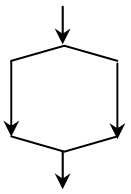
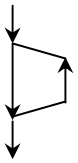
Chapitre 2 : Notions d'algorithme et de programme

Partie 2

8. Les structures de contrôle (structures ou primitives algorithmiques)

Séquencement linéaire	*Structures conditionnelles (ou de choix)*	*Structures répétitives (ou itératives) (Boucles)*
		

8.1. Les structures de contrôle conditionnelles (ou de choix)

En fonction d'une condition, le programme exécute des opérations différentes.

8.1.1. L'alternative (Le test - Le choix simple - La sélection - If ... Then ... Else ...)

Syntaxe en **algorithmique** :

```
Si Condition Alors
      Instruction1(s)
Sinon
      Instruction2(s)
FinSi
```

Syntaxe en **Pascal** :

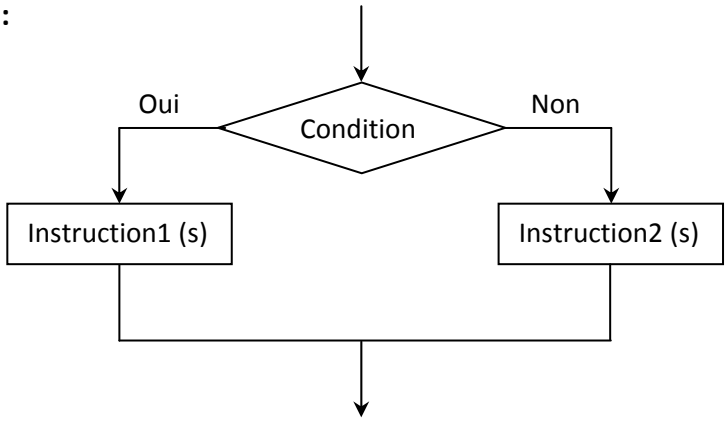
```
If Condition(s) Then Instruction1(s) Else Instruction2(s) ;
```

Ou

```
If Condition Then
      Instruction1(s)
Else
      Instruction2(s) ;
```

Si la condition est *VRAIE*, l'instruction **Then** est exécutée, *Sinon* l'instruction **Else** est exécutée si elle existe.

L'organigramme :



Remarques :

- Surtout pas de point virgule immédiatement avant le **Else** !!! Le **`If ... Then ... Else ...`** est une unique instruction composée, sans "**;**" au milieu.
- L'instruction "test" peut se limiter à **If ... Then** ; si le test est négatif alors aucune instruction ne sera effectuée et la suite du programme sera lue … (on peut omettre le **Else**).
- S'il y a plus d'une instruction à exécuter (instruction composée : *suite d'instructions*), il faut les encadrer par "**Begin**" et "**End**" :

```
If condition Then
      Begin
            Instructions ;
            ...
            Instructions ;
            Instructions
      End
Else
      Begin
            Instructions ;
            ...
            Instructions ;
            Instructions
      End;
```

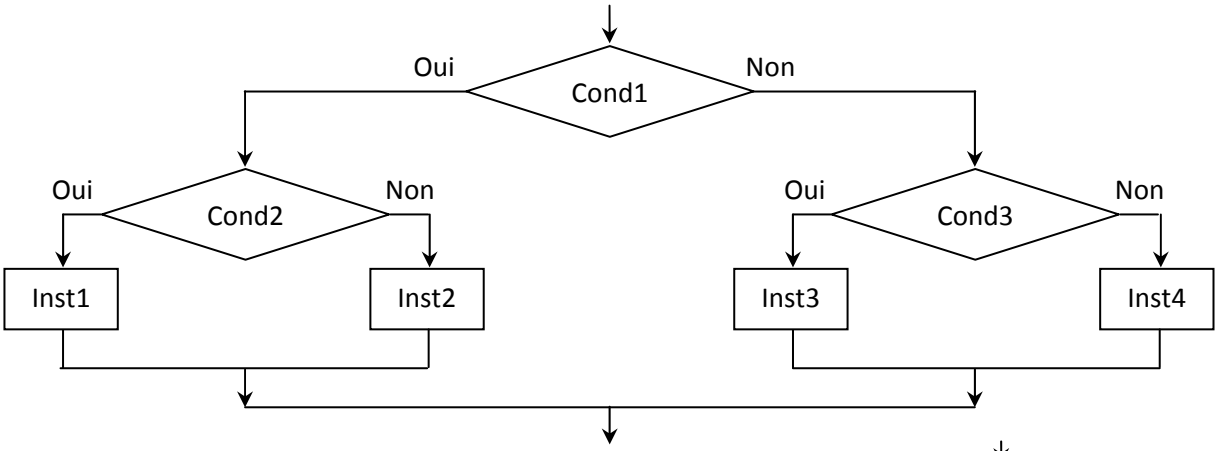
Exemple :

```
If (a>b) And (b>c) Then
Begin
      Writeln('premier :', a) ;
      Writeln('deuxième :', b) ;
      Writeln('troisième :', c)
End ;
```

On peut imbriquer plusieurs "**`If`**". Utiliser des indentations (marges décalées) pour une meilleure présentation.

