**Chapitre 4. Les instructions de contrôle**

Les instructions de contrôle servent à contrôler le déroulement de l’enchaînement des instructions à l’intérieur d’un programme, ces instructions peuvent être des instructions conditionnelles ou itératives.

1. **Les instructions conditionnelles**

Les instructions conditionnelles permettent de réaliser des tests, et suivant le résultat de ces tests, d’exécuter des parties de code différentes.

Le langage C offre deux types d'instructions conditionnelles :

* + l'instruction **if**
  + l'instruction **switch**

## Instruction conditionnelle if

1. **l’instruction if-else Syntaxe :**

**if** ( <*expression*>) <*instruction1*>

**if** ( <*expression*>) <*instruction1*> **else** <*instruction2*>

**Description :**

* + Instruction1 et instruction2 : sont des instructions quelconques, c’est-à-dire :
    - simple,
    - bloc,
    - instruction structurée.
  + Si le résultat de l’expression est « vrai », c'est-à-dire non nul, on exécute l’instruction

<*instruction1>*, dans le cas contraire on exécute l’instruction <*instruction2>* si elle existe. Puis, le programme passe à l’exécution de l’instruction suivante.

**Diagramme syntaxique du if :**



**Expression**

**!=0**

oui

Début de **if**

non

15

\* facultatif

Fin de **if**

instruction2 \*

instruction1

**Exemples :**

int qte\_cmd ;

float prix, remise=0 ;

printf("La quantité commandée : ") ;scanf("%d",&qte\_cmd) ; printf("Prix unitaire : ") ;scanf("%f",&prix) ;

if(qte\_cmd > 100) remise = 0.1;

printf’("Prix à payer : %.2f",prix\*qte\_cmd\*(1 – remise) ) ;

char car ;

printf("Tapez une lettre minuscule non accentuée : "); scanf("%c",&car);

if **(**car == 'a' || car == 'e' || car == 'i' || car == 'o' || car == 'u' || car == 'y'**)** printf("la lettre %c est une voyelle",car);

else

printf("la lettre %c est une consonne",car);

if (qte\_cmd <= qte\_stock)

{

printf("Prix total = %.2f", qte\_cmd\*prix); qte\_stock -= qte\_cmd;

}

else

printf("\aLa quantité commandée est sup. à la quantité en stock ! ");

**Exercice 1 :**

L'utilisateur saisit un caractère, le programme teste s'il s'agit d'une lettre majuscule, si oui il affiche cette lettre en minuscule, sinon il affiche un message d'erreur.

*Solution :*

............................................................................................................................................................................................

.......................................................................................................................................................................................... ..

............................................................................................................................................................................................

............................................................................................................................................................................................

................................................................................................................................... .........................................................

............................................................................................................................................................................................

............................................................................................................................................................................................

............................................................................................................................................................................................

............................................................................................................................................... .............................................

............................................................................................................................................................................................

............................................................................................................................................................................................

............................................................................................................................................................................................

........................................................................................................................................................... .................................

...........................................................................................................................................................................................

............................................................................................................................................................................................

## Exercice 2 :

Quel résultat affiche le programme suivant :

#include<stdio.h> void main()

{

int x = 3 ; if (x < 0) ;

{

x = -x ;

printf("x = %d ", x);

}

if (x = 4) printf("x = %d", x);

}

### Solution :

............................................................................................................................................................

1. **Imbrication d’instructions if**

Il est possible d’imbriquer une instruction **if** à l’intérieur d’une autre.

**Syntaxe :**

**if** (<*expression1>*) **if** (<*expression2>*) <*instruction1>* **else** <*instruction2>*

**Exemple :**

**if** (a < b) **if** (c < b) z = b ; **else** z = a ;

A quel « **if** » se rapporte « **else** » ?

**Règle** : le **else** se rapporte au dernier **if** rencontré auquel un **else** n’a pas encore été attribué sauf si on utilise les accolades.

La bonne mise en page du code ci-dessus est :

**if (a < b)**

**if (c < b)**

**z = b ;**

**else**

**z = a ;**

On peut mettre des { } pour rendre les choses plus claires.

