

La Programmation en langage C

Partie 2 : les instructions de contrôle

La cible : SMI S3

Pr S.ELFILALI

Introduction

Les instructions de contrôle

servent à

- contrôler le déroulement de l'enchaînement des instructions à l'intérieur d'un programme

peuvent être

- des instructions conditionnelles
- des instructions itératives.

Les instructions conditionnelles

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.

C offre deux types d'instructions conditionnelles :

- l'instruction **if**
- l'instruction **switch**

Instruction conditionnelle if

l'instruction if-else

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

Instruction conditionnelle if

Description :

Instruction1 et
instruction2 :

- simple
- bloc
- instruction structurée

Si le résultat de
l'expression est « vrai »

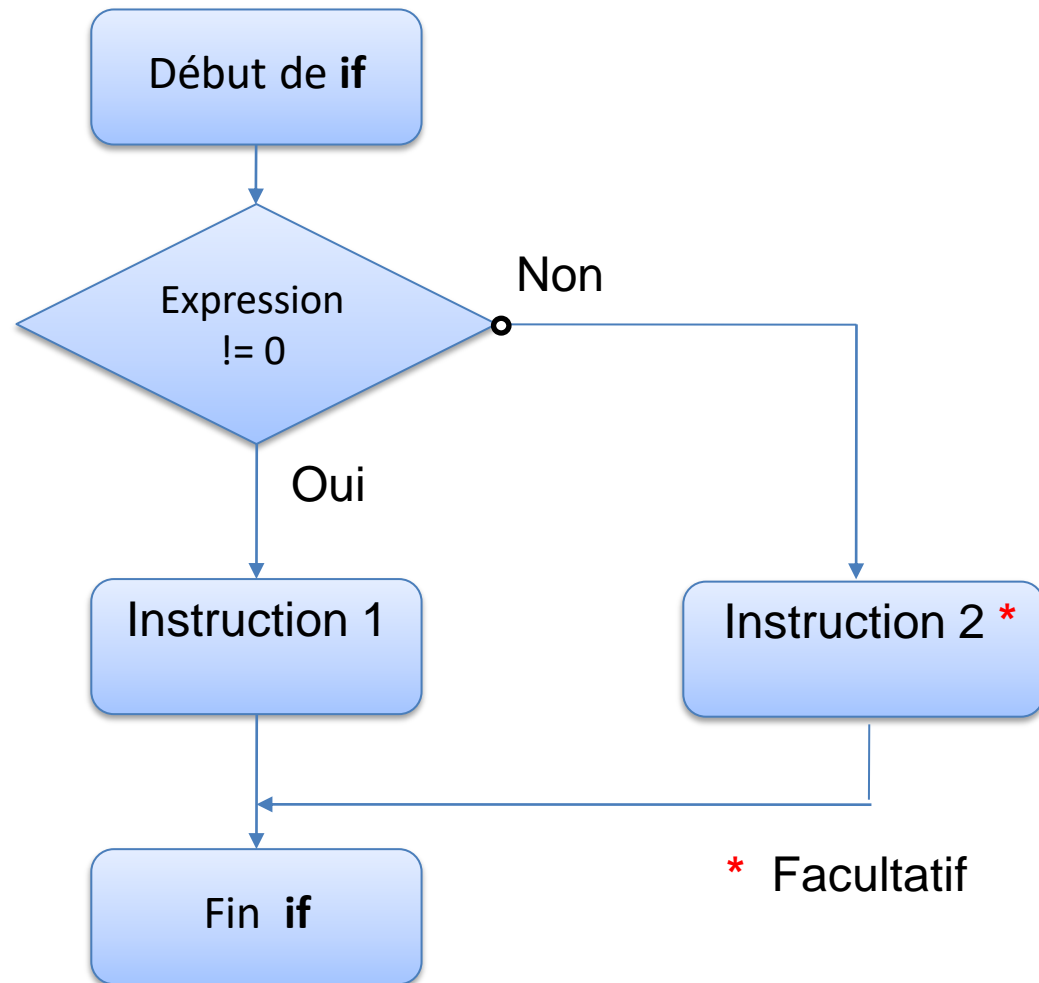
- on exécute l'instruction *<instruction1>*,

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 :



À quoi sert l'instruction if ?