Exemple :

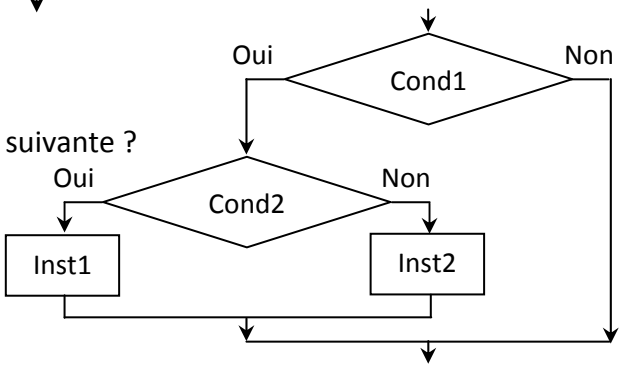
```
If Cond1 Then If Cond2 Then Inst1        {Cond1 et Cond2}
                  Else Inst2             {Cond1 et pas Cond2}
      Else If Cond3 Then Inst3           {pas Cond1 mais Cond3}
                  Else Inst4 ;           {ni Cond1 ni Cond3}
```



Remarque :

Quel est l'effet de la partie de programme suivante ?

```
If Condition_1 Then
      If Condition_2 Then
            Instruction_1
      Else Instruction_2 ;
```

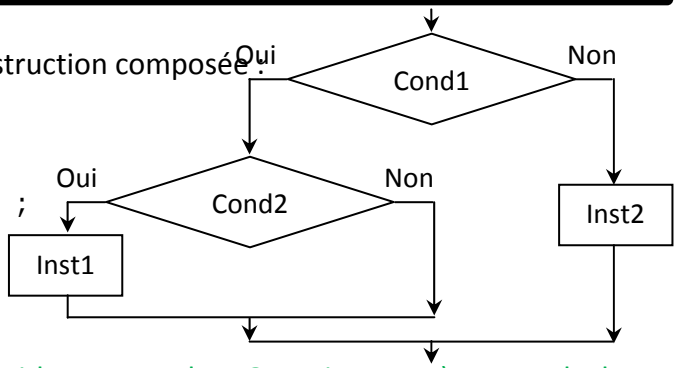


Cette situation est ambiguë. Par convention, la partie **Else** correspond à l'instruction **If** *la plus proche*, ici **If** condition_2 **Then …**

> *Un **Else** se rapporte toujours au dernier **Then** rencontré auquel un **Else** n'a pas encore été attribué.*

Le contraire s'obtient à l'aide d'une instruction composée :

```
If Condition_1 Then
Begin
      If Condition_2 Then
            Instruction_1 ;
End
Else Instruction_2 ;
```



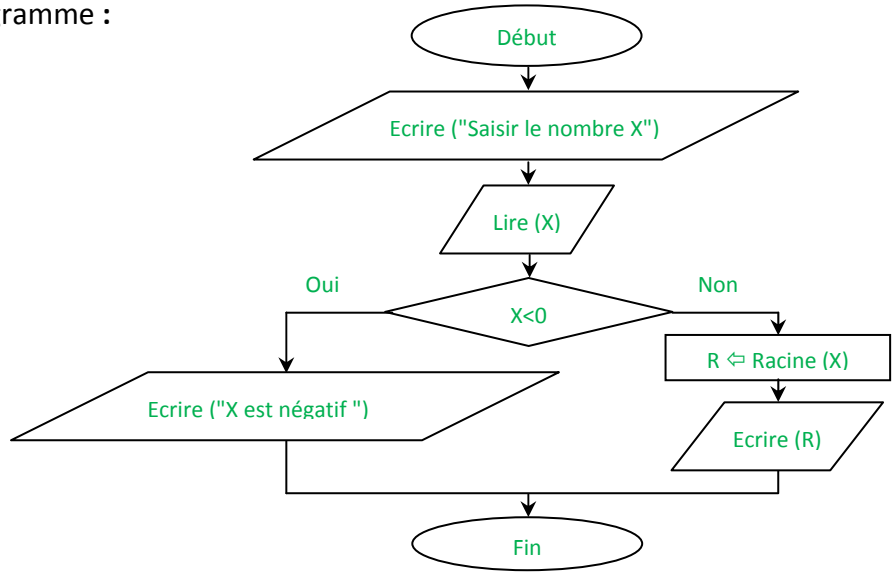
Exemple : Ecrire un programme (Algorithme, Pascal et Organigramme) pour calculer et afficher la racine carrée.

