Chapitre

3

Les structures conditionnelles

Objectif

En programmation nous nous trouvons souvent devant des cas nécessitant l'étude de plusieurs situations qui ne peuvent pas être traitées tous durant l'exécution. Le programme ne traite qu'une seule situation selon les données introduites. On parle ici de choix et de conditions. Pour pouvoir réaliser ce cas de programmation on sollicite les structures conditionnelles qui font l'objectif de ce chapitre.

Introduction

Il est souvent nécessaire lorsque l'on écrit un programme de distinguer entre plusieurs cas conditionnant l'exécution de telles ou telles instructions. Pour ce faire, on utilise une structure alternative (Conditionnelle) : **Si on est dans tel cas on fait ceci sinon on fait cela.**

Dans les structures conditionnelles on se base sur ce qu'on appelle ***prédicat*** ou ***condition***. Ce dernier est un énoncé ou proposition qui peut être **vrai** ou **faux**. Dans l'établissement des conditions on utilise des opérateurs dit de comparaison, à retenir :

Symbole en Algo	Symbole en C	Rôle	Formalisme
<	<	Strictement inférieur à ...	x < y (x est inférieur à y)
<=	<=	Inférieur ou égal à ...	x <= y (x est inférieur ou égal à y)
>	>	Strictement supérieur à ...	x > y (x est supérieur à y)
>=	>=	Supérieur ou égal à ...	x >= y (x est supérieur ou égal à y)
=	==	Egal à ...	x == y (x est égal à y)
≠	!=	Différent de ... (non égal à ...)	x != y (x est différent de y)

Tableau 8. Opérateurs de comparaison

Comme on utilise des opérateurs dit logiques, à retenir également :

Symbole en Algo	Symbole en C	Rôle	Formalisme
Ou	||	OU logique Vérifie qu'une des conditions est réalisée	((condition1) || (condition2))
Et	**&&**	ET logique Vérifie que toutes les conditions sont réalisées	((condition1) && (condition2))
Non	**!**	NON logique Inverse l'état d'une variable booléenne (retourne la valeur 1 si la variable vaut 0, 0 si elle vaut 1)	!(condition)

Tableau 9. Opérateurs logiques

Dans ce qui suit, nous allons étudier les quatre formes d'instructions conditionnelles qui sont:

- Les structures conditionnelles simples
- Les structures conditionnelles composées
- Les structures conditionnelles de choix multiple
- Les branchements

1. Structure conditionnelle simple

Une structure de contrôle conditionnelle est dite à forme simple (réduite) lorsque le traitement dépend d'une condition. Si la condition est évaluée à « vrai », le traitement est exécuté. Dans cette structure la non-satisfaction de la condition ne correspond à aucune action à faire. Le formalisme de cette structure dans un organigramme est comme suit :

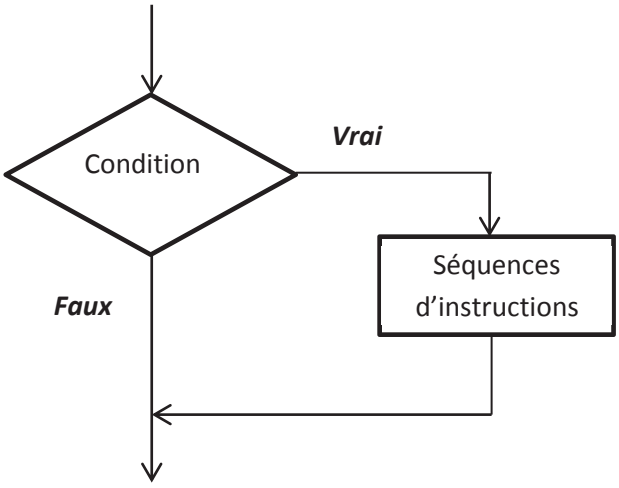

Figure 3. Structure conditionnelle simple

Vocabulaire et syntaxe:

Algorithme	***Langage C***
Si condition **Alors** Instruction 1 Instruction 2 …. Instruction N **FinSi**	**if** (*expression*) */*obligatoirement entre parenthèses */* *{* *Instructions* *}*

Rem : En langage C, si on ne met pas les accolades, le compilateur va considérer uniquement la première instruction comme instruction subordonnée à la structure if.

