Chapitre 3 : Instructions conditionnelles

1. Introduction

Le programme est un ensemble d'instructions. La plupart de ces instructions sont exécutées dans l'ordre dans lequel elles apparaissent. Une fois qu'une instruction est exécutée, elle passe à l'instruction suivante (séquentiel). Les instructions sont souvent séparées par un point-virgule « ; ». Cependant, certaines instructions modifient le déroulement du programme, appelées structures de contrôles. Par exemple : structures conditionnelles, boucles, appels de fonction et les instructions de sauts inconditionnels. La structure conditionnelle se décline en trois types: la structure conditionnelle simple (si alors), la structure conditionnelle complexe (si alors sinon) et la structure à choix multiples (selon).

2. La Structure conditionnelle simple "si alors"

En programmation, nous rencontrons souvent des cas où il faut décider si une instruction doit être exécutée ou non, selon que la condition est vraie ou non. Par exemple, dans une recette de dessert, on retrouve les instructions de la figure : Si vous avez des amandes, ajoutez-les à la recette, ou si vous aimez les citrons, ajoutez-en un peu plus. Le flux d'exécution du programme changera à mesure que l'entrée change. Pour exprimer la condition en programmation, nous utilisons le test si…alors (if…then…), qui est l'instruction conditionnelle la plus simple. Il se compose de deux parties :

- Condition : une expression de type booléen dont la valeur est soit vrai soit faux.
- Bloc d'instructions : exécuté si la condition est vraie, ou ignoré si la condition est fausse.

2.1. Syntaxe

Algorithme	C

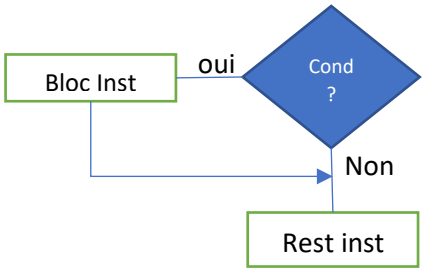
Si Condition **Alors**
  Un bloc d'instruction
FinSi
Le reste des instructions
	if (Condition)
	{
	  Un bloc d'instruction
	}
	Le reste des instructions

Les mots **si**, **alors** et **finSi** ou **fsi** sont des mots réservés dans l'algorithme. Il en est de même pour **if** en **C**.
Dans l'algorithme, la condition est toujours entre les mots si et alors, et en C, elle est toujours entre parenthèses (). Pour construire la condition, nous utilisons des opérations de comparaison (>, <, =, ≠, ...) et des opérations logiques (et, ou, non, ...).
Les instructions qui appartiennent à if en C sont entourées de deux accolades {} qui peuvent être omises si elles ne contiennent qu'une seule instruction. ({} facultatif). Si nous trouvons un ensemble d'instructions après le **if**, et que nous ne trouvons pas les deux accolades, alors, seule la première instruction est liée à la condition, et le reste des instructions sera toujours exécuté et n'ont rien à voir avec la condition. Cependant, s'il y a plus d'une instruction à l'intérieur des deux accolades {}, alors, dans ce cas, les deux accolades sont obligatoires.

Remarque :

- En C, le type booléen est exprimé par un int. Où faux est exprimé par 0 et vrai est n'importe quel nombre autre que 0.
- Il n'y a pas de « ; » après }.

2.2. Algorigramme



2.3. Exécution

Le processus d'exécution de l'instruction conditionnelle est effectué en calculant la condition, dont le résultat est de type logique booléen, si le résultat est correct **vrai**, le bloc d'instruction entre le mot **alors** et **fin si** est exécuté dans l'algorithme, ou entre **{}** en **C**, puis le reste des instructions du programme sont exécutées. Si le résultat est **faux**, les instructions entre **alors** et **finSi** sont ignorées et le reste des instructions du programme sont exécutées directement.

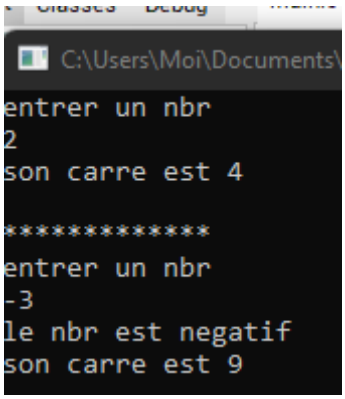
2.4. Exemple

Écrivez un programme qui lit un entier, puis affiche un avertissement s'il est négatif, puis nous montre son carré.