**if (a < b)**

**{**

**if (c < b)**

**z = b ;**

**else**

**}**

**z = a ;**

Si l’on veut que « else » se rapporte au premier « if » :

**if (a < b)**

**{**

**}**

**else**

**if (c < b)**

**z = b ;**

**z = a ;**

1. **L’instruction if-else if**

Il est fréquemment utile d’utiliser une série d’instructions **if-else if** pour répondre à une situation dans laquelle les choix sont multiples.

**Syntaxe :**

**if** (<*expression1>*) <*instruction1>*

**else if** (<*expression2>*) <*instruction2>*

**else if** (<*expression3>*) <*instruction3>*

...

[**else** <*instruction\_n+1>*]

Chaque instruction est associée à une condition et le dernier else, s’il existe, correspond au cas où aucune condition n’est vraie.

#include<stdio.h> void main()

{

char c;

printf("Entrez une lettre non accentuée : "); scanf("%c", &c);

if (c >='A' && c<='Z')

printf("Cette lettre en minuscule est : %c",c+32); else if (c >='a' && c<='z')

printf("Cette lettre en majuscule est : %c",c-32) ;

else

printf("Erreur ! Ce n’est pas une lettre non accentuée") ;

}

**Exemples :**

if ( qte < 1000) taux\_remise = 5;

else if ( qte < 10000)

taux\_remise = 10; else

taux\_remise = 15;

**Exercice 3 :**

Ecrivez un programme qui permet de saisir une valeur de type entière et indiquez à l'utilisateur si celle-ci est positive, négative ou nulle.

*Solution :*

…………………………………………………………………………………………………………………………

…………………………………………………………………………………………………………………………

…………………………………………………………………………………………………………………………

…………………………………………………………………………………………………………………………

…………………………………………………………………………………………………………………………

…………………………………………………………………………………………………………………………

…………………………………………………………………………………………………………………………

…………………………………………………………………………………………………………………………

…………………………………………………………………………………………………………………………

…………………………………………………………………………………………………………………………

…………………………………………………………………………………………………………………………

…………………………………………………………………………………………………………………………

…………………………………………………………………………………………………………………………

…………………………………………………………………………………………………………………………

## Remarque :

Le langage C offre la possibilité de remplacer l’instruction conditionnelle **if-else** dans le cas d’une affectation conditionnelle de variable par l’opérateur conditionnel « **?:** » et qui a l'avantage de pouvoir être intégré dans une expression.

**Syntaxe :**

**expr1 ? expr2 : expr3**

On évalue **expr1**:

* + Si elle n'est pas nulle (donc elle est vraie), alors la valeur de <**expr2>** est fournie comme résultat.
  + Sinon, la valeur de <**expr3**> est fournie comme résultat.

**Exemples :**



int x = 5, y = 4, max; if (x>y)

max=x;

else

max=y;

printf("Le max est : %d",max) ;

int x = 5, y = 4, max; max = (x>y) **?** x **:** y;

printf("Le max est : %d",max) ;

int x = 5, y = 4;

printf("Le max est : %d",(x>y) **?** x **:** y) ;

* 1. **Instruction conditionnelle switch**

L’instruction conditionnelle **swtich** permet de remplacer plusieurs if-else imbriqués lorsqu’il s’agit d’effectuer un choix multiple.

**Sa syntaxe est la suivante :**

**switch** (expression)

{

**case** expression\_const\_1 :

instruction\_1; [**break;**]

**case** expression\_const\_2 :

instruction\_2; [**break;**]

**….**

**case** expression\_const\_n :

instruction\_n; [**break;**]

[**default:**

instruction\_par\_défaut;]

}

**Notes :**

* + - **expression** et **expression\_const\_i** ne peuvent être ni de type réel ni de type chaîne de caractère.
    - **expression\_const\_1,expression\_const\_2,…,expression\_const\_n** doivent obligatoire- ment être distinctes.
    - **break** et **default** sont optionnelles.

## Description :

L’<expression> est évaluée puis le résultat est comparé à chacune des expressions constantes spécifiées après les différentes cases. Si l’<expression> vaut l’<expression\_const\_i>, l’exécution se poursuit par l’<instruction\_i> jusqu’à la fin du switch à moins de rencontrer une instruction break qui permet de la terminer. Si l’<expression> ne correspond à aucune <expression constante> alors l’<instruction\_par\_défaut> qui sera exécuter.