Exemple :

Calcul de la valeur absolue d'un entier

Solution algorithmique	***Solution en langage C***

Algorithme Absolue
Variables X : entier
Début
Lire (x)
Si (x < 0) **alors**
x← -x
fin si
Ecrire ('la valeur absolus est ',x)
Fin
	#include<stdio.h>
	int main()
	{
	int x;
	printf("Donner x: ");
	scanf("%d",&x);
	if (x<0)
	{
	x= -x;
	}
	printf("la valeur absolue = %d \n", x);
	return 0 ;
	}

2. Structure conditionnelle composée

Une structure de contrôle conditionnelle est dite composée ou à forme alternative lorsque le traitement dépend d'une condition à deux états: Si la condition est évaluée à « vrai », le premier traitement est exécuté, si la condition est évaluée à « faux », le second traitement est exécuté. Le formalisme de cette structure dans un organigramme est comme suit :

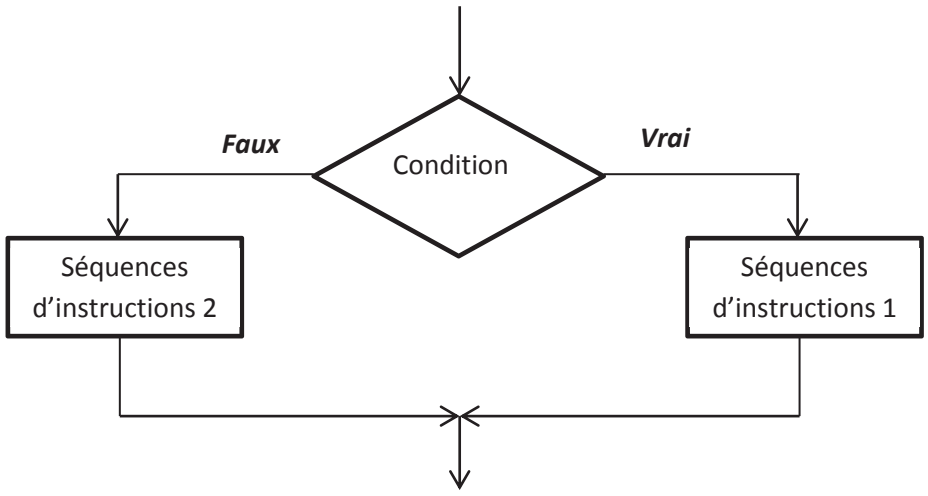

Figure 4. Structure conditionnelle composée

Vocabulaire et syntaxe:

Algorithme	***Langage C***
Si condition **Alors** Instruction 1.1 Instruction 1.2 …. Instruction 1.N **Sinon** Instruction 2.1 Instruction 2.2 …. Instruction 2.M **FinSi**	**if** (*expression*) ***{*** *Bloc d'instructions 1* ***}*** ***else*** ***{*** *Bloc d'instructions 2* ***}***

Les actions qui suivent le *Alors* ou le *Sinon* peuvent être :

- Une simple instruction
- Une suite d'instructions
- Une autre structure alternative
- Une autre structure répétitive

Exemple :

Déterminer si le nombre entier introduit est pair ou impair

Catégorie âge
Algorithme pair_impair **Variables** a : entier **Début** Lire (a) **Si** (a mod 2 = 0) **alors** Ecrire (a,' est pair') **Sinon** Ecrire (a,' est impair') **Fin si** **Fin**

```
#include<stdio.h>
int main()
{
int a;
printf("Donner a: ");
scanf("%d",&a);
if (a%2 == 0) {
printf("%d est pair \n", a);
}
else {
printf("%d est impair \n",a);
}
return 0 ;
}
```

3. Structures conditionnelles imbriquées

Il faut bien comprendre que les blocs d'instructions de ***(SI)*** et de ***(SINON)*** sont des séquences d'instructions. Ces blocs peuvent donc comporter des structures conditionnelles.

Par exemple, la deuxième branche d'une structure conditionnelle peut déboucher sur une nouvelle structure à deux branches, créant ainsi une structure conditionnelle à trois branches.

