ALGORITHMES : TESTS ET BOUCLES

Les algorithmes que nous avons utilisés dans le chapitre précédent exécutent toujours la même tâche ce qui limite leur intérêt. Les tests et les boucles vont enrichir nos algorithmes leur permettant d'agir différemment en fonction des données entrées par l'utilisateur.

1. CONDITIONS

Une condition est une expression qui peut prendre l'une des deux valeurs suivantes **vrai** ou **faux**. On dit également que c'est une valeur de type **"logique"** ou **"booléen"**.

Les principaux opérateurs de comparaison que vous rencontrerez sont les suivants :

- égal à (= en pseudo code)
- différent de (!= en pseudo code)
- strictement supérieur (> en pseudo code)
- strictement inférieur (< en pseudo code)
- supérieur ou égal (> = en pseudo code)
- inférieur ou égal (< = en pseudo code)

Ces comparaisons n'ont un sens que si les variables que l'on compare sont de même type.

CONDITIONS COMPOSÉES

On peut écrire des conditions plus complexes en reliant des comparaisons à l'aide des opérateurs logiques **ET**, **OU** et **NON**.

- **Condition 1 ET condition 2** sera vraie si les deux conditions sont **toutes les deux vraies**.
  Par exemple, la condition : "**âge supérieur à 5 ET âge inférieur à 10**" sera vraie si la variable âge est **strictement comprise entre 5 et 10**.
- **Condition 1 OU condition 2** sera vraie si **l'une au moins** des deux conditions est vraie.
  Par exemple, la condition "**prénom=Jean OU nom=Dupont**" sera vraie pour :
  - Jean Dupont (conditions 1 et 2 vraies)
  - Jean Durand (condition 1 vraie)
  - Pierre Dupont (condition 2 vraie)

  mais fausse pour
  - Pierre Durand (conditions 1 et 2 fausses)
- **NON (condition 1)** sera vraie si et seulement si **condition 1 est fausse**.
  Par exemple : "**NON (x < 3)**" sera vraie si **x > = 3**

2. TESTS

DÉFINITION

Un **test** est une instruction qui permet d'effectuer un traitement différent selon qu'une condition est vérifiée ou non.

PREMIÈRE FORME

La première forme possible est la suivante :

```
si condition alors
   instructions fin si
```

Les instructions ne seront exécutées que **si la condition est vérifiée**. Par exemple :

```
variable
   x : entier
début algorithme
   lire x
   si x > 10 alors
      x prend la valeur 10
   fin si
   afficher x
fin algorithme
```

Si l'utilisateur entre un entier supérieur à 10 l'algorithme affichera 10 sinon il affichera le nombre saisi par l'utilisateur.

SECONDE FORME

La seconde forme est légèrement plus complexe :

```
si condition alors
   instructions 1
sinon
   instructions 2
fin si
```

Si la condition est vraie, l'algorithme effectuera les "instructions 1" puis passera aux instructions situées après le "fin si". **Si la condition est fausse**, l'algorithme effectuera les "instructions 2" puis passera aux instructions situées après le "fin si".

EXEMPLE

```
variables
   âge, prix : entier
début algorithme
   afficher "entrez votre âge :"
   lire âge
   si âge < 16 alors
      afficher "vous bénéficiez du tarif réduit"
      prix prend la valeur 10
   sinon
      afficher "vous ne bénéficiez pas du tarif réduit"
      prix prend la valeur 15
   fin si
   afficher "vous devez payer", prix, "euros"
fin algorithme
```

Si vous entrez **15** comme âge, vous obtiendrez le résultat suivant :

Vous bénéficiez du tarif réduit
Vous devez payer 10 euros

Si vous entrez **16** comme âge, vous obtiendrez :

Vous ne bénéficiez pas du tarif réduit
Vous devez payer 15 euros

3. BOUCLE

DÉFINITION

Une **boucle** permet de répéter un traitement un certain nombre de fois.

PREMIÈRE FORME

Boucle "Tant que"

```
tant que condition
   instructions
fin tant que
```

L'algorithme ci-dessus effectuera les instructions tant que la condition sera vraie. Dès que la condition

devient fausse, on se branchera sur l'instruction suivant le fin tant que.

EXEMPLE

```
variables
   nombre, somme: nombres
   continuer: texte
début algorithme
   continuer prend la valeur "oui" // initialisation
   afficher 'entrez un nombre :'
   lire nombre
   somme prend la valeur nombre
   tant que continuer="oui"
      afficher "entrez le nombre suivant"
      lire nombre
      somme prend la valeur somme+nombre
      afficher "voulez-vous continuer (oui/non)"
      lire continuer
   fin tant que
   afficher "la somme des nombres entrés est" somme
fin algorithme
```

L'algorithme précédent demande à l'utilisateur d'**entrer un premier nombre**.

Puis il lui demande **s'il veut entrer un autre nombre**.

Tant que l'utilisateur répond *"oui"*, l'algorithme lui **demande un nouveau nombre** qu'il **additionne** au contenu de la variable "somme".

Dès que l'utilisateur répond autre chose que *"oui"*, l'algorithme **sort de la boucle**, **affiche le total** et se **termine**.

DEUXIÈME FORME

Boucle "Pour"

EXEMPLE

```
variables
   i : nombre
...
début algorithme
...
pour i variant de 1 à 10
   instructions
fin pour
...
fin algorithme
```

L'algorithme ci-dessus va exécuter **dix fois** les *instructions* situées dans la boucle.

Plus précisément :

- **La première fois** que l'algorithme va rencontrer l'instruction *"pour i variant de 1 à 10"*, il va affecter la valeur 1 à i ; comme i est strictement inférieur à 10, il passe ensuite aux *instructions* situées **à l'intérieur** de la boucle
- après les avoir exécutées, la ligne *"fin pour"* va faire boucler l'algorithme et le faire revenir à l'instruction *"pour i variant de 1 à 10"*
- **La seconde fois (et les fois suivantes...)** que l'algorithme va exécuter l'instruction *"pour i variant de 1 à 10"*, il va :
  - ajouter 1 à i (on dit **incrémenter i**)
  - si i est inférieur ou égal à 10, il passe aux *instructions* situées **à l'intérieur** de la boucle
  - si i est supérieur à 10, il passe aux instructions situées **après** la ligne *"fin tant que"*

REMARQUE

Si l'on souhaite incrémenter l'indice avec une valeur différente de 1 on utilise l'instruction :

```
pour i variant de ... à ... avec un pas de ...
```

Par exemple :

```
pour i variant de 2 à 8 avec un pas de 2
```

i va prendre successivement les valeurs : 2 ; 4 ; 6 ; 8 (et il quittera la boucle lorsqu'il vaudra 10)

EXEMPLE

L'algorithme ci-dessous affiche les carrés des 21 premiers nombres entiers naturels (de 0 à 20)

```
variables
   n : nombre
   c : nombre
début algorithme
pour n variant de 0 à 20
   c prend la valeur n*n
   afficher "Le carré de ", n, " est ", c
fin pour fin algorithme
```

REMARQUE

On utilise généralement une instruction *"pour"* lorsqu'on connaît le nombre d'itérations à réaliser dès le début de la boucle et une instruction *"Tant que"* lorsque ce nombre est inconnu ou difficile à déterminer.