En **algorithmique** :	En **Pascal** :

Algorithme Racine_Carree
Variable X, R : Réel
Début
  Ecrire ("Saisir le nombre X")
  Lire (X)
  Si X<0 Alors
  Ecrire ("X est négatif")
  Sinon
  R ⇦ Racine (X)
  Ecrire (R)
  FinSi
Fin
	Program Racine_Carree ;
	Var X, R : Real ;
	Begin
	  WriteLn ('Saisir le nombre X') ;
	  ReadLn (X) ;
	  If X<0 Then
	  WriteLn ('X est négatif')
	  Else
	  Begin
	  R := Sqrt (X) ;
	  WriteLn (R) ;
	  End ;
	End.

Pour les nombres complexes et si X<0, on affiche : `Write ('i', Sqrt (-X)) ;`

L'organigramme **:**



Exemple : (Pour la maison)

Ecrire un programme devant donner l'état de l'eau selon sa température, doit pouvoir choisir entre 3 réponses possibles : Solide, Liquide ou Gazeuse.

En **algorithmique** :	En **Pascal** :

`Algorithme Temperature_Eau`
`Variable Temps : Entier`
`Début`
`  Ecrire ("Entrer la température de l'eau")`
`  Lire (Temps)`
`  Si Temps ≤ 0 Alors`
`  Ecrire ("C'est de la glace")`
`  Sinon`
`  Si Temps < 100 Alors`
`  Ecrire ("C'est du liquide")`
`  Sinon`
`  Ecrire ("C'est de la vapeur")`
`  FinSi`
`  FinSi`
`Fin`
	`Program Temperature_Eau ;`
	`Var Temps : Integer ;`
	`Begin`
	`  WriteLn ('Entrer la température de l''eau') ;`
	`  ReadLn (Temps) ;`
	`  If Temps <= 0 Then`
	`  WriteLn ('C''est de la glace')`
	`  Else`
	`  If Temps < 100 Then`
	`  WriteLn ('C''est du liquide')`
	`  Else`
	`  WriteLn ('C''est de la vapeur') ;`
	`End.`

8.1.2. La structure de choix, Cas - Parmi (Sélection Multiple : le Case...Of)

Une donnée est comparée successivement à des valeurs constantes :

*Souvent, avec les types énumérés, on veut faire un traitement qui dépend de la valeur. Pour cela, on peut bien sûr utiliser des instructions **If**, mais cela a tendance à rendre le programme plus lourd que nécessaire. Lorsqu'on veut faire un traitement par cas, il y a une instruction beaucoup plus pratique, l'instruction **Case**.*

*On parle de **type énuméré** pour désigner un type dans lequel on peut donner tous les éléments les uns à la suite des autres. Pour l'instant, on en a vu deux, le type **Integer** et le type **Char**, mais on en verra d'autres dans la suite du cours*

Syntaxe en **algorithmique** :

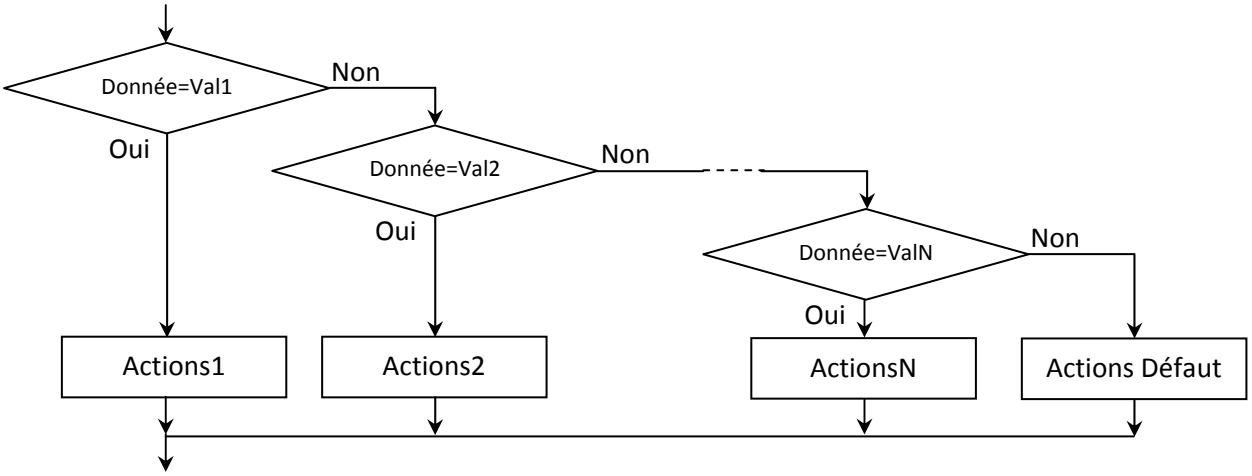
```
Cas où Donnée Vaut
   Valeur1 : Actions1
   Valeur2 : Actions2
   ...
   ValeurN : ActionsN
   Autre : Actions défaut
FinCas
```

Syntaxe en **Pascal** :

```
Case <Expression> Of
   Valeur 1 : Instruction(s) 1 ;
   Valeur 2 : Instruction(s) 2 ;
   ...
   Valeur n : Instruction(s) n ;
   Else Instruction(s) par défaut ;
End ;
```

*En **Turbo Pascal**, on accepte une **Valeur** particulière qui est **Else** (et doit être placée en dernier), pour prévoir le cas où Expression n'appartient à aucun des cas cités au dessus. En **Ms-Pascal** on utilise de même **Otherwise**.*

L'organigramme **:**



Remarques :

- La partie « Action Défaut » peut ne pas exister.
- Plusieurs valeurs différentes peuvent être regroupées sur une même ligne si les actions correspondantes sont identiques.
- S'il y a plusieurs instructions, on doit les encadrer par « **Begin** » et « **End ;** ».

