Chap 3 : Les structures conditionnelles

En programmation procédurale comme en algorithmique, l'ordre des instructions est primordial.
Le processeur exécute les instructions dans l'ordre dans lequel elles apparaissent dans le programme.
On dit que l'exécution est séquentielle.
Une fois que le programme a fini une instruction, il passe à la suivante. Cependant des fois, il est nécessaire que le processeur n'exécute pas toutes les instructions, ou encore qu'il recommence plusieurs fois les mêmes instructions.
Pour cela, il faudra casser la séquence. C'est le rôle des structures de contrôle.

Il existe deux grands types de structures de contrôle :
- les structures conditionnelles vont permettre de n'exécuter certaines instructions que sous certaines conditions ;
- les structures répétitives, encore appelées boucles, vont permettre de répéter des instructions un certain nombre de fois, sous certaines conditions.

1) Les structures conditionnelles

a. La conditionnelle

Un programme s'exécute de manière séquentielle, c'est à dire instruction par instruction. Il se peut qu'un programmeur souhaite employer certaines instructions seulement en fonction de certaines conditions. L'action est subordonnée à la vérification d'une condition. L'instruction conditionnelle *si* permet cela.

```
si (ExpressionBooléenne) alors
      BlocInstructions
fin-si
```

Exemple 1 : indiquer si un nombre est positif, l'indiquer à l'utilisateur

```
si (x > 0) alors
      afficher ("x est positif")
fin-si
```

L'*ExpressionBooléenne* détermine si les instructions *BlocInstructions* vont être exécutées.
L'exécution de la conditionnelle se fait en deux temps :
La condition est évaluée (calculée)
Si la condition est vraie, les instructions *BlocInstructions* sont exécutées.

Il y a deux points implicites :
Si l' *ExpressionBooléenne* vaut *faux*, il ne se passe rien (c.-à-d. Les instructions *BlocInstructions* ne sont pas exécutées).
Quand l'instruction *si* est terminée (finie d'exécutée, donc soit *instruction* a été exécutée, soit rien), l'instruction suivant le *si* est exécutée (exécution séquentielle).

Ainsi :

```
instruction0
si (ExpressionBooléenne) alors
      instruction1
fin-si ;
instruction2
```

est équivalent

(si le test est vrai)

```
instruction0
instruction1
instruction2
```

(si le test est faux)

```
instruction0
instruction2
```

Exemple 2 :
L'entreprise accorde une remise de 10 % sur le prix TTC pour tout achat de plus de 100 pièces ; écrire le programme calculant le prix TTC en fonction de la quantité et du prix unitaire.

```
programme remise

var
   prix_u, prix_ttc : réel
   quantite         : entier

début
   afficher ("\nEntrez le prix unitaire : ")
   saisir     (prix_u)
   afficher ("\nEntrez la quantité achetée)
   saisir     (quantité)
   prix_ttc ← quantité * prix_u
   si (quantité > 100) alors
         prix_ttc ← prix_ttc * 0.9
   fin-si ;
   afficher ("\nLe prix ttc à payer est de ", prix_ttc, " €.")
fin
```

Le *si* est une instruction comme les autres. De même, les instructions composant le *si* peuvent elles-mêmes être des *si*.

b. L'alternative

Tel qu'il est présenté ci-dessus, le *si* manque d'efficacité.
Prenons le cas d'un nombre, on veut afficher si ce nombre est positif ou négatif :

```
si (x > 0)alors
      afficher (x," est positif")
fin-si
si (x <= 0) alors
      afficher (x, " est négatif ou nul")
fin-si
```

Souvent, donc, on souhaite exécuter des instructions si un test est vrai (si une condition est réalisée), mais on veut aussi exécuter d'autres instructions si ce même test est faux. C'est-à-dire que selon qu'un test est vrai ou faux, on réalisera un bloc d'instructions ou un autre. Ce cas est tellement fréquent que le *si* le prend en compte.

L'alternative permet d'exécuter une instruction (bloc d'instructions) ou une autre en fonction d'une condition.

```
si (booléen) alors
      instruction1
sinon
      instruction2
fin-si
```

L'exécution de l'instruction *si* réalise l'évaluation du booléen. Si il vaut *vrai*, *instruction1* est exécutée, sinon (le booléen est alors faux) *instruction2* est exécutée. L'instruction suivant le *si* est ensuite exécutée. Ainsi :

```
instruction0
si (booléen)
   alors
      instruction1
   sinon
      instruction2
fin-si
instruction3
```

est équivalent à

(si le test est vrai)

```
instruction0
instruction1
instruction3
```

(si le test est faux)

```
instruction0
instruction2
instruction3
```

Modifier le programme précédent pour que l'on indique si un nombre est strictement positif ou négatif :

```
programme Si_2
var
   entier x

début
   saisir (x)
   si (x > 0) alors
   |     afficher ("x est positif.")
   sinon
   |
   |     |afficher ("x est négatif ou nul")
   |
   fin-si
fin
```

Remarque :
Les structures conditionnelles peuvent être imbriquées (c'est-à-dire incluses les unes dans les autres).

