

Chapitre 2 : Les structures de contrôle conditionnelles

Qu'arrive-t-il lorsqu'un algorithme doit choisir entre plusieurs actions possibles ? Comment un algorithme peut-il décider d'exécuter une instruction plutôt qu'une autre ? Peut-on faire en sorte qu'un algorithme réagisse différemment selon les données saisies par l'utilisateur ?

Ce chapitre répondra à ces questions.

2.1 Introduction

Dans les algorithmes précédents, les instructions étaient exécutées de manière séquentielle, c'est-à-dire toujours dans le même ordre, une après l'autre. Or, dans la plupart des situations réelles, un algorithme doit être capable de prendre des décisions en fonction de certaines conditions. Ces dernières sont indispensables pour adapter le comportement d'un algorithme selon différents cas.

2.2 Intérêt

Afin de comprendre l'intérêt de l'utilisation des structures conditionnelles, revenons au chapitre 1 dans lequel nous avons écrit un algorithme qui calcule la moyenne d'une matière à partir des notes de TD et de l'examen. Cependant, si nous souhaitons maintenant déterminer si cette matière est acquise ou non, nous faisons face à une nouvelle situation : le résultat à afficher dépend de la valeur de la moyenne. Or, avec les instructions de base (affectation, lecture, écriture), nous ne pouvons pas exprimer ce choix entre deux actions possibles. C'est pourquoi nous avons besoin d'un nouveau type d'instruction : les structures conditionnelles, qui permettent à l'algorithme de prendre une décision selon qu'une condition soit vraie ou fausse et d'orienter, ainsi, l'exécution d'un algorithme selon cette même condition.

2.3 Notion de condition

Une condition est une expression logique qui permet à l'algorithme de tester une situation et de prendre une décision selon le résultat de ce test.



Remarque :

En mathématiques, on écrit souvent : $10 \leq x \leq 20$ pour dire que x est compris entre 10 et 20. Cependant, en algorithmique (et en langage C), cette écriture n'est pas valable telle quelle, car une expression ne peut comparer que deux valeurs à la fois. Il faut donc décomposer cette condition en deux comparaisons reliées par un opérateur logique : $(10 \leq x)$ et $(x \leq 20)$

2.4 Test alternatif simple (Si.. alors)

Le test alternatif simple permet d'exécuter une instruction seulement si une condition est vraie. Si la condition est fausse, l'instruction est simplement ignorée.

Syntaxe

Algorithmique :

```
Si (condition) alors  
    Instructions ;  
Fin-Si ;
```

En langage C :

```
if (condition) {  
    // instructions ;  
}
```

Exemple:

Algorithme Majorite ;
Variables age : entier;

Début

```
Ecrire("Veuillez introduire votre âge :") ;  
Lire(age) ;  
Si (age >= 18) alors  
    Ecrire("Vous êtes majeur.");  
Fin-Si ;  
Fin.
```

2.5 Test alternatif double (Si .. Sinon)

Le test alternatif double permet de faire un choix entre deux blocs d'instructions : l'un s'exécute si la condition est vraie, l'autre s'exécute si elle est fausse.

Syntaxe

Algorithmique :

```
Si (condition) alors  
    Instructions ;  
Sinon Instructions ;  
Fin-Si ;
```

En langage C :

```
if (condition) {  
    // instructions ; }  
else {  
    // instructions ;  
}
```

Exemple : On améliore l'algorithme précédent

Algorithme Majorite ;

Variables age : entier;

Début

Ecrire("Veuillez introduire votre âge :") ;

Lire(age) ;

Si (age >= 18) **alors**

 Ecrire("Vous êtes majeur.");

Sinon Ecrire("Vous êtes mineur.");

Fin-Si ;

Fin.

2.6 Choix multiple (structure conditionnelle imbriquée ou selon)

Le choix multiple permet de tester plusieurs cas possibles d'une même variable ou expression. Il peut être fait de deux manières :

- En utilisant plusieurs *si/sinon*,
- Ou en utilisant la structure selon (*switch*) pour des cas discrets.