**Diagramme syntaxique de switch :**



Début de **switch**

Evaluer l’expression

expression

==

expression\_const\_1 ?

oui

instruction\_1

non

expression

== non

expression\_const\_2 ?

expression

== non

expression\_const\_n ?

oui

oui

**break ?**

non

**break ?**

non

**break ?** non

oui

oui

oui

Fin de **switch**

instruction\_par\_défaut

instruction\_n

instruction\_2

**Exemples :**



void main()

{

int jour;

printf("Entrez le numéro d’un jour de la semaine (1 à 7) : ");

scanf("%d",&jour);

printf("Le jour %d de la semaine est le ",jour);

if (jour == 1)

printf("DIMANCHE"); else if (jour == 2)

printf("LUNDI"); else if (jour == 3)

printf("MARDI"); else if (jour == 4)

printf("MERCREDI"); else if (jour == 5)

printf("JEUDI"); else if (jour == 6)

printf("VENDREDI"); else if (jour == 7)

printf("SAMEDI");

else

printf("Erreur!");

void main()

{

int jour;

printf("Entrez le numéro d’un jour de la semaine (1 à 7) : "); scanf("%d",&jour);

printf("Le jour %d de la semaine est le ",jour);

**switch (**jour**)**

**{**

**case** 1 : printf("DIMANCHE");

**break;**

**case** 2 : printf("LUNDI");

**break;**

**case** 3 : printf("MARDI");

**break;**

**case** 4 : printf("MERCREDI");

**break;**

**case** 5 : printf("JEUDI");

**break;**

**case** 6 : printf("VENDREDI");

**break;**

**case** 7 : printf("SAMEDI");

**break;**

**default**: printf("Erreur!");

}

}

1. **Instructions itératives**

Les instructions itératives ou boucles sont réalisées à l'aide d'une des trois instructions de contrôlesuivantes : **do-while, while et for**.

* 1. **Instruction do-while <faire-tant que> :**

La boucle *do-while* permet de répéter une instruction ou un bloc d’instructions, un certain nombre de fois, tant qu'une condition est vérifiée.

**Syntaxe :**

**do**

<instruction> /\*ou <bloc d’instructions> \*/

**while** (<expression>);

**Description :**

La boucle **do-while** débute par l’exécution de l’<instruction>, puis l’<expression> est évaluée. Tant qu’elle est VRAIE, l’exécution de l’<instruction> est répétée ; lorsqu’elle devient fausse, le programme quitte la boucle et passe aux instructions suivantes.

**Diagramme syntaxique :**

Début de

**do...while**

oui

**Expression != 0**

non

Fin de

**do...while**

**Instruction**

**Remarques :**

* + - L’<instruction> est exécutée au moins une fois, car le test de l’<expression> s’effectue à la fin de la boucle.
    - Il faut s’assurer que l’<instruction> modifie la valeur de l’<expression> si on veut éviter d’être pris dans une boucle sans fin.

**Exemples :**

void main ()

{

int compteur = 1;

**do**

{

printf("La valeur du compteur vaut : %d\n", compteur); compteur++ ;

}**while (**compteur <=10**);**

printf("La valeur finale du compteur vaut : %d",compteur);

}

#define PI 22/7.0 void main ()

{

float rayon ;

**do**

**{**

printf("Donnez le rayon : ") ;scanf("%f",&rayon); if(rayon < 0) printf("\aLe rayon doit être positif\n") ;

**}while (**rayon < 0**);**

printf("Aire = %.2f",rayon\*rayon\*PI);

}

#include <stdio.h> void main ()

{

char rep ;

**do**

{

….

printf("Vous-voulez continuer (o/n) : ") ; scanf(" %c",&rep); /\* rep = getche(); \*/

}**while (**rep == ‘o’ || rep == ‘O’**);**

}

* 1. **L’instruction while <tant que> :**

L’instruction itérative **while** se comporte comme la boucle **do-while**, à la différence près que l’<expression> est évaluée en début de boucle. Il y a donc possibilité que l’<instruction> à répéter ne soit jamais exécutée, si le premier test n’est pas vérifié.