Modifier le programme précédent pour que l'on indique si un nombre est positif ou négatif ou nul :

```
programme Si_2
var
   entier x

début
   saisir (x)
   si (x > 0) alors
   |     afficher ("x est positif.")
   sinon
   |     si (x < 0) alors
   |     |     afficher ("x est négatif")
   |     |sinon
   |     |     afficher ("x est nul")
   |     fin-si
   fin-si
fin
```

EXERCICES

1. Ecrire un algorithme qui saisit un entier au clavier et qui affiche un message permettant de dire si ce nombre est égal à 1.

```
saisir (x)
si (x = 1) alors
      afficher ("x = 1")
sinon
      afficher ("x <> 1")
fin-si
```

2. Ecrire un algorithme qui saisit deux nombres au clavier et qui indique si ces nombres sont égaux :

```
si (x = y)   alors
      afficher ("ils sont égaux")
sinon
      afficher ("ils ne sont pas égaux")
fin-si
```

3. Ecrire un algorithme qui saisit un nombre entier au clavier et qui indique si ce nombre est pair ou impair.

```
Saisir (x)
si (x mod 2 = 0) alors
      afficher ("pair")
   sinon
      afficher ("impair")
fin-si
```

c. L'alternative (Choix)

Une compagnie de transports en commun d'une ville gère en partie ses propositions de réductions de prix sur les billets en fonction de l'âge du voyageur.
Elle a décidé notamment d'utiliser le barème suivant :

- si le voyageur a moins de 5 ans, le billet est gratuit;
- s'il a entre 5 et 17 ans, il bénéficie d'une réduction de 50%
- s'il a entre 18 et 25 ans, il bénéficie d'une réduction de 20%
- entre 26 et 64 ans, pas de réduction,
- plus de 65 ans, il bénéficie d'une réduction de 25%.

Ecrivez l'algorithme du calcul du prix d'un billet en fonction de l'âge en utilisant des alternatives :

```
programme scoring_1

var
   âge, prix : entier

début
   afficher ("Entrer l'âge du client :")
   saisir (âge)
   si (âge < 5)alors
   |     prix ← 0
   sinon
   |     si (âge < 18) alors
   |     |     prix ← prix/2
   |     sinon
   |     |     si (âge < 26) alors
   |     |     |     prix ← prix*20/100
   |     |     sinon
                        si (age < 65)  alors
                              | prix ← prix
                        sinon
   |     |     |               prix ← prix*25/100
                        fin-si
   |     |     fin-si
   |     fin-si
   fin-si
   afficher ("prix du billet avec reduction:", prix) ;
fin
```

L'indentation n'y fait rien :ce programme n'est pas lisible, les différents cas n'apparaissent pas clairement. Lorsque l'on souhaite exécuter différentes instructions selon la valeur d'une variable, l'instruction *si* n'est pas très adaptée, car elle impose une succession peu pratique et lisible de *si alors sinon* imbriqués.

Ce cas est suffisamment fréquent (menus…) pour qu'une instruction adaptée existe.

```
selon <expression>
   <liste de valeurs 1> : <instruction 1>
   <liste de valeurs 2> : <instruction 2>
                          …
   <liste de valeurs n> : <instruction n>
   sinon    :             <instruction_sinon>
fin-selon
```

L'exécution de l'instruction *selon* réalise les opérations suivantes :
L'expression est évaluée.

Les listes de valeurs sont examinées une par une, en commençant par la première. Dès que l'une d'elles contient la valeur de l'expression, l'instruction correspondante est exécutée. L'instruction *selon* est alors terminée.
Si aucune des listes de valeurs ne contient la valeur cherchée et que *selon* contient une branche *sinon*, on exécute <*instruction_sinon*> ; l'instruction *sinon* est alors terminée.

Une liste de valeur peut être soit une seule valeur, soit plusieurs valeurs séparées par des « , ». On peut également spécifier des intervalles de valeurs en séparant les bornes par « .. ».