La syntaxe d'une telle construction est la suivante :

Syntaxe algorithmique

SI condition1 **ALORS**
    Bloc d'instructions 1 ⟵ (Condition 1 vrai)
SINON
    SI condition2 **ALORS**
        Bloc d'instructions 2 ⟵ (condition 1 faux et condition 2 vraie)
    SINON
        Bloc d'instructions 3 ⟵ (condition 1 faux et condition 2 faux)
    FIN SI
FIN SI

On peut itérer ce procédé : le bloc instruction3 peut lui aussi être un conditionnelle et ainsi de suite. Cela permet de créer des structures conditionnelles comportant un nombre (fini) quelconque de branches.

Syntaxe en langage C

En combinant plusieurs structures **if - else** en une expression nous obtenons une structure qui est très courante pour prendre des décisions entre plusieurs alternatives:

```
if (Condition1 )
   {bloc 1}
  else
    if (Condition 2)
       {bloc 2}
    else
     if (Condition 3)
       {bloc 3}
     else
       if (Condition N)
          {bloc N}
       else
          {bloc N+1}
```

Ordre d'exécution

L'exécution de la structure imbriquée précédente est la suivante.

1) si condition1 est réalisée, alors le bloc instruction1 est exécuté.

2) sinon, autrement dit si condition1 n'est pas réalisée :

a) soit condition2 est réalisée, auquel cas le bloc instruction2 est exécuté.

(Cas où condition1 non réalisée et condition2 réalisée)

b) soit condition2 n'est pas réalisée, auquel cas le bloc instruction3 est exécuté.

(Cas où condition1 et condition2 non réalisées)

Les conditions (Condition 1) ... (Condition N) sont évaluées du haut vers le bas jusqu'à ce que l'une d'elles soit différente de zéro. Le bloc d'instructions y lié est alors exécuté et le traitement de la commande est terminé.

Exemple :

Comparaison entre deux entiers :

```
Algorithme Comparaison
Variables A,B: Entier
Début
    Lire (A)
    Lire (B)
    Si (A>B) Alors
        Ecrire ('A est plus grand que B')
    Sinon
        Si (A<B) Alors
            Ecrire ('A est plus petit que B')
        Sinon
            Ecrire ('A égale à B')
        Fin si
    Fin si
Fin
```

```
#include <stdio.h>

main()
{
 int A,B;
 printf("Entrez deux nombres entiers :");
 scanf("%d %d", &A, &B);
 if (A > B)
    printf("%d est plus grand que %d\n", A,B);
 else if (A < B)
    printf("%d est plus petit que %d\n", A,B);
 else
    printf("%d est égal à %d\n", A, B);
 return 0;
}
```

4. Structure conditionnelle de choix multiple

Une structure de contrôle conditionnelle est dite à choix multiple lorsque le traitement dépend de la valeur que prendra le sélecteur, Ce sélecteur doit être de type scalaire (*entier ou caractère*).

Cette structure conditionnelle est appelée aussi ***sélective*** car elle sélectionne entre plusieurs choix à la fois, et non entre deux choix alternatifs (le cas de la structure **Si**..**Sinon**). Le formalisme de cette structure dans un organigramme est comme suit :

