Chapitre 3

Les structures conditionnelles

Les instructions d'un algorithme s'exécutent les unes après les autres, dans l'ordre où elles ont été écrites. On exécute normalement les instructions de la première à la dernière, sauf lorsqu'on rencontre une structure conditionnelle qui permet de suivre différents chemins suivant les circonstances. L'instruction ne sera exécutée que sous certaines conditions.

I. Structure conditionnelle simple (à un choix)

Une structure conditionnelle simple contient un seul bloc d'instructions. Selon une condition (expression logique ou booléenne), on décide est-ce-que le bloc d'instructions est exécuté ou non. Si la condition est vraie, on exécute le bloc, sinon on ne l'exécute pas. La syntaxe d'une structure conditionnelle simple est comme suit :

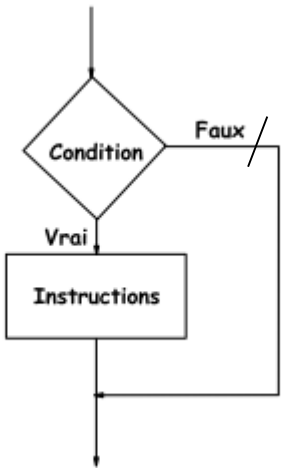
En algorithmique

si (condition) **alors**
    instruction(s) ;
finsi ;

En langage C

```
if (condition)
{
instruction(s);
}
```

Organigramme correspondant :



Exemple :

Écrire un algorithme qui demande un entier (A) à l'utilisateur, teste si ce nombre est positif (A≥ 0) et affiche "positif".

Solution :

algorithme positif ;
debut
    var A : **entier** ;
    ecrire ("entrer la valeur de A ") ;
    lire (A) ;
  si (A ≥ 0) **alors**
    ecrire (" positif ") ;
  finsi ;
fin.

II. Structure conditionnelle alternative (à deux choix)

Une structure conditionnelle alternative contient deux blocs d'instructions : on est amené à décider entre le premier bloc ou le second. Cette décision est réalisée selon une condition (expression logique ou booléenne) qui peut être vraie ou fausse. Si la condition est vraie on exécute le premier bloc, sinon on exécute le second. La syntaxe d'une structure conditionnelle alternative est comme suit :

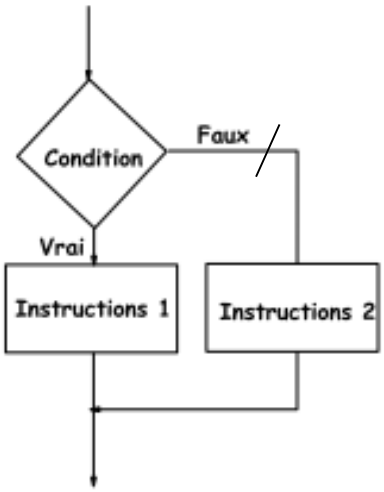
En algorithmique

si condition **alors**
    instructions 1
sinon
    instructions 2
finsi ;

En langage c

```
if (condition)
{
   instructions 1;
}
else
{
   instructions 2;
}
```

Organigramme correspondant



Exemple :

Écrire un algorithme qui demande un entier (A) à l'utilisateur, teste si ce nombre est positif ($A \geq 0$) ou non, et affiche "positif" ou "négatif".

Solution :

algorithme positif ;
debut
    var A : **entier** ;
    ecrire ("entrer la valeur de A ") ;
    lire (A) ;
    si ($A \geq 0$) **alors**
        ecrire ("positif ") ;
     sinon
        ecrire ("négatif ") ;
    finsi ;
fin.