Syntaxe (avec SI imbriqués) :

Algorithmique :

Si (condition1) **alors**

 Instructions1 ;

Sinon

Si (condition2) **alors**

 Instructions2 ;

Sinon Instructions ;

Fin-Si ;

Fin-Si ;

En langage C :

```
if (condition1)
    { // instructions1; }
else
    if (condition2)
        { // instructions2; }
    else
        { // instructions ; }
```

Syntaxe (avec Selon)

Algorithmique :

Selon (id_variable ou Expression) **faire**

cas valeur1: instructions pour valeur1 ;

cas valeur2: instructions pour valeur2 ;

 ...

cas valeurN: instructions pour valeurN ;

autre: instructions si aucune valeur ne correspond ;

FinSelon ;

En langage C :

```
switch (id_variable ou Expression) {
    case valeur1:
        // instructions pour valeur1
        break;
    case valeur2:
        // instructions pour valeur2
        break;
    case valeurN:
        // instructions pour valeurN
        break;

    default: /* instructions si aucune
valeur ne correspond*/
        break; }
```




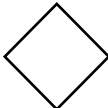


Exemple en C :

```
#include<stdio.h>
int main(){
int jour;
/* Les jours sont représentés en entier car selon n'accepte pas les
chaines de caractères*/
printf("Veuillez intrduire un jour\n");
scanf("%d", &jour);
switch (jour) {
    case 1: printf("Dimanche\n");
            break;
    case 2: printf("Lundi\n");
            break;
    case 3: printf("Mardi \n");
            break;
    case 4: printf("Mercredi\n");
            break;
    case 5: printf("Jeudi\n");
            break;
    case 6: printf("Vendredi\n");
            break;
    case 7: printf("Samedi\n");
            break;

    default: printf("Jour inconnu\n");
}
return 0 ;}
```

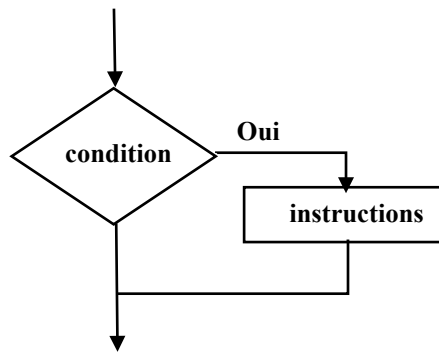
2.7 Représentation en organigramme (Algorigramme)

Un organigramme est la représentation graphique de la résolution d'un problème. Il est similaire à un algorithme. Chaque type d'instructions dans l'algorithme possède une représentation dans l'organigramme. Les symboles utilisés dans les organigrammes sont :

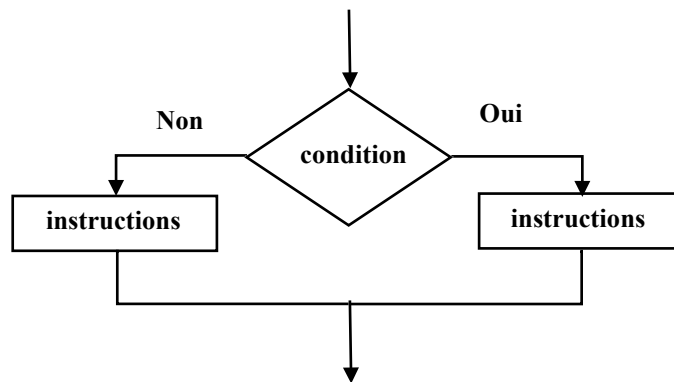
	Représente le début et la fin de l'organigramme.
	Entrées / Sorties : Lecture des données et écriture des résultats.
	Calculs, Traitements
	Tests et décisions.
	Ordre d'exécution des opérations (Enchainement).
	Connecteur.

