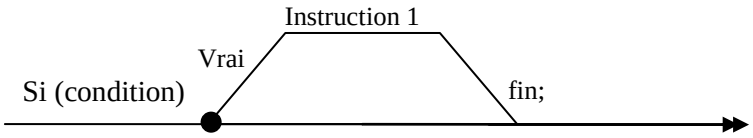
Chapitre 1 : Les boucles

Rappel sur les tests

Dans les tests conditionnels qu'on a vus avant, on a deux formes possibles qui permettent de traiter les différents problèmes, la première est la plus simple :

```
si condition alors
      Instructions
fin;
```



Ceci appelle quelques explications.

Une **condition** est une **expression** dont la valeur est **booléenne** (VRAI ou FAUX). Cela peut donc être (il n'y a que deux possibilités) :

- Un traitement de la suite d'instructions 1, dans le cas où la condition est Vrai.
- Une poursuite de l'enchainement des instructions après le test, dans le cas où la condition est Faux.

la deuxième forme comporte un traitement explicite des deux différent cas possibles (Vrai et faux).

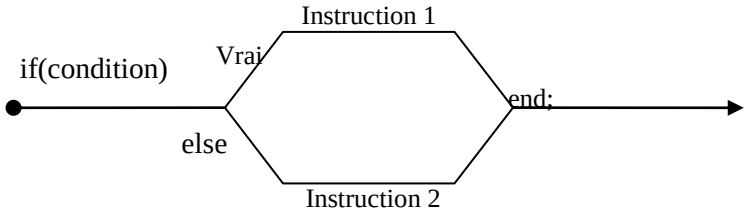
```
if booléen then
    Instructions 1
else
    Instructions 2
end;
```

dans ce cas un traitement est prévu dans chacune des deux possibilités.

Exemple:

Ecrire un programme qui demande un nombre à l'utilisateur, et l'informe ensuite si ce nombre est positif ou négatif (on considère le zéro comme positif).

```
program test ;
Var n : integer ;
Begin
Write( "Entrez un nombre : " ) ;
Read (n);
 If n < 0 then
      Write ("négatif ") ;
 else
      Write ("positif") ;
 end;
end.
```

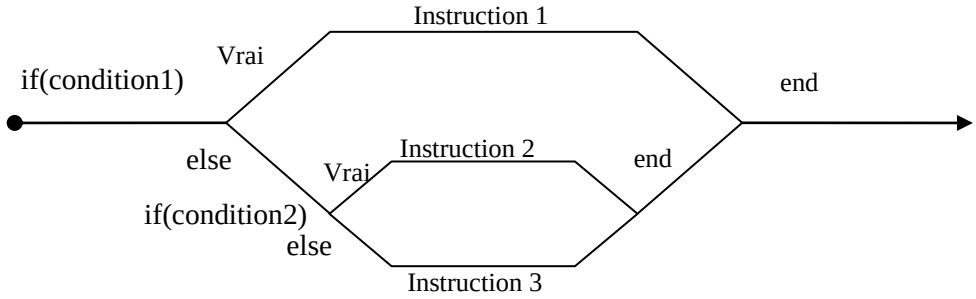


Tests imbriqués

Exemple :

Ecrire un programme qui donne l'état de l'eau selon sa température (doit pouvoir choisir entre trois réponses possibles; solide, liquide ou vapeur).

```
program Temperature;
Var Temp : integer ;
Begin
 Write ("Entrez la température de l'eau :" ) ;
 Read (Temp);
 if Temp =< 0 then
      Write ("C'est de la glace" );
 else
      if Temp < 100 then
            Write ("C'est du liquide" );
      else
            Write ("C'est de la vapeur");
      end ;
 end ;
end.
```



Les boucles

Prenons le cas d'une saisie au clavier (une lecture), où par exemple, le programme demande à l'utilisateur de saisir une valeur entre 10 et 20. Mais tôt ou tard, l'utilisateur risque de taper autre chose que la réponse attendue. Dès lors, le programme peut planter soit par une erreur d'exécution (parce que le type de réponse ne correspond pas au type de la variable attendu) soit par une erreur fonctionnelle (il se déroule normalement jusqu'au bout, mais en produisant des résultats inattendues).