III. Structures conditionnelles imbriquées

Les structures conditionnelles imbriquées fait appel à plus de deux blocs d'instructions. L'exécution d'un bloc entraîne automatiquement la non-exécution des autres blocs. La syntaxe d'une structure conditionnelle imbriquée est comme suit :

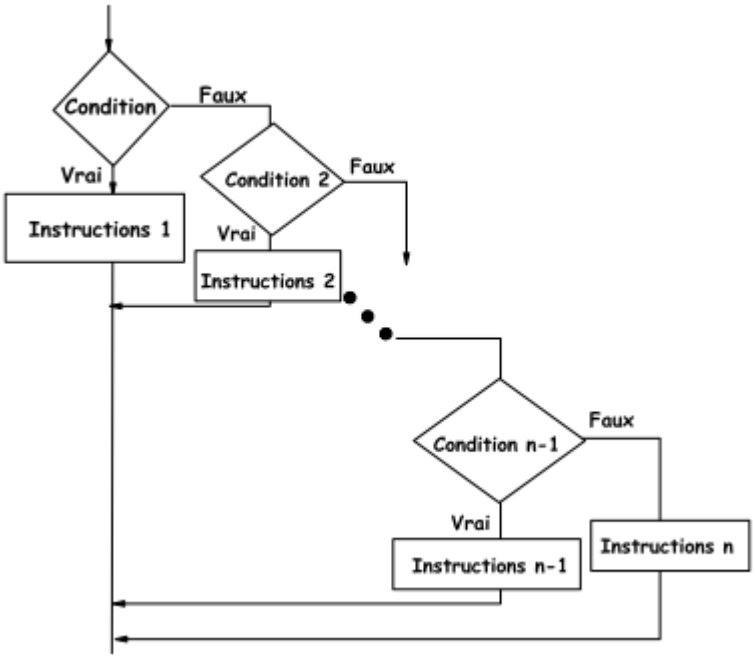
En algorithmique

```
si (condition) alors
      instructions 1 ;
sinon
      si (condition 2) alors
            instructions 2 ;
      sinon
                  .
                  .
                  .
            si (condition (n-1)) alors
                  instructions (n-1) ;
            sinon
                  instructions n ;
            finsi ;
       finsi ;
finsi ;
```

En langage c

```
if (condition)
{
     instructions 1 ;
}
else
     if (condition2)
     {
          instructions 2 ;
     }
          .
          .
          .
     if (condition(n-1))
     {
          instructions(n-1) ;
     }
     else
     {
          Instructions n ;
     }
```

Organigramme correspondant



Exemple :

Écrire un algorithme qui demande un entier à l'utilisateur, teste si ce nombre est strictement positif, nul ou strictement négatif, et affiche ce résultat .

Solution :

```
algorithme positif ;
var A : entier ;
  ecrire ("entrer la valeur de A") ;
  Lire (A) ;
  si (A ≥ 0) alors
    ecrire ("positif") ;
  sinon
    si (A = 0) alors
          ecrire ("nul" ) ;
    sinon
          ecrire ("négatif" ) ;
    finsi ;
  finsi ;
fin.
```

VI. Structure conditionnelle de choix multiple

Le choix d'un traitement se fait suivant la valeur d'un sélecteur. Cette structure permet d'éviter le recours à une structure conditionnelle imbriquée et offre une meilleure lisibilité de la solution.

En algorithmique

```
selonque selecteur faire
      Valeur1 : instructions 1 ;
      Valeur2 : instructions 2 ;
           ⋮
      Valeu n : instructions n ;
 sinon
       instructions n+1 ;
finselonque ;
```

En langage c :

```
switch (selecteur) {
case Valeur1 :
   instructions 1;
break;
case Valeur2 :
   instructions 2;
break;
  ⋮
case Valeurn :
   instructions n;
break;
default:
   instructions n+1;
}
```