Exemple1 :

```
//declaration des variables
```

```
int qte_cmd ;
```

```
float prix, remise=0 ;
```

```
//lecture des donnees
```

```
printf("La quantité commandée : ") ; scanf("%d",&qte_cmd) ;
```

```
printf("Prix unitaire : ") ; scanf("%f",&prix) ;
```

```
//traitement
```

```
if(qte_cmd > 100) remise = 0.1;
```

```
//affichage des resultats
```

```
printf("Prix à payer : %.2f",prix*qte_cmd*(1 - remise) ) ;
```

Exemple2 :

```
//declaration des variables
```

```
char car ;
```

```
//lecture des donnees
```

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

```
//traitement et affichage des resultats
```

```
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);
```


Exemple3 :

```
//declaration des variables
```

```
.....
```

```
//lecture des donnees
```

```
.....
```

```
.....
```

```
//traitement et affichage des resultats
```

```
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.

..... (tolower, toupper)

Solution :

```
#include <stdio.h>
void main(void)
{
    char ch;

    printf("veuillez saisir un caractere : "); scanf("%c",&ch);

    if(ch==toupper(ch)) printf("%c",tolower(ch).);
    else printf("erreur de saisi ... ");

}
```

```
veuillez saisir un caractere : r
erreur de saisi ...
```

```
veuillez saisir un caractere : R
r
```

Fin séance 5

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 :

.....
.....**x=-3**.....
.....

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 ;
```

Question : 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.

Exemple

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)
{
  if (c < b)
    z = b ;
}
else
  z = a ;
```

L'instruction if-else if

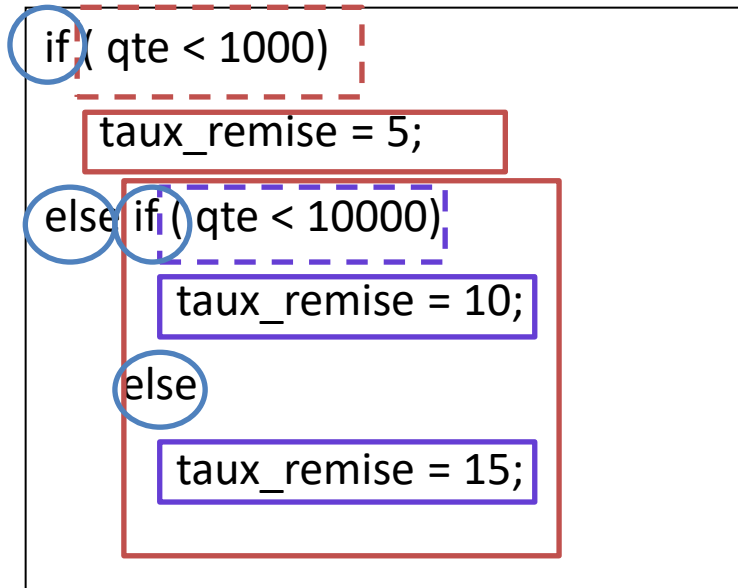
Il est fréquemment utile d'utiliser une série d'instructions **if-else if** pour répondre à une situation dans laquelle les choix sont multiples.

Syntaxe :

```
if (<expression1>) <instruction1>  
else if (<expression2>) <instruction2>  
else if (<expression3>) <instruction3>  
...  
[else    <instruction_n+1>]
```

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

Exemples :



Exemples :

```
#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");
```

```
}
```

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 :

[illegible]

L'opérateur ?:

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 **expr1** est vraie

- Résultat est la valeur de **<expr2>**

Si **expr1** n'est pas
vraie,

- Résultat est la valeur de **<expr3>**

Examples :

```
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) ;
```

Instruction conditionnelle switch

À quoi sert l'instruction switch?

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

Exemples :

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;  
}
```

NB :

expression et
expression_const_i

- ni de type réel
- ni de type chaîne de caractère.