*Opportunité du **Case** :*
- *Si structures **If... Then... Else...** imbriquées.*
- *Si égalités testant la valeur d'une expression.*
- *Très utile pour la **lisibilité du programme** !!!*

***Case** peut être remplacée par **If imbriquée**.*
L'inverse n'est possible que si les tests concernent une seule variable, et que les tests concernant cette variable est soit une égalité soit un intervalle.

Exemple :

Affichage de la nature d'un caractère.

En **algorithmique** :

```
Algorithme Nature_Caract
Variable C : Caractère
Début
   Ecrire ("Tapez un caractère")
   Lire (C)
   Cas Où C Vaut
      "A".."Z" : Ecrire ("Lettre majuscule")
      "a".."z" : Ecrire ("Lettre minuscule")
      "0".."9" : Ecrire ("Chiffre")
      Autre : Ecrire ("Symbole")
   FinCas
Fin
```

En **Pascal** :

```
Program Nature_Caract ;
Var C : Char ;
Begin
   WriteLn ('Tapez un caractère') ;
   ReadLn (C) ;
   Case C Of
      'A'..'Z' : WriteLn ('Lettre majuscule') ;
      'a'..'z' : WriteLn ('Lettre minuscule') ;
      '0'..'9' : WriteLn ('Chiffre')
      Else WriteLn ('Symbole') ;
   End ;
End.
```

*Attention : Là, le point-virgule avant le **Else** est **FACULTATIF**. Mais pour plus de sécurité afin de ne pas faire d'erreur avec le bloc **If**, choisissez systématiquement d'omettre le point-virgule avant un **Else**.*

Exemple 2 :

```
Program case_age;
Var age:Integer;
Begin
      Write('Entrez votre âge : ') ;
      Readln(age) ;
      Case age Of
            18 : Writeln('18 ans');
            0..17 : Writeln('Petits enfants...') ;
            60..99 : Writeln('Les vieux')
            Else Writeln('Vous êtes d''un autre âge...') ;
      End ;
End.
```

Note : On peut effectuer un test de plusieurs valeurs en une seule ligne par séparation avec une virgule si on souhaite un même traitement pour plusieurs valeurs différentes. Ainsi la ligne :

```
0..17 : Writeln('Petits enfants …') ;
```

Peut devenir :

```
0..10, 11..17 : Writeln('Petits enfants …') ;
```

Ou encore :

```
0..9, 10, 11..17 : Writeln('Petits enfants …') ;
```

Ou même :

```
0..17, 5..10 : Writeln('Petits enfants …') ;
```

{Cette dernière ligne est stupide mais correcte !}

8.2. Les structures de contrôle répétitives (ou itératives) (Boucles)

Souvent un algorithme pose une question à laquelle l'utilisateur doit répondre par « OUI » ou par « NON » et propose un modèle de réponse, par exemple : "Voulez-vous conserver ces résultats ? O/N". Lorsque la réponse est incorrecte (elle n'est ni O ni N), elle doit être rejetée, et la question doit être posée à nouveau.

On pourrait essayer avec un « SI » comme suit :

```
Algorithme Question
Variable Rep : Caractère
Début
   Ecrire ("Voulez-vous un livre ? O/N")        .     .     .     (1)
   Lire (Rep)     .     .     .     .     .     .     .     .     (2)
   Si (Rep ≠"O") et (Rep ≠"N") Alors.     .     .     .     .     (3)
      Ecrire ("Vous devez répondre par O ou N")  .     .     .     (4)
      Lire (Rep)   .     .     .     .     .     .     .     .     (5)
   FinSi    .     .     .     .     .     .     .     .     .     (6)
Fin
```

Ainsi, lorsque la réponse n'est ni O ni N, un message d'erreur apparait.

Quoi faire ?

*Reproduire le même « **SI** », après le second **Lire (Rep)**, permettrait de régler une deuxième erreur, mais pas une troisième … L'algorithme ne devrait passer à la suite que lorsque la variable **Rep** contient, soit la lettre « **O** », soit la lettre « **N** ».*

La bonne solution consiste à utiliser une structure itérative (boucle).

Une itération consiste à exécuter un bloc d'instructions un certain nombre de fois. Dans la plupart des cas, on ne connait pas le nombre de répétitions.

Nous allons examiner différentes manières de faire répéter une séquence d'instructions, appelée « corp de la boucle ».

8.2.1. La boucle (structure) "While ... Do" (TantQue ... Faire)

Une action ou un groupe d'actions est exécuté répétitivement tout le temps où une condition est **VRAI**.