2.8 Les structures conditionnelles en organigramme

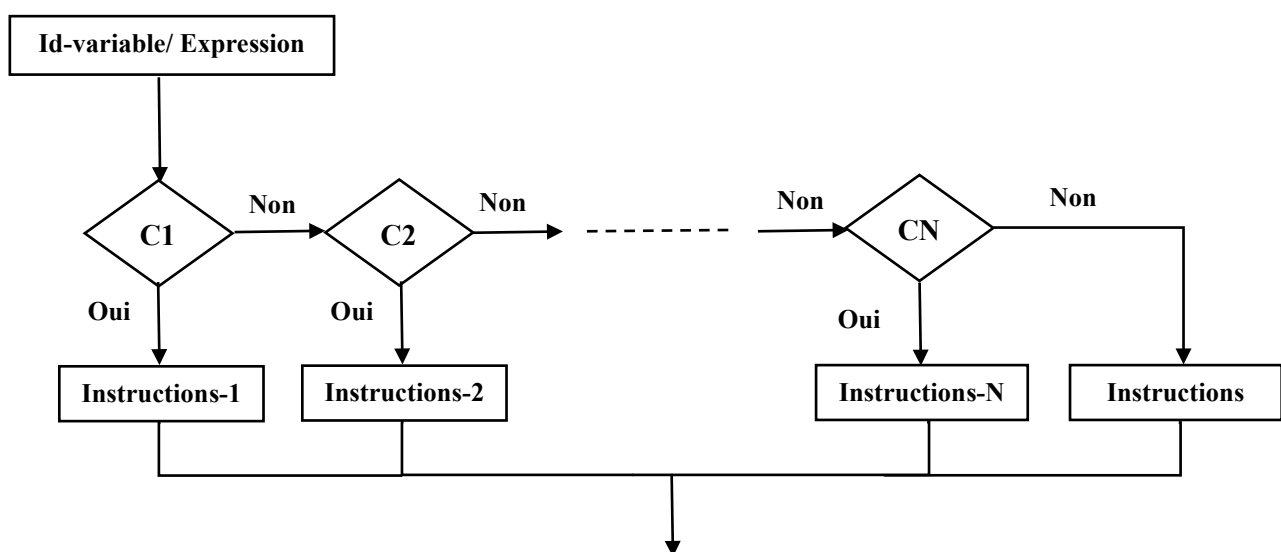
a. Le test alternatif simple



b. Le test alternatif double



C . Choix multiple (Selon)



2.9 Conclusion

Dans ce chapitre, nous avons étudié les structures conditionnelles, éléments essentiels permettant à un algorithme de prendre des décisions et d'adapter son comportement selon différentes situations. Grâce à ces structures, un programme peut désormais réagir intelligemment aux données d'entrée et choisir le chemin d'exécution approprié.

Cependant, de nombreux problèmes nécessitent non seulement de prendre des décisions, mais aussi de répéter certaines actions plusieurs fois, c'est pourquoi, dans le prochain chapitre, nous aborderons les structures de répétition.

Test d'acquisition des connaissances

1. Questions à choix multiples (QCM) :

1.1 Quelle instruction en langage C permet d'exécuter un bloc uniquement si une condition est vraie ?

- a) `while`
- b) `if`
- c) `for`
- d) `switch`

1.2 Quel est l'opérateur logique en C pour "ET" ?

- a) `||`
- b) `!`
- c) `==`
- d) `&&`

1.3 Quelle structure permet de traiter plusieurs cas à partir de la valeur d'une variable ?

- a) `if`
- b) `while`
- c) `switch`
- d) `do...while`

2. Vrai ou faux :

2.1 L'instruction `else` est obligatoire après un `if`.

2.2 L'opérateur `!=` signifie "égal à".

2.3 La structure `switch` est utile uniquement pour les types entiers et caractères.

3. Questions ouvertes / codage :

En utilisant `switch`, écris un programme en C qui affiche le nom des wilayas de l'Algérie correspondant à leurs codes (Ex 6 Béjaïa).

Corrigé:

QCM :

1.1 → b) `if`

1.2 → d) `&&`

1.3 → c) `switch`

Vrai/Faux :

2.1 → Faux

2.2 → Faux

2.3 → Vrai