Algorithme	C	l'écran

```
algorithme racine
var x :entier
début
  écrire("entrer un nbr")
  lire(x)
  Si x<0 Alors
   écrire("nbr est négatif")
  FinSi
  écrire("le carré est " , x*x)
fin
```

```
#include <stdio.h>
int main()
{
int x ;
printf("entrer un nbr \n") ;
scanf("%d", &x) ;
if ( x<0 )
{// peut être dispensé
   printf("nbr   est   négatif
   \n") ;
}
   printf("le carré est %d" ,
   x*x) ;
}
```



3. La Structure conditionnelle composée "si alors sinon"

Dans un conditionnel simple, **si** spécifie ce qu'il faut faire si la condition est vraie, mais pas ce qu'il faut faire si elle est fausse. Cependant, il est parfois nécessaire de décider quoi faire dans les deux cas. Par conséquent, vient la structure **si sinon** (**si.. alors.. sinon**...) qui est une extension de **si** simple. L'instruction conditionnelle composée **si sinon** comporte trois parties :

- Condition : une expression de type booléen dont la valeur est vraie ou fausse.
- Premier bloc d'instructions : Il est exécuté si la condition est vraie, ou ignorée si la condition est fausse.
- Deuxième bloc d'instructions : Il est exécuté si la condition est fausse, ou ignorée si la condition est vraie condition vraie.

3.1. Syntaxe

Algorithme	C

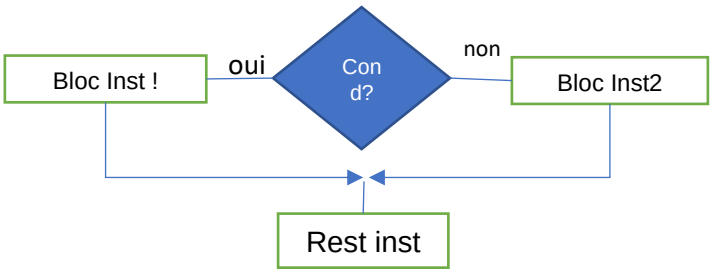
```
Si Condition Alors
  bloc d'instruction 1
sinon
  bloc d'instruction 2
FinSi
Le reste des instructions
```

```
if (Condition)
{
   bloc d'instruction 1
}
else
{
   bloc d'instruction 2
}
Le reste des instructions
```

Le mot **sinon** est un mot réservé dans l'algorithme. Il en va de même pour **else** en C.

{} en C peut être supprimé s'il ne contient qu'une seule instruction. Si nous trouvons un ensemble d'instructions après **if** ou après **else**, et que nous ne trouvons pas les accolades, cela signifie que seule la première instruction concerne **if** ou **else**.

3.2. Algorigramme



3.3. Exécution

L'instruction conditionnelle est exécutée en calculant la condition dont le résultat est de type booléen. Si le résultat est vrai, le premier bloc d'instructions entre **alors** et **sinon** dans l'algorithme est exécuté, ou entre **{}** avant **else** en C, puis le reste des instructions du programme est exécuté. Si le résultat est faux, le deuxième bloc d'instructions entre **sinon** et **finSi** dans l'algorithme, ou entre **{}** après **else** en C, est exécuté, puis le reste des instructions du programme est exécuté.

3.4. Exemple

Écrivez un programme qui calcule la valeur absolue d'un nombre entier, puis l'affiche à l'écran.

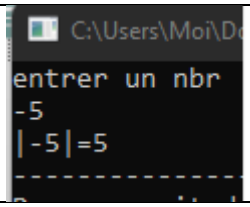
Algorithme

```
algorithme absolue
var x, y :entier
début
  écrire("entrer un nbr")
  lire(x)
  Si x>=0 Alors
    y←x
  sinon
    y←-x
  FinSi
  écrire("|" , x ,"|=",y)
fin
```