Syntaxe en **algorithmique** :

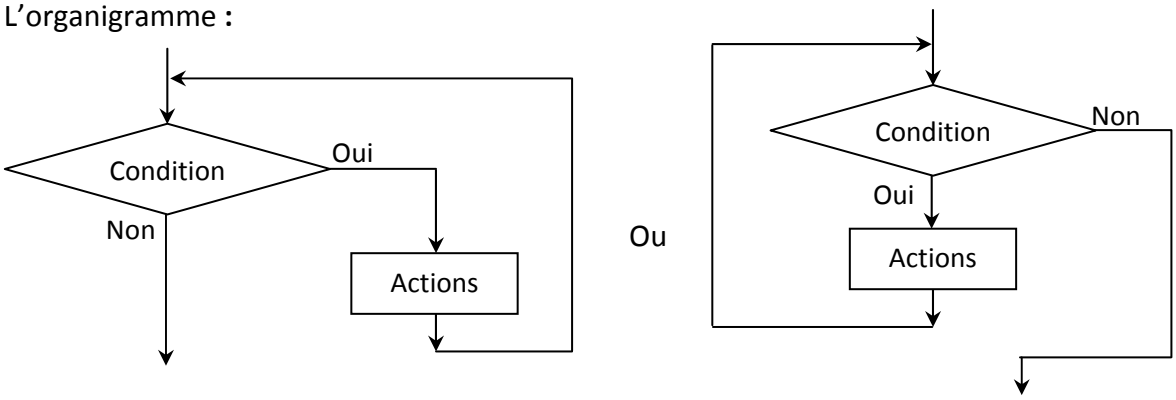
```
TantQue Condition Faire
      Actions
FinTantQue
```

Séquence (bloc d'instructions)

Syntaxe en **Pascal** :

```
While <Condition> Do
      <Instruction> ;
```

L'organigramme **:**



Lorsque le corps de la boucle contient plus d'une instruction (comme pour l'instruction **IF**) il faut l'encadrer par les termes « **Begin** » et « **End ;** ».

Illustration avec notre exemple :

Une première approximation de la solution consiste à écrire :

```
Algorithme Question2
Variable Rep : Caractère
Début
   Ecrire ("Voulez-vous un livre ? O/N")        .     .     .     (1)
   TantQue (Rep ≠"O") et (Rep ≠"N") Faire .     .     .     .     (2)
      Lire (Rep)   .     .     .     .     .     ..    .     .     (3)
   FinTantQue     .     .     .     .     .     .     .     .     (4)
Fin
```

C'est le squelette de l'algorithme correct (manque les muscles les organes pour qu'il fonctionne correctement).
Son principal défaut est de provoquer une erreur à chaque exécution :
*La variable **Rep** n'a pas été affectée avant l'entrée dans la boucle.*
On teste donc une variable qui n'a pas de valeur, ce qui provoque une erreur et l'arrêt immédiat de l'exécution.
*Pour éviter ceci, Il faut que la variable **Rep** ait déjà été affectée avant qu'on en arrive au premier tour de boucle*
*⇨ **2** possibilités :*
***1ère Possibilité** : On peut faire une première lecture da la variable **Rep** avant la boucle. Dans ce cas, celle-ci ne servira qu'en cas de mauvaise saisie lors de cette première lecture. On ajoute dans l'algorithme précédent entre les lignes **(1)** et **(2)** :*

```
Lire (Rep)
```

Erreur ⇨ La variable **Rep** n'a pas été affectée avant l'entrée dans la boucle.

2ème Possibilité : (fréquemment employée) consiste à affecter arbitrairement la variable avant la boucle. Provoquer l'entrée obligatoire dans la boucle.

L'affectation doit donc faire en sorte que la condition soit mise à **VRAI** pour déclencher le 1er tour de la boucle.

Dans notre exemple, on peut donc affecter **Rep** avec n'importe quelle valeur à part « **O** » et « **N** ».

On ajoute par exemple entre les lignes **(1)** et **(2)** l'instruction :

```
Rep ⇦ "X"
```

Remarque : Il existe une différence importante dans les structures logiques des 2 possibilités.
*Pour la 1ère possibilité, la boucle sera exécutée uniquement dans l'hypothèse d'une mauvaise saisie initiale. Si l'utilisateur saisit une valeur correcte à la 1ère demande de **Rep**, l'algorithme passera sur la boucle sans entrer dedans.*
Avec la 2ème solution (possibilité), l'entrée de la boucle est forcée, et l'exécution de celle-ci au moins une fois est rendue obligatoire à chaque exécution du programme.

