition d’entrer.
- Le teste de la condition est à la fin de la boucle (la fin de l’itération) pour répéter. Par contre, il est au début de l'itération pour la boucle tantque. C’est-à-dire, dans tantque on teste la condition avant d’entrer à l’itération, et dans répéter on fait l’itération après on teste la condition.

Boucle Pour

```
Pour cpt ← vi à vf Faire
    <Bloc_Inst_Pour> ;
Fin-Pour;
```

Transformé

Boucle Tant-que

```
cpt ← vi;
Tantque (cpt <= vf) do
    <Bloc_Inst_Tantque> ;
    cpt ← cpt + 1 ;
Fin-Tantque;
```

Transformé

Boucle Répéter

```
cpt ← vi;
Répéter
    <Bloc_Inst_Répéter> ;
    cpt ← cpt + 1 ;
Jusqu'à (cpt > vf) ;
```

Telque :

ctp : compteur(varible entière)
vi : valeur initiale
vf : valeur finale

Modèle de conversion de la boucle Pour à Tant-que et à Répéter

Boucles	**Quand l'utiliser ?**	**Particularité**
Boucle Pour (For);	Quand le nombre d'itérations est connu	Variable auto-incrémentée
Boucle Tant-que (While);	Quand on ne connaît pas le nombre d’itérations	Condition testée **avant**
Boucle Répéter (Repeat).	Quand on veut exécuter au moins une fois	Condition testée **après**

Exemples

Exemples 1: Somme des n premiers nombres

Écrire un programme qui permet de calculer la somme des n premiers nombres entiers positifs.

Exemples 2 : Factorielle d'un nombre

Écrire un programme qui calcule la factorielle d'un nombre *n* (notée *n!*).

Exemples 3 : Nombres positifs entre deux bornes

Écrire un programme qui affiche les nombres positifs compris entre deux bornes *a* et *b* (avec $a < b$).

Exemples 4 : Somme des nombres impairs

Écrire un programme qui calcule la somme des nombres impairs de 1 à `n`.

```
program SommeNombres;
var
  n, i, somme: integer;
begin
  clrscr;
  write('Entrez un nombre n : ');
  readln(n);
  somme := 0;
  for i := 1 to n do
    somme := somme + i;
  writeln('La somme des ', n, ' premiers nombres est : ', somme);
  readln;
end.
```

```
program Factorielle;
var
  n, i: integer;
  fact: longint;
begin
  clrscr;
  write('Entrez un nombre n : ');
  readln(n);
  fact := 1;
  for i := 1 to n do
    fact := fact * i;
  writeln('La factorielle de ', n, ' est : ', fact);
  readln;
end.
```

```
program NombresPositifs;
var
  a, b, i: integer;
begin
  writeln('Entrer la borne a : ');
  readln(a);
  writeln('Entrer la borne b : ');
for i := a to b do
  begin
    if i > 0 then
      writeln(i);
  end;
end.
```

```
program SommeImpairs;
var
  n, i, somme: integer;
begin
write('Entrez un nombren : ');
  readln(n);
  somme := 0;
  for i := 1 to n do
    if i mod 2 <> 0 then
      somme := somme + i;
  writeln('La somme des nombres impairs de 1 à ', n, ' est : ', somme);
  readln;
end.
```