C

```
#include <stdio.h>
int main(){
int x, y ;
printf("entrer    un    nbr
\n") ;
scanf("%d", &x) ;
if ( x>=0 )
{ // peut être supprimé
   y=x ;
}
else
{ // peut être supprimé
   y=-x ;
}
printf(("|%d|=%d", x, y)) ;
}
```

l'écran




```
if ( x>=0 )
   y=x ;
else
   y=-x ;
```

3.5. Affectation conditionnelle en C

Si nous avons une variable v prend l'une des valeurs v1 ou v2 selon la condition b, c'est-à-dire :

```
if (b)
  v=v1 ;
else
  v=v2 ;
```

Dans ce cas, l'opération ? : peut être utilisée et sa syntaxe est comme suit :

condition ? expression_vrai : expression_faut

condition est une condition de type booléen

expression_vrai L'expression renvoyée si la condition est vraie.

expression_faut L'expression renvoyée si la condition est fausse

Exemple

```
v=b ? v1 :v2 ;
result= moy>=10 ? "Admis" : "Ajourné" ;
```

3.6. Extension de si sinon

Si sinon peut être utilisé pour tester plusieurs conditions et pour sélectionner le traitement approprié pour chaque cas. Par exemple : pour savoir si l'étudiant est admis ou non, il existe plusieurs cas. Soit il est admis sans compensation, soit il est admis avec compensation, soit il est admis avec des dettes, soit il est ajourné. Pour le savoir, il faut regarder les moyennes des premier et deuxième semestres s1 et s2, la moyenne annuelle MA et le total des crédits obtenus (Crd).

La solution

Algorithme	C

algorithme absolue
var s1, s2, MA:réel
Crd :entier
début
écrire("Entrez les moyennes de premier et deuxième semestres ")
lire(s1,s2)
écrire("Entrez la moyenne annuelle ")
lire(MA)
écrire("Entrez le total des crédits ")
lire(Crd)
Si s1>=10 et s2>=10 **Alors**
écrire("admis sans compensation ")
sinon
Si MA>=10 **Alors**
écrire("admis avec compensation ")
sinon
Si Crd>=45 **Alors**
écrire("admis avec dettes ")
sinon
écrire("ajourné ")
FinSi
FinSi
FinSi
fin
	#include <stdio.h>
	int main(){
	float s1, s2, MA ;
	int Crd ;
	printf("Entrez les moyennes de premier et deuxième semestres \n") ;
	scanf("%f%f", &s1, &s2) ;
	printf("Entrez la moyenne annuelle \n") ;
	scanf("%f", &MA) ;
	printf("Entrez le total des crédits \n") ;
	scanf("%d", &Crd) ;
	if (s1>=10 && s2>=10 **)**
	printf("admis sans compensation ") ;
	else if (MA>=10 **)**
	printf("admis avec compensation ") ;
	else if (Crd>=45 **)**
	printf("admis avec dettes ") ;
	else
	printf("ajourné ") ;
	}

4. La Structure conditionnelle de choix multiple "selon"

Pour choisir une action parmi deux options, nous utilisons l'instruction **si sinon**. Cependant, en présence de plusieurs choix (plus de 2), le **si** imbriqué peut être utilisé, divisant les choix en deux options: la première et le reste des choix, à chaque fois. Cependant, il en résultera des **si** imbriqués aux nombres de choix, ce qui rendra le programme difficile à lire.

Le test **selon** : est un cas particulier de l'instruction **si sinon** imbriqués. Il permet de déterminer le bloc à exécuter en fonction de la valeur de la variable testée. Il est utilisé lorsque nous avons plusieurs sorties et que la condition est testée plusieurs fois, en utilisant toujours la même variable. Avec **selon**, nous pouvons rendre le programme plus lisible. Il se compose de :

- L'expression à tester: est de type entier ou caractère (ou booléen et dans ce cas, il vaut mieux d'utiliser **si sinon**). Habituellement, c'est une variable. Par exemple : age.
- Les valeurs à tester avec le bloc d'instructions de chaque valeur.
- Un bloc d'instructions facultatif s'il n'y a pas de valeur correspondant à la valeur actuelle de la variable (ou de l'expression).