Alors, dans tout programme un tant soit peu sérieux, on met en place ce qu'on appelle un **contrôle de saisie**, afin de vérifier que les données entrées au clavier correspondent bien à celles attendues par l'algorithme.

On pourrait essayer avec un SI. Voyons voir ce que ça donne :

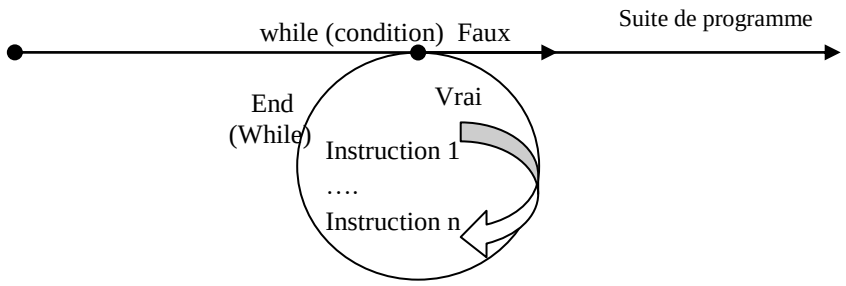
```
program Saisie;
Var Rep : Char;
Begin
 write ("Donner votre une valeur entre 10 et 20 " ) ;
 read (Rep);
 if Rep < 10 OR Rep > 20 then
   write ("Saisie erronnée. Recommencez");
   read (Rep);
 end;
write('Réponse correcte');
end
```

C'est impeccable. Du moins tant que l'utilisateur a le bon goût de ne se tromper qu'une seule fois, et d'entrer une valeur correcte à la deuxième demande. Si l'on veut également renforcer en cas de deuxième erreur, il faudrait rajouter un SI. Et ainsi de suite, on peut rajouter des centaines de SI, et écrire un algorithme très lourd, on n'en sortira pas, il y aura toujours moyen de bloquer le programme.

La solution consistant à aligner des SI… est donc une impasse. La seule issue est donc de lancer une **structure de boucle**, qui se présente ainsi :

```
while booléen do
…
Instructions
…
End ;
```

Le principe est simple : le programme arrive sur la ligne du **While**. Il examine alors la valeur du booléen (qui, je le rappelle, peut être une variable booléenne ou, plus fréquemment, une condition). Si cette valeur est VRAI, le programme exécute les instructions qui suivent, jusqu'à ce qu'il rencontre la ligne EndWhile. Il retourne ensuite sur la ligne du While, procède au même examen, et ainsi de suite. Le manège enchanté ne s'arrête que lorsque le booléen prend la valeur FAUX.



La solution du problème précédent est alors :

```
program Saisie;
```

```
Var Rep : Char;
Begin
 write ("Donner votre une valeur entre 10 et 20 " ) ;
 read (Rep);
 while (Rep < 10 OR Rep > 20) do
   write ("Saisie erronnée. Recommencez");
   read (Rep);
 end;
write('Réponse correcte');
end
```

Exemple 1:

Ecrire un programme qui demande à l'utilisateur un nombre supérieur à 6 jusqu'à ce que la réponse convienne.

Solution :

```
program exemple ;
Variable N :integer ;
Begin
 Write ("Entrez un nombre supérieur à 6 …" ) ;
 Read (N);
   While (N <= 6) do
      Write ("Saisie erronée. Recommencez") ;
      Read (N)
   End ;
 Write ("Saisie acceptée" ) ;
End.
```

Exemple 2 :

Ecrire un algorithme qui demande un nombre compris entre 10 et 20, jusqu'à ce que la réponse convienne. En cas de réponse supérieure à 20, on fera apparaître un message : « Plus petit ! », et inversement, « Plus grand ! » si le nombre est inférieur à 10.

Corrigé

```
Algorithme contrôle_saisie ;
Variable N : Entier ;
Debut
 N ← 0 ;
 Ecrire ("Entrez un nombre entre 10 et 20" ) ;
 tantque (N < 10 ou N > 20 ) faire
   lire (N ) ;
   si N < 10 alors
      Ecrire ("Plus grand !" ) ;
   sinon
      Ecrire ("Plus petit !" ) ;
   Fin;
 fin ;
Ecrire ("C'est bon" );
Fin.
```

Exemple 3 :

Ecrire un programme qui demande un nombre de départ, et qui ensuite affiche les dix nombres suivants. Par exemple, si l'utilisateur entre le nombre 17, le programme affichera les nombres de 18 à 27.