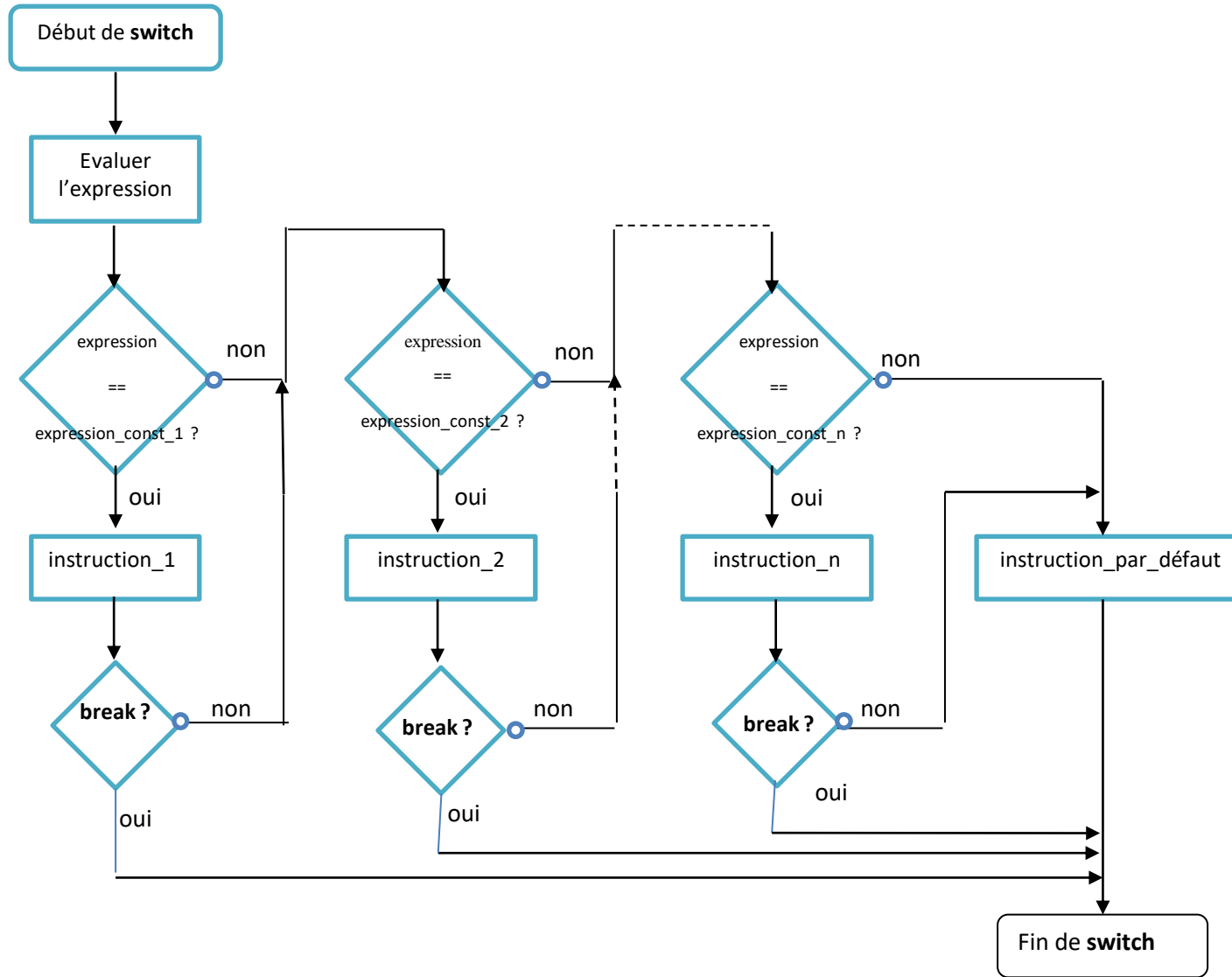
expression_const_i

- doivent obligatoirement être distinctes.

break et default

- sont optionnelles.

Diagramme syntaxique de switch :



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);
    switch (jour)
    {
        case 1 :      printf("DIMANCHE");
                       break;

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

        .....

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

        default: printf("Erreur!");
    }
}
```

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 == 7)
                printf("SAMEDI");
            else
                printf("Erreur!");
}
```

Instructions itératives

Les instructions itératives ou boucles sont réalisées à l'aide d'une des trois instructions de contrôle suivantes :

Instructions itératives

do-while

while

for

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.