4.1. Syntaxe

Algorithme	C
selon expression **faire** **case** val_1 : bloc d'instruction 1 **case** val_2 : bloc d'instruction 2 … **case** val_n : bloc d'instruction n **sinon** un autre bloc d'instruction **FinSelon** Le reste	**switch (** expression **)** { **case** val_1 : bloc d'instruction 1 **break ;** … **case** val_n : bloc d'instruction n; **break ;** **default:** un autre bloc d'instruction **}** Le reste

Les mots **selon**, **faire**, **sinon**, **case** ou **cas** et **FinSelon** ou **FSelon** sont des mots réservés dans l'algorithme. Il en va de même pour **switch** , **case** et **default** en C.

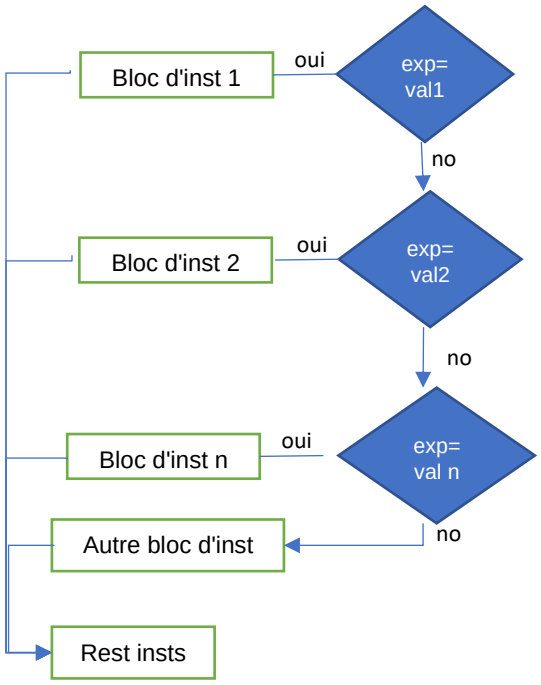
- **expression** : une expression qui est calculée pour obtenir une valeur de type entier, ou caractère. Généralement, il s'agit d'une variable. Dans l'algorithme, il est entre les mots selon et faire, alors qu'en C, il est entre parenthèses ().
- **val_1, ..., val_n** : une valeur ou une constante du même type que l'expression.
- **Bloc d'instruction** : C'est une instruction ou plusieurs instructions qui sont exécutées si la valeur de l'expression correspond à val_i.

Remarque : selon est utilisé à la place de si imbriqué si nous allons tester une seule instruction ou variable, de type entier ou caractère, plusieurs fois avec des valeurs constantes.

4.2. Les règles consternants switch

- Les accolades {} du **switch** et les parenthèses () sont nécessaires et ne peuvent pas être supprimés.
- Chaque valeur val_i doit être différente de l'autre. Par exemple, il est illégal d'écrire le cas 1 deux fois.
- Le cas val_i peut être placé dans n'importe quel ordre. Cependant, il est recommandé de les placer par ordre croissant. Cela augmente la lisibilité du programme.
- Un bloc d'instructions peut être n'importe quel nombre d'instructions et de n'importe quel type.
- L'instruction break ; optionnel. Il est utilisé pour terminer un **switch** directement, déplaçant le flux de programme hors du **switch**.
- Le bloc **default** est facultatif. Si aucun cas val_i ne correspond, le contexte d'exécution est déplacé vers le bloc **default**. Ce doit être le dernier cas.

4.3. Algorigramme



4.4. Exécution

L'instruction selon est exécutée en calculant la valeur d'expression, puis en allant à la valeur correspondante à partir de val_i, et en exécutant son bloc d'instructions, puis en exécutant le reste des instructions du programme. Et dans le cas où il n'y a pas de valeur égale à celle-ci, alors il exécute le bloc d'instructions de sinon, s'il y en a, ensuite le reste des instructions du programme est exécuté.

L'exécution **switch** en C diffère légèrement de selon, en ce que, après que le bloc de val_i a été exécuté, et que l'exécution ne rencontre pas l'instruction **break** ;, elle continuera à exécuter le bloc qui la suit, jusqu'à ce qu'elle rencontre la **break** ; L'exécution passe au reste des instructions en dehors du switch .
Pour que **switch** soit équivalent à selon, il faut ajouter **break** ; à la fin de chaque bloc.
Si deux valeurs ou plus ont le même bloc d'instructions, l'algorithme peut utiliser une virgule. Alors qu'en C nous laissons la première valeur sans instructions ni **break** ;. Si l'on suppose que les valeurs 7 et 9 ont le même traitement, on écrit :

L'algorithme	C

`case 7 ,9 : bloc d'instruction`	`case 7 :`
	`case 9 :`
	`  bloc d'instruction`
	`  break ;`

4.5. Exemple

Écrivez le programme qui lit un entier inférieur à 10, puis ce nombre apparaît en lettre à l'écran en anglais.