**Syntaxe :**

**while** (<expression>)

<instruction>; /\*ou <bloc d’instructions>\*/

**Diagramme syntaxique :**

Début de

**while**

**Expression != 0**

non

oui



Fin de

**while**

**Instruction**

## Exemple :

#include <conio.h> void main ()

{

int color = 1 ;

**while** (color <= 15)

**{**

textcolor(color) ;

cprintf("Couleur N° %d\n\r", color++) ;

**}**

}

**Exercice 4 :**

Ecrire un programme qui demande un entier positif n et qui affiche la somme des n premiers entiers.

### Solution :

……………………………………………………………………………………………………………………………

……………………………………………………………………………………………………………………………

……………………………………………………………………………………………………………………………

……………………………………………………………………………………………………………………………

……………………………………………………………………………………………………………………………

……………………………………………………………………………………………………………………………

……………………………………………………………………………………………………………………………

……………………………………………………………………………………………………………………………

……………………………………………………………………………………………………………………………

……………………………………………………………………………………………………………………………

……………………………………………………………………………………………………………………………

……………………………………………………………………………………………………………………………

……………………………………………………………………………………………………………………………

……………………………………………………………………………………………………………………………

……………………………………………………………………………………………………………………………

## Exercice 5 :

Ecrire un programme qui calcule la somme des nombres positifs donnés par l'utilisateur, le programme lira des nombres tant qu'ils seront positifs.

### Solution :

……………………………………………………………………………………………………………………………

……………………………………………………………………………………………………………………………

……………………………………………………………………………………………………………………………

……………………………………………………………………………………………………………………………

……………………………………………………………………………………………………………………………

……………………………………………………………………………………………………………………………

……………………………………………………………………………………………………………………………

……………………………………………………………………………………………………………………………

……………………………………………………………………………………………………………………………

……………………………………………………………………………………………………………………………

……………………………………………………………………………………………………………………………

……………………………………………………………………………………………………………………………

……………………………………………………………………………………………………………………………

……………………………………………………………………………………………………………………………

……………………………………………………………………………………………………………………………

## Instruction itérative for <Pour> :

La syntaxe de la boucle for en C est la suivante :

**for ([**expression\_1**];[**expression\_2**];[**expression\_3**])**

<instruction>; /\*ou <bloc d’instructions>\*/

**Description :**

La boucle for s’utilise avec trois expressions, séparées par des points virgules :

* + - expression\_1 : est évaluée une seule fois, au début de l’exécution de la boucle . Elle sert à initialiser les données de la boucle.
    - expression\_2 : est la condition d’exécution de l’<instruction> à répéter. Elle est évaluée et testée avant chaque parcours de la boucle. Si son résultat est VRAI alors l’<instruction> est exécutée sinon la boucle est terminée.
    - expression\_3 : est évaluée après l’exécution de l’<instruction> à répéter. Elle utilisée pour mettre à jour les données de la boucle.

**Diagramme syntaxique :**



Début de

**for**

Expression\_1

non



**Expression\_2 !=0**

Fin de

**for**

oui

Instruction

Expression\_3

**Remarques :**

* La boucle for est équivalente à la structure suivante :

**expression\_1;**

while (**expression\_2**)

{

instruction ;

**expression\_3**;

}

* Par rapport à la boucle while, la boucle for met en valeur l’<expression\_1> et l’<expression\_3>.
* Expression\_1, expression\_2 et expression\_3 sont facultatives. Si l’<expression\_2> est omise alors la boucle est infinie.
* Quand tout le traitement peut être spécifié dans les trois expressions du for, l’instruction à répéter peut se réduire à une instruction vide.

**Exemples :**

#include <stdio.h> void main ()

{

int m,i ;

printf("Donnez le multiplicande compris entre 1 et 9 : ") ; scanf("%d",&m);

printf("\n\t\t\tTable de multiplication par %d\n",m) ;

**for (**i=1**;** i <= 10 **;** i++**)**

printf("%2d **x** %2d **=** %2d\n",m,i,m\*i) ;

}

**Exercice 6 :**

Ecrire un programme qui affiche la somme des **n** premiers termes d'une progression arithmétique de raison **r** et de premier terme **m**.