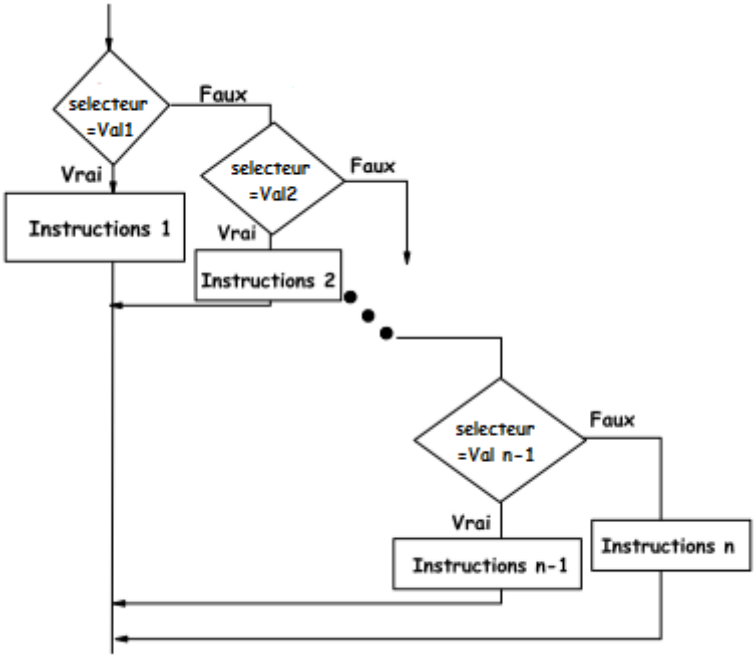
Remarques :

- Le sélecteur doit être de type entier ou caractère et doit avoir une valeur avant d'être impliqué dans le ***Selonque***.
- ***Selecteur*** et ***valeur1***, ***valeur2***, …, ***valeurN*** doivent être du même type.
- Les instructions relatives aux valeurs peuvent comporter également d'autres structures conditionnelles.
- ***Valeuri*** peut être un ensemble des valeurs qui seront séparées par des virgules ou intervalle des valeurs (borne_inf .. borne_sup).
- Si expression n'est égale à aucune des valeurs allant de ***valeur1*** à ***valeurN***, c'est le bloc ***Sinon*** qui est exécuté. Ce bloc est optionnel.

Exécution du selonque

On suppose que la valeur du sélecteur est connue avant de commencer le ***Selonque*** (soit par une lecture, soit par une affectation). À partir de cette valeur, on va s'aiguiller directement au traitement (instructions) correspondant à cette valeur puis l'exécuter. À la fin de l'exécution, on continuera la suite de l'exécution du programme à partir de la 1ère instruction qui vient après ***finselonque***. Dans le cas où la valeur du sélecteur n'existe pas parmi les choix possibles, on exécutera le traitement Sinon du ***Selonque***.

Organigramme correspondant



V. Structure de contrôle de branchements / sauts (l'instruction "aller à ")

Un algorithme utilise l'instruction *"aller à"* ainsi que des étiquettes pour faire des branchements. L'étiquette donne une position dans l'algorithme, l'instruction *"aller à"* demande à l'algorithme d'aller directement à l'endroit de l'étiquette désigné pour continuer l'exécution.

La syntaxe d'un branchement est comme suit :

En algorithmique

aller à <etiq>

⋮

<etiq> :

⋮

En langage c :

goto <etiq>;

⋮

<etiq> :

⋮

Exemple :

algorithme branchement ;
var a, b: **entier** ;
debut
lire (a) ;
1 : b ←a ;
si b mod 2 = 0 **alors**
aller à 2 ; *// branchement conditionnel (branchement appartenant au bloc si)*
finsi ;
a ← a+1 ;
aller à 1 ; *// branchement inconditionnel (branchement n'appartenant pas au bloc si)*
2 : ecrire ("c'est pair") ;
Fin.

Dans l'exemple ci-dessus, il y a deux étiquettes : l'étiquette 1 (b←a) et l'étiquette 2 (ecrire ('c'est pair'))**.** Voici un exemple de déroulement de l'algorithme pour a = 5.

Variables / Instructions	A	B	Affichage
Lire (a)	5		
b← a	5	5	
b mod 2 = 0 ←faux on n'entre pas au bloc du si a← a+1 ;	6	5	
aller à 1 b ← a ;	6	6	
b mod 2 = 0 ←vrai on entre au bloc si aller à 2 ➡ecrire('c'est pair')	6	6	C'est pair

Remarque :

Un branchement en lui-même n'est pas très intéressant. La pratique des informations a montré que l'utilisation des goto donne souvent des programmes non maintenables (impossible à corriger ou modifier). Les problèmes qu'ils posent ont amené les programmeurs expérimentés à ne s'en servir qu'exceptionnellement.