Corrigé

```
Program Exemple3 ;
Var N, i : integer ;
Begin
 Write ("Entrez un nombre : ") ;
 Read ( N ) ;
 Write ("Les 10 nombres suivants sont : " ) ;
 i ← N + 1
   while i<= N + 10 do
        Write (i) ;
   End;
End.
```

1.2 La boucle « for »

Dans le dernier exemple, vous avez remarqué qu'une boucle pouvait être utilisée pour augmenter la valeur d'une variable. Cette utilisation des boucles est très fréquente, et dans ce cas, il arrive très souvent qu'on ait besoin d'effectuer un nombre **déterminé** de passages. Or, a priori, notre structure **While** ne sait pas à l'avance combien de tours de boucle elle va effectuer (puisque le nombre de tours dépend de la valeur d'une booléen).

C'est pourquoi une autre structure de boucle est à notre disposition, c'est la boucle 'for'.

Exemple 1 :

On veut écrire un programme permettant d'afficher les entiers de 1 à 15.

```
Program incrementation;
Var Num : integer ;
begin
 Num := 0;
  While Num < 15  do
    Num := Num + 1 ;
    Write ("Passage numéro: ", Num) ;
  end;
end.
```

Equivaut à :

```
Program incrementation;
Var Num : integer ;
Begin
 for Num := 1 to 15 do
  Write ("Passage numéro : ", Num) ;
 end;
end.
```

L**a structure « for … End » n'est pas du tout indispensable**; on pourrait fort bien programmer toutes les situations de boucle uniquement avec un « While ». Le seul intérêt du « Pour » est d'épargner un peu d'effort au programmeur, en lui évitant de gérer lui-même la progression de la variable qui lui sert de compteur (on parle d'**incrémentation**).

Il faut noter que dans une structure « for … end », la progression du compteur est laissée à votre libre disposition. Dans la plupart des cas, on a besoin d'une variable qui augmente de 1 à chaque tour de boucle. On ne précise alors rien à l'instruction « for » ; celle-ci, par défaut, comprend qu'il va falloir procéder à cette incrémentation de 1 à chaque passage, en commençant par la première valeur et en terminant par la deuxième.

Nous pouvons donc maintenant donner la formulation générale d'une structure « Pour ». Sa syntaxe générale est :

```
for Compteur ← Initial à Final do
…
Instructions
…
end;
```

Exemple 2:

Ecrire un algorithme qui permet d'afficher les nombres pairs de 1 à 16.

```
algorithme Nombre_paires ;
Variable i :entier ;
debut
  pour i ←0 à 16 faire
       write('Num=', i);
       i←i+1;
  end;
end.
```

```
program Nombre_paires ;
Var i :integer ;
begin
  for i :=0 to 16 do
       write('Num=', i);
      i:=i+1;
  end;
end.
```

Les structures **While** sont employées dans les situations où l'on doit procéder à un traitement systématique sur les éléments d'un ensemble dont on ne connaît pas d'avance la quantité, comme par exemple :

- Le contrôle d'une saisie
- 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.

Les structures **for** sont employées dans les situations où l'on doit procéder à un traitement systématique sur les éléments d'un ensemble dont le programmeur connaît d'avance la quantité.

1.3 La boucle « Repeat … until »

Dans cette structure, le traitement est exécuté une première fois puis sa répétition se poursuit jusqu'à ce que la condition soit vérifiée.

Syntaxe:

```
repeat
  action ;
until condition ;
```

L'action dans cette boucle est toujours exécutée au moins une fois.

Exemple1 :

Ecrire un algorithme permettant d'afficher les valeurs entières de 1 à 10.

```
Algorithme Affichage ;
Variables i : entier ;
Début
      i← 1 ;
  répéter
      écrire ('i=',i) ;
      i←i+1 ;
  jusqu'à (i>10)
```

En pascal :

```
program Affichage ;
Var i : integer ;
begin
      i := 1 ;
  repeat
      write ('i=',i) ;
      i :=i+1 ;
  until (i>10)
```

Exemple2 :

Ecrire un algorithme qui permet de calculer et d'afficher les carrées des nombres entiers lus au clavier, l'exécution doit être terminée une fois un zéro est saisie.

```
Algorithm Nombre_paires ;
Variables n,p :entier ;
Debut
  n ←0 ;
  repéter
     écrire('Donner un nombre:');
     lire(n);
```

```
    p←n*n ;
    write('Le carrée=',p) ;
  until n=0
end.
```

Les instructions encadrées par les mots repeat et until constitue le bloc de la boucle qu'il faut répéter jusqu'à ce que la condition n=0 soit vérifiée. Donc le nombre de répétitions de cette boucle dépend des données fournies par l'utilisateur.