Par exemple, si **n** = 7, **r** = 3 et **m** = 23 alors **somme** = 23 + 26 + 29 + 32 + 35 + 38 + 41 **= 224**

### Solution :

……………………………………………………………………………………………………………………………

……………………………………………………………………………………………………………………………

……………………………………………………………………………………………………………………………

……………………………………………………………………………………………………………………………

……………………………………………………………………………………………………………………………

……………………………………………………………………………………………………………………………

……………………………………………………………………………………………………………………………

……………………………………………………………………………………………………………………………

……………………………………………………………………………………………………………………………

……………………………………………………………………………………………………………………………

……………………………………………………………………………………………………………………………

……………………………………………………………………………………………………………………………

……………………………………………………………………………………………………………………………

……………………………………………………………………………………………………………………………

……………………………………………………………………………………………………………………………

……………………………………………………………………………………………………………………………

……………………………………………………………………………………………………………………………

# Remarque sur les instructions itératives

Les différentes boucles peuvent être imbriquées.

**Exemple :**

#include <stdio.h> #include <conio.h> void main ()

{

int m,n ;

**for** (m=1; m <= 9 ; m++)

{

clrscr();

printf("\t\t\tTable de multiplication par %d\n\n",m) ;

**for** (n=1**;** n <= 10 **;** n++**)**

printf("\t\t\t\t%2d **x** %2d **=** %2d\n",m,n,m\*n);

getch();

}

}

Lorsque l’on imbrique des instructions for, il faut veiller à ne pas utiliser le même compteur pour chacune des instructions.

1. **Les instructions de rupture de séquences**

Ces instructions permettent de rompre le déroulement séquentiel d'une suite d'instructions.

* 1. **L’instruction break**

L'instruction **break** provoque le passage à l'instruction qui suit immédiatement le corps de la boucle *while*, *do-while*, *for* ou *switch*.

**Exemples :**

#include <stdio.h> void main()

{

int i = -1;

**do**

{

printf("La valeur du compteur vaut : %d\n", ++i); if(i == 10) **break**;

}while(1) ;

}

#include <stdio.h> void main()

{

int i;

**for**(i=0; ;i++)

{

printf("La valeur du compteur vaut : %d\n", i); if(i == 10) **break**;

}

}

**Remarques :**

* + - L’instruction break ne peut être utilisée que dans le corps d'une boucle ou d'un switch.
    - L’action de break ne s’applique qu’à la boucle la plus intérieure dans laquelle elle se trouve. Elle ne permet pas de sortir de plusieurs boucles imbriquées.
  1. **L’instruction continue**

Elle relance immédiatement la boucle **while, do, for**, dans laquelle elle se trouve.

* pour les boucles **while et do**, la condition d’arrêt est immédiatement réévaluée.
* pour la boucle **for**, on passe à l’évaluation de l’<expression\_3>. C'est pour cette raison que l'équivalence entre le for et le while n'est pas totale.

**Exemple :**

#include <stdio.h> void main()

{

int i;

**for**(i=0; i<=20; i++)

{

if(i%2 == 0) **continue**;

printf("La valeur du compteur vaut : %d\n", i);

}

}

* 1. **L’instruction goto**

Cette instruction permet de se brancher (inconditionnellement) à une étiquette (identificateur) à l’intérieur de la même fonction.

Sa syntaxe est la suivante :

**goto** étiquette ;

**étiquette** :

instruction ;

Et la déclaration d'une étiquette se fait de la manière suivante :

**Exemple :**

void main()

{

int i,j,k;

for(i=1 ;i<=5 ;i++)

for(j=1;j<=5;j++)

for(k=1;k<=5;k++)

{

printf("i = %d; j = %d; k = %d",i,j,k); if (i\*j\*k== 40) **goto** sortie;

}

**sortie :** printf("Fin du programme") ;

}

**Remarque** :

Il est déconseillé de l'utiliser systématiquement, elle n'est vraiment utile que dans des cas très extrêmes.