On peut ajouter des affichages de libellés qui font encore un peu défaut. L'algorithme devient :

```
Algorithme Question2
Variable Rep : Caractère
Début
   Rep ⇦ "X"
   Ecrire ("Voulez-vous un livre ? O/N")
   TantQue (Rep ≠"O") et (Rep ≠"N") Faire
      Ecrire ("Vous devez répondre par O ou N")
      Lire (Rep)
   FinTantQue
   Ecrire ("Saisie acceptée")
Fin
```

Traduction en Pascal** : On doit encadrer les 2 instructions de **TantQue** par **Begin** et **End ;

*Si on ajoute dans la boucle (par exemple avant **FinTantQue**) l'instruction :*

```
Rep ⇦ "X"
```

⇨ Boucle INFINIE

Remarque (Conclusion) :

Si la 1ère évaluation de la condition fournit la valeur **FAUX**, le corps de la boucle n'est exécuté (le programme ne rentre jamais dans la boucle). Et si la séquence (les actions) ne change pas la valeur de la condition et si celle-ci a la valeur **VRAI**, la séquence sera répétée sans que l'on passe jamais à la suite : boucle Infinie, il est donc indispensable de prendre les 2 précautions suivantes :

- Initialiser les variables qui interviennent dans la condition avant d'aborder la boucle.
- S'assurer que la séquence est capable de faire évoluer la valeur de la condition vers la valeur **FAUX**, ce qui permettra l'arrêt de la boucle.

8.2.2. La boucle "Repeat ... Until" (Répéter... Jusqu'à)

Dans une boucle « **TantQue** », la condition est évaluée avant d'exécuter la 1ère instruction du bloc contenu dans la boucle (séquence).

On peut souhaiter dans certains cas, évaluer la condition après l'exécution de la dernière instruction de ce bloc. Un tel algorithme correspond à une boucle « **Répéter... Jusqu'à** ».

Syntaxe en **algorithmique** :

```
Répéter
      Actions
Jusqu'à Condition
```

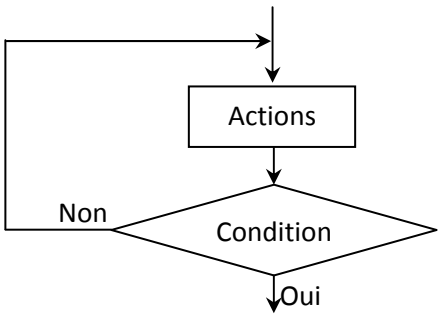
Syntaxe en **Pascal** :

```
Repeat
      <Instruction(s)> ;
Until <Condition> ;
```

Ou

```
Repeat < Instruction1; ... ; InstructionN > ; Until < Condition > ;
```

L'organigramme **:**



Remarques :

1. Ici, même si le corps de la boucle contient plus d'une instruction, il n'est pas nécessaire de l'encadrer par « **Begin** » et « **End ;** ».
2. Le bloc d'instruction sera répété jusqu'à ce que la condition soit vérifiée.
3. Au contraire de la boucle « **TantQue** », l'utilisation de la boucle « **Répéter... Jusqu'à** » garantit que le bloc sera exécuté **au moins une fois** puisque le test a lieu après son exécution.

4. La signification de la condition n'est plus la même :
   - Avec « **TantQue** » ⇨ Condition pour continuer la répétition (test d'entrée).
   - Avec « **Répéter... Jusqu'à** » ⇨ Condition d'arrêt (test de sortie).

Note :
*La commande **Inc** permet d'incrémenter une variable d'une certaine valeur. La commande **Dec** permet au contraire de décrémenter une variable d'une certaine valeur. Ces commandes permettent d'éviter la syntaxe :*

Variable := Variable + 1 et **Variable := Variable - 1.**

Syntaxe :

```
Inc (Variable, Nombre) ;
Dec (Variable, Nombre) ;
```

Remarque :
*On utilise généralement les instructions **While** ou **Repeat** lorsque l'on ne connaît pas, à l'avance, le nombre d'itérations.*

Exemple 1 : Refaire l'exemple précédent en utilisant « **Répéter... Jusqu'à** ».

*Avec **Répéter ... Jusqu'à** :*

```
Algorithme Question3
Variable Rep : Caractère
Début
   Ecrire ("Voulez-vous un livre ? O/N")
   Répéter
      Ecrire ("Vous devez répondre par O ou N")
      Lire (Rep)
   Jusqu'à Rep ="O" ou Rep ="N"
   Ecrire ("Saisie acceptée")
Fin
```

Traduction en Pascal

Exemple 2 : Répéter la multiplication de 2 nombre.

```
Algorithme Multiplication
Variable    a, b, P : Réel
            c : Caractère
Début
   Répéter
      Ecrire ("Saisir les nombres a et b")
      Lire (a,b)
      P ⇦ a*b
      Ecrire ("Le produit est",P)
      Ecrire ("Encore un calcul ? NON touche N ; OUI autre touche")
      Lire (c)
   Jusqu'à c = "N"
Fin
```

*Avec **TantQue** :*

```
c ⇦ "k"
TantQue c≠"N" Faire
...
```

8.2.3. La boucle "For" (Pour)

Il est pratique d'utiliser la boucle « **Pour** » si le nombre de passage dans la boucle est connu au préalable, une variable particulière **i** appelée **Variable d'itération ou de contrôle** (indice) sert à compter le nombre de répétitions du bloc d'instructions.