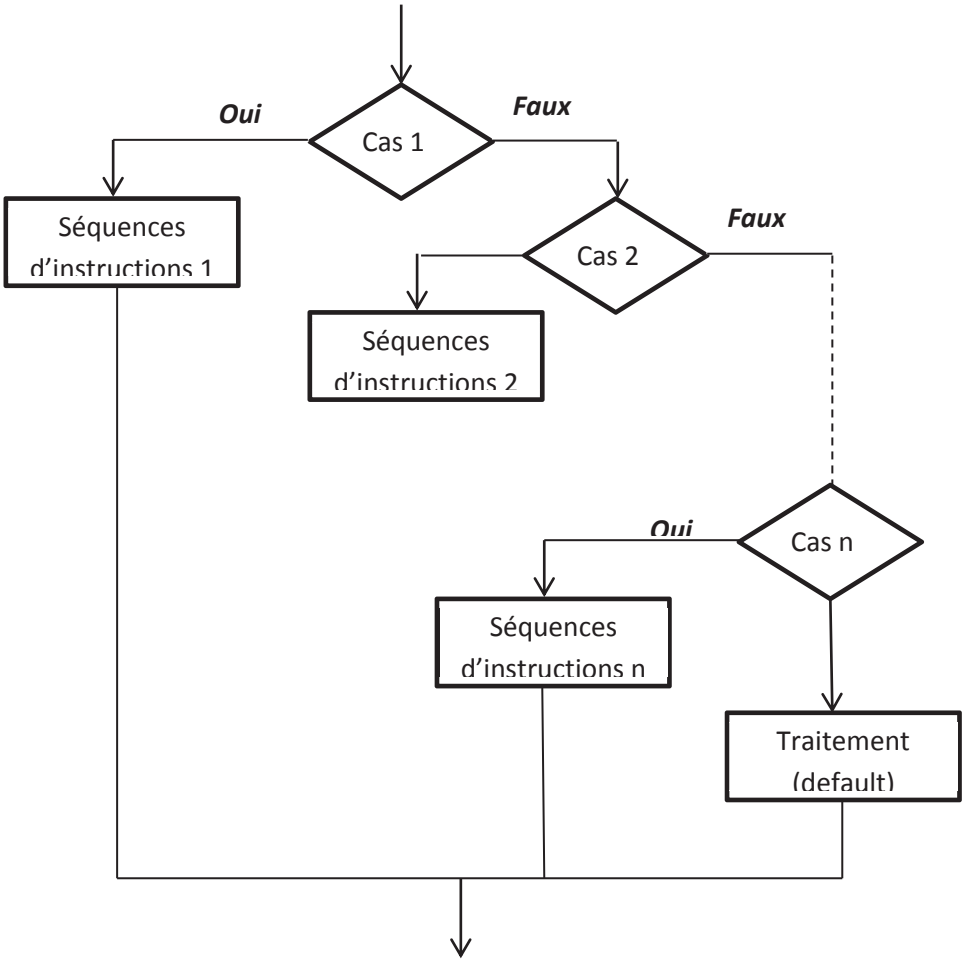

Figure 5. Structure conditionnelle de choix multiple

Vocabulaire et syntaxe:

Algorithme	***Langage C***
SELON (sélecteur) **FAIRE** **Cas <liste de valeurs-1>** : <suite d'action (s)-1> **Cas <liste de valeur-2>** : <suite d'action (s)-2> ……. **SINON** : <suite d'action (s)-n> **FIN SELON**	**switch** (*expression*) *{* **case** *constante-1*: *liste d'instructions 1* **break;** **case** *constante-2*: *liste d'instructions 2* **break;** *...* **case** *constante-n*: *liste d'instructions n* **break;** **default:** *liste d'instructions default* **break;** *}*

Si la valeur de « *expression ou sélecteur»* est égale à l'une des *constantes*, la *liste d'instructions* correspondant est exécutée. Sinon la *liste d'instructions* correspondant à « *Sinon ou default* » est exécutée. Au cas où l'action sinon ou default n'existe pas alors aucune action n'est exécutée.

Important :

- L'instruction *default est facultative.*
- Le mot clé break indique la sortie de la structure conditionnelle. Le mot clé default précède la liste d'instructions qui sera exécutée si l'expression n'est jamais égale à une des valeurs. Si le break est omis, l'exécution continue dans les blocs suivants. Cet état de fait peut d'ailleurs être utilisé judicieusement afin de faire exécuter les mêmes instructions pour différentes valeurs consécutives [13].

```
switch (variable)
{
case 1: instructions 1 exécutées pour variable = 1
case 2: { instructions exécutées pour variable = 1 ou pour variable = 2 }
break;
case 3:{ instructions exécutées pour variable = 3 uniquement }
break;
default:{ instructions exécutées pour toute autre valeur de variable }
}
```