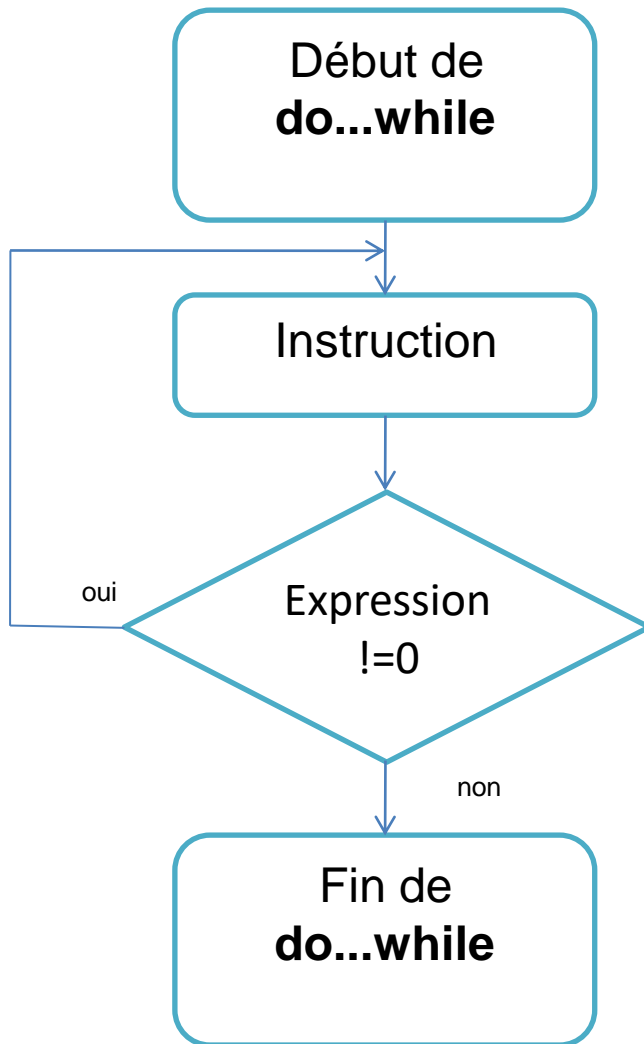
Syntaxe

do

<instruction> /*ou <bloc d'instructions> */

while (<expression>);

Diagramme syntaxique do-while :



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);
}
```

Exemples :

```
#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);
}
```


Exemples :

```
#include <stdio.h>
void main ()
{
    char rep ;
    do
    {
        ....
        printf("Vous-voulez continuer (o/n) : " ) ;
        scanf(" %c",&rep); /* rep = getche(); */