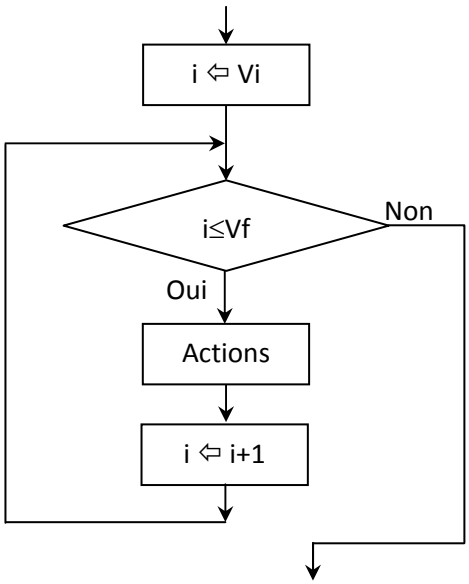
Syntaxe en **algorithmique** :

```
Pour i = Vi à Vf Faire
      Actions
FinPour
```

Syntaxe en **Pascal** :

```
For i := Vi To Vf Do
      Instruction(s) ;
```

L'organigramme **:**



Vi : Valeur initiale

Vf : Valeur finale

i : indice (variable d'itération, variable de contrôle, compteur) : Entier, on peut utiliser un autre caractère.

Remarques :

1. Le nombre de passage dépond de **Vi** et **Vf** (le nombre de passage = **Vf - Vi + 1**).
2. Si **Vi** > **Vf**, le bloc d'instructions n'est jamais exécuté.
3. S'il y a plus d'une instruction à exécuter, il faut les encadrer par **Begin** et **End :**

```
For i := Vi To Vf Do
Begin
      Instructions ;
End ;
```

4. La boucle « **Pour** » ne convient pas à l'écriture d'un algorithme de lecture de réponse **OUI** ou **NON**.
5. La variable **i** peut être non seulement de type entier, mais aussi de tout type sur lequel est défini un ordre, par exemple : le type caractère dans lequel le suivant de "**A**" est "**B**", etc.
6. La variable d'itération **i** change automatiquement de valeur à chaque itération (s'incrémente), la valeur de l'incrément par défaut est égale à 1. Certains langages autorisent la définition d'un pas d'itération différent de 1. Il suffit de le préciser dans l'algorithme, comme le montre l'exemple suivant :

```
Pour i = Vi à Vf Pas de P Faire
      Actions
FinPour
```

7. La valeur du pas **P** peut être négative ; dans ce cas **Vi>Vf**.

Ou
```
Pour i = Vi en descendant jusqu'à Vf Faire
            Actions
FinPour
```

8. E Pascal, lorsque la *valeur initiale* est supérieure à la *valeur finale*, le **To** doit être remplacé par **Downto** comme suit (syntaxe) **:**

```
For i:= Vi Downto Vf Do
      Instruction(s);
```

Exemple 1 : Affichage d'une ligne d'étoiles (80 étoiles).

En **algorithmique** :	En **Pascal** :

Algorithme Etoiles
Variable i : Entier
Début
  Pour i=1 à 80 Faire
  Ecrire ("*")
  FinPour
Fin
	Program Etoiles ;
	Var i : Integer ;
	Begin
	  For i :=1 to 80 Do
	  Write ('*') ;
	End.

Exemple 2 :

```
Program Compteur2 ;
Var n : Integer ;
Begin
      For n:=1 To 10 Do
            Writeln(n,' a pour triple ',3*n)
End.
```

Exemple 3 :

```
Program Compteur3 ;
Var n : Integer ;
Begin
      For n:=10 Downto 1 Do
            Writeln(n,' a pour triple ',3*n)
End.
```

Étape d'exécution :

1) *Evaluation et vérification des bornes (les bornes sont évaluées avant l'exécution du* ***For****).*
2) *Pour chaque valeur consécutive :*
   - *Affectation à la variable de contrôle.*
   - *Exécution de l'instruction à subir.*
   - *La borne supérieure est incluse*

Remarque : cette structure algorithmique peut être remplacée par une structure « **TantQue** » ou « **Répéter** ». ⇨

*(Comparer les organigrammes des boucles « **Pour** » et « **TantQue** »)*

Exemple :

```
Algorithme Etoiles_TantQue
Variable i : Entier
Début
   i ⇦ 1
   TantQue i≤80 Faire
      Ecrire ("*")
      i ⇦ i+1
   FinTantQue
Fin
```

Ou ***TantQue i ≠ 81 Faire***
Ou ***TantQue i < 81 Faire***

Initialisation

Incrémentation

*Gagnées avec « **Pour** »*

Traduction en Pascal

```
Algorithme Etoiles_Répéter
Variable i : Entier
Début
   i ⇦ 1
   Répéter
      Ecrire ("*")
      i ⇦ i+1
   Jusqu'à i>80 Faire
Fin
```

Ou ***Jusqu'à i=81 Faire***
Ou ***Jusqu'à i≥81 Faire***

Remarque :

- On pourrait bien programmer toutes les situations de boucles uniquement avec un « **Tantque** » (ou avec « **Répéter** »). Le seul intérêt du « **Pour** » est d'épargner un peu de fatigue au programmeur, en lui évitant de gérer lui-même la progression de la variable i qui lui sert de compteur (incrémentation automatique). Autrement dit, la structure « **Pour** » est un cas particulier de « **TantQue** » (ou « **Répéter** »).