Exemple : Déterminer la nature d'un groupe

Algorithme compte_pers
Variables Nbr_Per : entier
Début
Lire (Nbr_Per)
Selon Nbr_Per **Faire**
Cas 1 : Ecrire ('vous êtes monôme')
Cas 2 : Ecrire ('vous êtes binôme')
Cas 3 : Ecrire ('vous êtes trinôme')
Sinon
Ecrire ('Vous constituez un groupe')
FinSelon
Fin
	#include<stdio.h>
	int main()
	{
	int Nbr_Per;
	printf("DONNER LE NOMBRE DE PERSONNE: ");
	scanf("%d",&Nbr_Per);
	switch (Nbr_Per)
	{
	case 1: printf ("Vous êtes monome \n"); break;
	case 2: printf ("Vous êtes binome \n"); break;
	case 3: printf ("Vous êtes trinome \n"); break;
	default:
	printf ("Vous constituez un groupe \n"); break;
	}
	}

- Lorsque la variable activée est égale à un cas, les instructions qui suivent ce cas seront exécutées jusqu'à ce qu'une instruction break soit atteinte.
- Lorsqu'une instruction break est atteinte, le commutateur se termine et le flux de contrôle passe à la ligne suivante qui suit l'instruction du commutateur.
- Tous les cas ne doivent pas nécessairement contenir une pause. Si aucune pause n'apparaît, le flux de contrôle se répercutera sur les cas suivants jusqu'à ce qu'une rupture soit atteinte.
- Une instruction de commutateur peut avoir un cas par défaut facultatif, qui doit apparaître à la fin du commutateur. Le cas par défaut peut être utilisé pour effectuer une tâche lorsqu'aucun des cas n'est vrai. Aucune pause n'est nécessaire dans le cas par défaut.

5. Le branchement

A priori, dans un programme, les instructions sont exécutées séquentiellement, c'est à dire dans l'ordre où elles apparaissent. Mais, avec les instructions de branchement nous pouvons casser cette règle en permettant à un programme d'interrompre un bloc d'instruction pour sortir ou pour passer à un autre bloc. En langage C, cette idée peut être réalisée à travers les instructions : *break, continu* et *goto.*

5.1. L'instruction *break*

L'instruction *break* permet d'interrompre le déroulement d'une boucle, et passe à la première instruction qui suit la boucle. En cas de boucles imbriquées, break fait sortir de la boucle la plus interne. Par exemple, le programme suivant

```
#include<stdio.h>
int main ()
{
int i;
for (i = 0; i < 5; i++)
{
printf("i = %d\n",i);
if (i == 3)
break;}
printf("valeur de i a la sortie de la boucle = %d\n",i);
}
```

Le programme va sortir de la boucle *for* avec i=3, c'est-à-dire avant l'expiration de la boucle en imprimant :

i = 1

i = 2

i = 3

valeur de i a la sortie de la boucle = *3*

5.2. L'instruction *continue*

L'instruction continue permet de passer directement au tour de boucle suivant, sans exécuter les autres instructions de la boucle [12]. Ainsi le programme :

```
#include<stdio.h>
int main ()
{int i;
for (i = 0; i < 5; i++)
{
if (i == 3)
continue;
printf("i = %d\n",i);
}
printf("valeur de i a la sortie de la boucle = %d\n",i);
}
```

imprime
i = 0
i = 1
i = 2
i = 4
valeur de i a la sortie de la boucle = 5

5.3. L'instruction *goto*

L'instruction *goto* permet d'effectuer un saut jusqu``a l'instruction *etiquette* correspondant. Elle est à proscrire de tout programme C digne de ce nom [12]. Ainsi le programme :

```
#include<stdio.h>
int main ()
{
    int i;
    for (i=1 ;i<=10 ;i++)
    {
    printf ("début de tour %d\n ",i) ;

    if(i==3)goto sortie ;

    }
    sortie: printf ("fin de tour avec i=%d\n",i) ;
            return 0 ;
}
```

```
 imprime :
début de tour 1
début de  tour 2
début de tour 3
fin de tour avec i=3
```

Conclusion

Dans ce chapitre, nous avons présenté les structures conditionnelles, qui vont nous permettre de faire des tests et d'exécuter une ou plusieurs instructions dans un cas, d'autres instructions dans un autre cas. Selon la situation et le positionnement de problème, le développeur peut choisir entre plusieurs structures conditionnelles à savoir : les structures simples, les structures composées, les structures imbriquées, les structures de choix multiples ou les branchements.