    }while (rep == 'o' || rep == 'O');
}
```

L'instruction while <tant que>

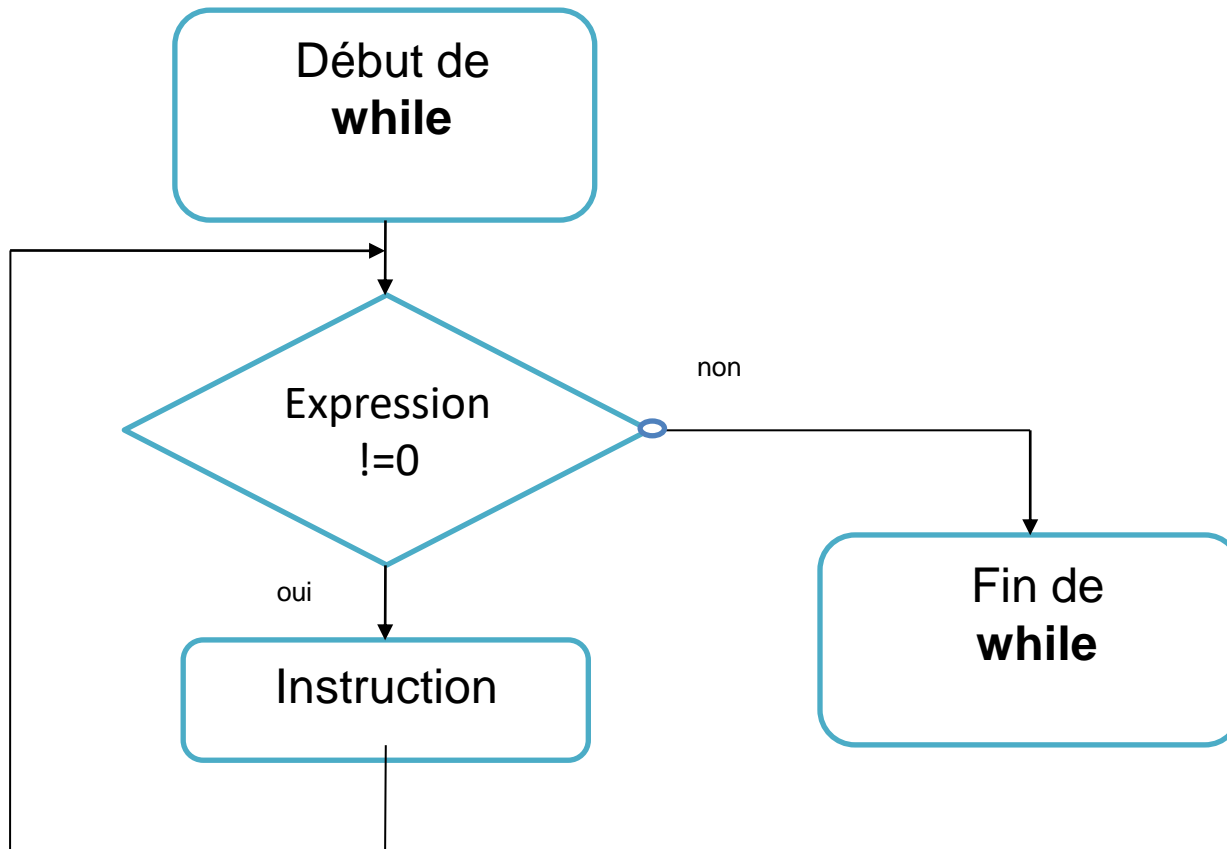
L'instruction **while** se comporte comme la boucle **do-while**, à la différence près que l'<expression> est évaluée en début de boucle. Il y a donc possibilité que l'<instruction> à répéter ne soit jamais exécutée, si le premier test n'est pas vérifié.

Syntaxe

While (<expression>)

<instruction> ; /*ou <bloc d'instructions> */

Diagramme syntaxique de while :



Exemples :

```
#include <conio.h>
void main ()
{
    int color = 1;
    while (color <= 15)
    {
        textcolor(color);
        cprintf("Couleur N° %d\n\r", color++);
    }
}
```

Exercices :

Exercice 4 :

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

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int s=0, n,i;
    printf("donner un entier
positif : "); scanf("%d",&n);
    i=0;
    do
    {
        s+=i;
        i++;
    } while (i <= n);

    printf("la somme est :%d
",s);

    return 0;
}
```

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int s=0, n,i;
    printf("donner un entier
positif : ");
    scanf("%d",&n);
    i=0;

    while (i <= n)
    {
        s+=i;
        i++;
    }
    printf("la somme est :%d
",s);

    return 0;
}
```

```
#include <stdlib.h>
#include <stdlib.h>

int main()
{
    int s=0, n,i;
    printf("donner un entier
positif : ");
    scanf("%d",&n);

    for(i=0;i<=n;i++)
        s+=i;

    printf("la somme est :%d
",s);

    return 0;
}
```

Exercices :

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 :

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int s=0, n;
    do
    {
        printf("donner un entier positif : "); scanf("%d",&n);

        if(n>=0)    s+=n;

    } while (n >= 0);
    printf("la somme est :%d ",s);

    return 0;
}
```


Instruction itérative for <Pour> :

Syntaxe

```
for ([expression_1];[expression_2];[expression_3])  
<instruction>; /*ou <bloc d'instructions>*/
```

Description :

expression_1 : est évaluée une seule fois. 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 est utilisée pour mettre à jour les données de la boucle.