- Les structures « **TantQue** » (ou « **Répéter** ») sont employées là où l'on ne connaît pas d'avance le nombre de répétitions, comme par exemple :
  - Le contrôle d'une saisie (O ou N).
  - La gestion des tours d'un jeu (Tant que la partie n'est pas finie on recommence).
  - La lecture des enregistrements d'un fichier de taille inconnue (bases de données).
- Les structures « **Pour** » sont employées là où on connaît le nombre de répétitions d'avance.

Les boucles dans les boucles ⇨ OUI (boucles imbriquées)
Mais pas de boucles croisées.

Le nombre de répétitions = Vf-Vi+1

Remarque :
*Les boucles "**For**" sont utiles mais il faut prendre quelques précautions quand on les utilise :*

1. *Il ne faut jamais changer la valeur de l'indice d'une boucle dans l'instruction. Autrement, si quand même besoin, on utilisera une autre variable qui prend la valeur de l'indice ; ou on utilise les autres instructions itératives.*
2. *La valeur de l'indice est limitée à l'instruction de la boucle (elle est perdue dès que l'on sort de la boucle). C'est une variable muette (sa valeur en sortie de boucle est indéterminée).*
3. *"Valeur Initiale" et "Valeur Finale" peuvent être des expressions, donc si la valeur de ces expressions est modifiée dans la boucle, le nombre d'itérations **ne change pas !***

*Ne pas suivre la première règle peut donner un programme valide (qui marche bien), mais il y a des risques d'erreurs et il est possible de rendre le programme incompréhensible. Par exemple, que fait la boucle **For** suivante :*

```
Program mauvaiseboucle;
(* Exemple de programme où on change le valeur de l'indice à l'intérieur d'une boucle For *)
Var lettre1 : Char;
Begin
      For lettre1 := 'A' To 'E' Do
            Begin
                  Writeln(lettre1); lettre1:= 'A' end
```

Exemple : Examinons l'algorithme suivant :

```
Algorithme Actuce
Variable Truc : Entier
Début
  Pour Truc=1 à 20 Faire     .     .     .     .     .     .     (1)
    Truc ⇦ Truc*3     .     .     .     .     .     .     .     (2)
    Ecrire ("passage numéro", Truc)     ..    .     .     .     (3)
  FinPour   .     .     .     .     .     .     .     .     .     (4)
Fin
```

- La ligne (1) augmente la valeur de **Truc** de **1** à chaque passage.
- La ligne (2) triple la valeur de **Truc** à chaque passage.

⇨ Contradiction ⇨ Plantage ⇨ éviter ce genre d'erreur (éviter de manipuler au sein d'une boucle « **Pour** » la variable qui sert de compteur à cette boucle).

Exemple : Calculer la somme des N premiers nombres entiers (avec les différentes boucles)

1ère méthode

```
Algorithme Suite
Variables N, i, Som : Entier
Debut
Ecrire ("Entrez un nombre : ")
Lire (N)
Som ← 0
Pour i = 1 à N Faire
  Som ← Som + i
FinPour
Ecrire ("La somme est : ", Som)
Fin
```

Som ← 1
Pour i = 2 à N **Faire**

2ème méthode sans boucles (c'est une suite arithmétique) :

```
Algorithme Suite2
Variables N, Som : Entier
Debut
Ecrire ("Entrez un nombre : ")
Lire (N)
Som ← (1+N)*N/2
Ecrire ("La somme est : ", Som)
Fin
```

3ème méthode

```
Algorithme Suite3
Variables N, i, Som : Entier
Debut
Ecrire ("Entrez un nombre : ")
Lire (N)
Som ← 0
i ← 1
TantQue i ≤ N Faire
  Som ← Som + i
  i ← i + 1
FinTantQue
Ecrire ("La somme est : ", Som)
Fin
```

Ou: **TantQue** i ≠ N + *1* **Faire**
Ou: **TantQue** i < N + *1* **Faire**

4ème méthode

```
Algorithme Suite4
Variables N, i, Som : Entier
Debut
Ecrire ("Entrez un nombre : ")
Lire (N)
Som ← 0
i ← 1
Répéter
  Som ← Som + i
  i ← i + 1
Jusqu'à i > N
Ecrire ("La somme est : ", Som)
Fin
```

Ou bien : ***Jusqu'à*** *i = N + 1*
Ou aussi : ***Jusqu'à*** *i ≥ N + 1*

- *Traduction en Pascal*
- *En déduire la factorielle F*

Avec :

F : Real
Write (F :0 :0)