Algorithme	C	L'écran

`algorithme conversion`
`var nb :entier`
`début`
`  écrire("entrez un nbr")`
`  lire(nb)`
`  Selon nb faire`
`  case 0 : écrire("zero")`
`  case 1 : écrire("one")`
`  case 2 : écrire("two")`
`  …`
`  case 9 : écrire("nine")`
`  sinon`
`  écrire("not treated")`
`  FinSelon`
`fin`
	`#include <stdio.h>`
	`int main(){`
	`int nb ;`
	`printf("entrez un nbr \n") ;`
	`scanf("%d", &nb ) ;`
	`switch ( nb ) {`
	`case 0 : printf("zero") ;`
	`break ;`
	`case 1 : printf("one") ;`
	`break ;`
	`…`
	`case 9 : printf("nine") ;`
	`break ;`
	`default:`
	`printf("not treated") ;`
	`}`
	`return 0 ;`
	`}`	

5. Les Instructions de Branchement

C'est le processus de déplacement entre les instructions de programme exécutées par le processeur, où il effectue le processus de « saut » à une adresse spécifique au lieu de continuer à exécuter des instructions séquentiellement. Il y a quatre instructions en C qui peuvent modifier inconditionnellement le flux d'exécution d'un programme : break, goto, continue et return.

5.1. Instruction break

Nous l'avons déjà vu avec "switch" qui termine l'instruction "switch", déplaçant le flux vers la première instruction après "switch". Dans le cas d'une instruction "switch" imbriquée, elle ne sort que du "switch" auquel elle est directement subordonnée. Il est également utilisé pour sortir des boucles (leçon suivante). Dans ce cas, "break ; " est généralement à l'intérieur de "if".

Exemple

```
switch (grade){
  case 'A' :
  case 'a' : printf("excellent\n") ;
      break ;
  case 'b' : printf("good\n") ;
  case 'c' : printf("you can do better\n") ;
      break ;
  default : printf("try again\n") ;
}
```

- Si grade contient la lettre a ou A, excellent .

- S'il contient b ‹il apparaîtra good et you can do better
- S'il contient la lettre c, cela montrera seulement you can do better
- • S'il contient un autre caractère, try again.

5.2. Instruction goto

L'exécution du programme s'est transformée vers une instruction nommée. Ce nom "étiquette" est appelé "label" en anglais. Toute instruction peut être nommée avec un ID valide et deux points ":" devant elle.

Syntaxe de label

label : instruction;
où label est un identifiant valide. Tel que:

```
ici : printf("zero") ;
```

Syntaxe d'appel

Pour accéder à cette instruction de n'importe où, nous utilisons la syntaxe suivante :

```
goto label ;
```

où label est le nom de l'instruction. Tel que:
Pour accéder à l'instruction ici de n'importe où, nous utilisons :

```
goto ici ;
```

Observation:

- **case** val_i : et **default** : sont une méthode de nommage spéciale utilisée à l'intérieur d'un **switch**.
- **Goto** peut être utilisé pour répéter des instructions sans avoir besoin de boucles.
- Il est conseillé de ne pas utiliser de **goto** et d'étiquettes, car il est difficile pour l'esprit humain de s'y adapter. Le programme, qui utilise beaucoup goto, est difficile à comprendre et à maintenir.

Exemple

again :
…

goto again **;**

5.3. Instruction continue

Il est utilisé avec des boucles et permet au flux d'être déplacé vers la fin de la boucle et de passer directement à l'itération suivante, sans terminer les instructions de la boucle. Il est, généralement, à l'intérieur d'un **if.**

Syntaxe

```
continue ;
```

5.4. Instruction return

Il est utilisé pour quitter les fonctions (semestre 2) et renvoyer le résultat

Syntaxe

```
return expression ;
```

Exemple

```
return 0 ;
```

Comme celui utilisé à la fin de la fonction main().