Exemples :

2
2 ,4, 6, -2, 0
0..10
0..10, 100..200, 50

Le programme précédent devient :

```
programme scoring_2

var
   âge, prix : entier

début
   afficher ("Entrer l'âge du client :")
   lire (âge)
   selon âge
      0..4   : prix ← 0
      5..17  : prix ← prix/2
     18..25  : prix ← prix*20/100
     26..64  : prix ← prix
      sinon  : prix ← prix*25/100
   fin-selon
   afficher ("prix avec reduction", prix)
fin
```

Remarques :
La branche *sinon* (facultative) permet de regrouper toutes les valeurs qui n'ont pas été énumérées. (Par exemple, un âge supérieur à 64, mais aussi un âge négatif, attribuera 150 points.) Si la branche *sinon* n'est pas présente et qu'aucune liste ne contient la valeur de l'expression, *selon* n'a aucun effet.
Les listes de valeurs doivent être disjointes (ne pas partager de valeurs) puisque l'on exécute que l'instruction associée à la première liste contenant la valeur de l'expression.
Pour diverses raisons, l'expression (et donc les valeurs des listes) ne peuvent être que des entiers, des caractères ou des booléens (peu d'intérêt pour les booléens). Les réels, chaînes ou autres ne sont pas autorisées. (En fait :uniquement types énumérés.)
Les valeurs des listes doivent bien entendu être de même type que l'expression.

2) Les expressions booléennes

Une expression booléenne est une expression dont la valeur est soit VRAI soit FAUX. Il existe plusieurs types d'expressions booléennes.

a. Les comparaisons simples

Dans nos exemples, les conditions que nous avons rencontrées (ex : x>0) sont des conditions simples. Une condition simple est une comparaison de deux expressions de même type.(x> 0 type entier ou réel)
Les symboles de comparaison utilisable en algorithmique sont : < , > , <= , >=, =, <>_

b. Les conditions complexes

Les expressions booléennes peuvent aussi être complexes, c'est à dire formées de plusieurs conditions simples ou variables booléennes reliées entre elles par les opérateurs logiques **et**, **ou**, **non**.

Exemples :
Si a < 0 **et** b < 0 Alors …
Si ((a +3 = b) **et** (c < 0)) **ou** (a = (c *2)) Alors …

Et:
Une expression booléenne composée de deux conditions simples reliées par 'et' est vraie si les deux conditions sont vraies. Si l'une des deux conditions est fausse alors
La condition a < 0 et b < 0 est vraie si a < 0 est vraie et si b < 0 est vraie.

Table de vérité :

A	B	A ET B
V	V	V
V	F	F
F	V	F
F	F	F

Ou:
Une condition composée de deux conditions simples séparées par ou est vraie si au mois l'une des conditions simples est vraie.
La condition a <0 ou b < 0 est vraie si a < 0 ou si b < 0 ou si a et b sont négatifs.

Table de vérité :

A	B	A OU B
V	V	V
V	F	V
F	V	V
F	F	F

Non :
Une conditions précédée par non est vraie si la condition simple est fausse et inversement.
La condition non (a < 0) est vraie si a >=0.

Table de vérité :

A	NON A
V	F
F	V

L'usage des parenthèses permet de régler d'éventuels problèmes de priorités des opérateurs logiques.

c. Les variables booléennes

Les variables booléennes, comme les expressions booléennes, sont soit vraies, soit fausses. On peut donc affecter une expression booléenne à une variable booléenne et on peut donc ainsi trouver une variable booléenne à la place d'une expression booléenne.

Exemple :

```
Algorithme intervalles
Variables
appartient : booléen
nb : réel
Début
   Afficher "veuillez entrer un nombre réel"
   Saisir nb
   appartient ← (nb<10 ET nb> 5) OU (nb >15 ET nb <20)
   Si appartient Alors
      Afficher "Le nombre appartient aux intervalles définis"
   Sinon
      Afficher "Le nombre n'appartient pas aux intervalles définis"
   Finsi
Fin
```

Que permet de faire ce programme ?
Ce programme saisit un nombre et affiche si ce nombre est compris dans les intervalles] 5-10[ou]15-20[

Exercice 1:
Évaluer les expressions booléennes suivantes :

	m vaut 2	m vaut 1
m>2	F	V
m<=2	V	V

	m vaut 2 r vaut 1
m>2 et r=1	F
m>2 ou r=1	V
m<=3 et r <> 1	F
m<3 et r >=0	V

m vaut 3 r vaut -2
F
V
F
F