Remarque :

Dans la structure « repeat… until », la condition telle qu'elle est exprimée ci-dessus, constitue une condition d'arrêt de la boucle.

Chapitre 2 : Les variables indicées (les tableaux)

Lorsque dans un programme on a besoin de calculer la moyenne de trois notes d'un étudiant, on a besoin de trois variables, ainsi :

```
Var Note1, Note2, Note3 : reel
begin
 write ('donner les notes ..')
 read(note1)
 read(note2)
 read(note3)
 Moy=(note1+note2+note3)/3
 write('La moyenne=',Moy)
end
```

Si nous avons 12 matières, nous ayons besoin simultanément de 12 notes pour calculer la moyenne. Evidemment, la seule solution dont nous disposons à l'heure actuelle consiste à déclarer douze variables, appelées par exemple Notea, Noteb, Notec, etc. Bien sûr, on peut opter pour une notation un peu simplifiée, par exemple N1, N2, N3, etc. Mais cela ne change pas fondamentalement notre problème, car arrivé au calcul, et après une succession de douze instructions « Read » distinctes, cela donnera obligatoirement une chose du genre :

```
Moy ← (N1+N2+N3+N4+N5+N6+N7+N8+N9+N10+N11+N12)/12
```

C'est laborieux. Et pour un peu que nous soyons dans un programme de gestion avec quelques centaines ou quelques milliers de valeurs à traiter, alors là c'est plus délicat.

Si en plus on est dans une situation où l'on ne peut pas savoir d'avance combien il y aura de valeurs à traiter, là on est carrément bloqué.

C'est pourquoi la programmation nous permet **de rassembler toutes ces variables en une seule**, au sein de laquelle chaque valeur sera désignée par un numéro. Cela donnerait donc quelque chose du genre « la note numéro 1 », « la note numéro 2 », « la note numéro 8 ». C'est largement plus pratique.

Un ensemble de valeurs portant le même nom de variable et repérées par un nombre, s'appelle un tableau, ou encore une variable indicée.

Le nombre qui, au sein d'un tableau, sert à repérer chaque valeur s'appelle l'**indice**.

Chaque fois que l'on doit désigner un élément du tableau, on fait figurer le nom du tableau, suivi de l'indice de l'élément, entre crochets.

2.1 Notation et utilisation algorithmique

Un tableau doit être déclaré comme tel, en précisant le nombre et le type de valeurs qu'il contiendra.

1. Les "cases" sont numérotées à partir de zéro, autrement dit que le plus petit indice est zéro.

2. Lors de la déclaration d'un tableau, on précise la plus grande valeur de l'indice (différente, donc, du nombre de cases du tableau, puisque si on veut 12 emplacements, le plus grand indice sera 11).

```
Note : array[0..11] of real
```

NB : On peut créer des tableaux contenant des variables de tous types (réel, entier, caractères, booléens). Par contre on ne peut pas faire un mixage de types différents de valeurs au sein d'un même tableau.

Exemple :

Par exemple, pour effectuer notre calcul de moyenne, cela donnera par exemple :

```
Algorithme   Moyenne ;
Variable     Note : tableau[0..11] de réel ;
             Moy, Som : réel ;
Début
 pour i ← 0 à 11
    write ("Entrez la note n°", i ) ;
    read( Note[i]) ;
 fin
Som ← 0
 pour i ← 0 à 11
    Som ← Som + Note(i)
 end
 Moy ← Som / 12
 Write('La moyenne =', Moy) ;
end
```

En pascal :

```
program   Moyenne ;
Var      Note : array[0..11] of real ;
         Moy, Som : real ;
begin
 for i := 0 to 11 do
    write ("Entrez la note n°", i ) ;
    read( Note[i]) ;
 end;
Som := 0;
 for i := 0 à 11
    Som := Som + Note[i];
 end;
 Moy := Som / 12 ;
 Write('La moyenne =', Moy) ;
end.
```

NB : On a fait deux boucles successives pour plus de lisibilité, mais on aurait tout aussi bien pu n'en écrire qu'une seule dans laquelle on aurait tout fait d'un seul coup.