Instruction itérative for <Pour> :

```
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.

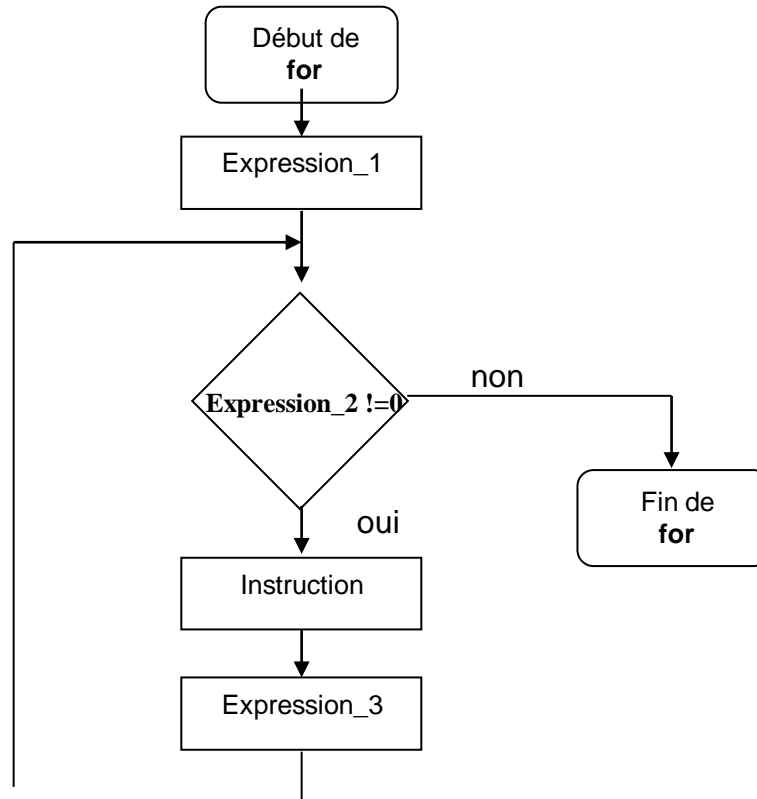
expression_2 :

- 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



Instruction itérative for <Pour> :

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) ;
}
```

Exercices :

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

$$\text{somme} = 23 + 26 + 29 + 32 + 35 + 38 + 41 = 224$$

$$= (m+0*r)+(m+1*r)+(m+2*r) + (m+3*r) + \dots (m+n*r)$$

Solution :

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <windows.h>
void main()
{
    int i,n,m,r,s;

    printf("donner le premier terme de la suite : ");scanf("%d",&m);
    printf("donner la raison de la suite : ");scanf("%d",&r);
    printf("donner le nombre de n elements de la suite : ");scanf("%d",&n);

    s=0;
    for (i=0;i<n;i++)

        s=s+(m+i*r);

    printf("la somme de %d premier terme de la suite de la raison %d et du premier terme %d est : %d ", n,r,m,s);
    getch();
}
```

Remarque sur les instructions itératives

Les différentes boucles peuvent être imbriquées.

```
#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.

Les instructions de rupture de séquences

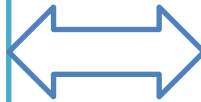
Ces instructions permettent de rompre le déroulement séquentiel d'une suite d'instructions.

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*.

```
#include <stdio.h>

void main()
{
    int i;
    for(i=0; ;i++)
    {
        printf("La valeur du compteur vaut :
%d\n", i);
        if(i == 10) break;
    }
}
```



```
#include <stdio.h>

void main()
{
    int i = -1;
    do
    {
        printf("La valeur du compteur vaut :
%d\n", ++i);
        if(i == 10) break;
    } while(1) ;
}
```

L'instruction break

Que se passe -t-il avec le code suivant?

```
#include <stdio.h>
void main()
{
    int i;
    for(i=0;i<=20 ;i++)
    {
        printf("La valeur du compteur vaut :
%d\n", i);
        if(i == 10) break;
    }
}
```

L'instruction 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.

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 ;
```

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

```
étiquette :  
    instruction ;
```

Examples :

```
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\t j = %d\t k = %d \n",i,j,k);

                if (i*j*k== 40) goto sortie;
            }
sortie : printf("Fin du programme") ;
}
```