Remarque générale : l'indice qui sert à désigner les éléments d'un tableau peut être exprimé directement comme un nombre en clair, mais il peut être aussi une variable, ou une expression calculée.

Dans un tableau, la valeur d'un indice doit toujours :

- **être égale au moins à 0** : Donc Tab[6] est le septième élément du tableau Tab.
- **être un nombre entier** L'élément Tab[3,1416] n'existe jamais.
- **être inférieure ou égale au nombre d'éléments du tableau** (moins 1). Si le tableau **Tab** a été déclaré comme ayant 25 éléments, la présence dans une ligne, sous une forme ou sous une autre, de Tab[32] déclenchera automatiquement une erreur.

Remarque :

Il ne faut pas confondre l'**indice** d'un élément d'un tableau avec le **contenu** de cet élément. La troisième maison de la rue n'a pas forcément trois habitants, et la vingtième vingt habitants. En notation algorithmique, il n'y a aucun rapport entre i et tab[i].

2.2 Manipulation d'un tableau

Une fois déclaré, un tableau peut être utilisé comme un ensemble de variables simples. Les trois manipulations de base sont l'affectation, la lecture et l'écriture.

a- L'affectation :

Pour affecter une valeur à un élément i d'un tableau nommé par exemple T, on écrira : T(i) := valeur.

Par exemple, l'instruction : T[0] := 20 ; affecte au premier élément du tableau T la valeur 20. Pour affecter la même valeur à tous les éléments d'un tableau T de type numérique et de dimension 100, on utilise une boucle.

Par exemple :

```
pour i de 0 à 99  faire
    T[i] := 0 ;
fin
```

Cette boucle permet de parcourir le tableau T élément par élément et affecter à chacun la valeur 0. La variable i est appelée indice.

b- La lecture :

Comme les variables simples, il est possible aussi d'assigner des valeurs aux éléments d'un tableau lors de l'exécution c.à.d. les valeurs sont saisies par l'utilisateur à la demande du programme.

Exemple :

```
write ("Enter une note :") ;
read T(5) ;
```

Dans cet exemple, la valeur saisie est affectée au sixième (6ème) élément du tableau T.

c- L'écriture :

De façon analogue à la lecture, l'écriture de la valeur d'un élément donné d'un tableau s'écrira comme suit : **`write T[i]`**

Cette instruction permet d'afficher la valeur de l'élément i du tableau T.

Exemple:

Ecrire un algorithme qui permet de saisir les prix dans un tableau de taille 5, et les afficher par la suite.

```
program  Tab_prix ;
var  i : entier ;
     Prix : array[0..5] of real ;
begin
 for i de 0 à 4
    write ("Donner un prix :") ;
    read (Prix[i] );
  end
 for i de 0 à 4
    write( Prix[i]) ;
 end ;
end
```

L'exécution du programme permet d'initialiser le tableau Prix comme suit :

Prix

Prix(0)	10
Prix(1)	120
Prix(2)	10.6
Prix(3)	99.99
Prix(4)	250

Où les valeurs : 10, 120, 10.6, 99.99, 250 sont saisies par l'utilisateur.

Déroulement de l'algorithme :

Lecture :

Donner un prix : 10
Donner un prix : 120
Donner un prix : 10.6
Donner un prix : 99.99
Donner un prix : 250

Affichage :

10.00
120.00
10.60
99.99
250.00

Rechercher dans un tableau

Écrire un algorithme permettant de chercher un élément donné dans un tableau T.

```
Algorithm Recherche_Tab;
var i,x:integer;
     b:boolean;
     t :array[0..5] of integer;

begin
 writeln('entrer les valeurs du tableau...');
 for i:=0 to 5 do
    readln (t[i]);

 writeln('entrer une valeur à chercher ');
 readln(x);
 for i:=0 to 5 do
   if t[i]=x then
       b:=true;

 if b=true then
    writeln(x,'existe')
 else
     writeln(x,' n''existe pas');

end.
```