Les structures de contrôle en C

Alternative: **`if-else`**

Choix Multiple: **`switch-case`**

Itérations: **`for, while, do-while`**

Rupture de Contrôle: **`break, continue, return`**

… `goto`

Les structures de contrôle en C

Les décisions - if then else

Pas de then en C

Le bloc " **else** " est optionnel.

```
if (expression booléenne vraie)
    {
      BLOC 1 D'INSTRUCTIONS
    }
else
    {
      BLOC 2 D'INSTRUCTIONS
    }
```

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

* Tout ce qui est 0 ('\0' 0 0.0000 NULL) est faux
* Tout ce qui est != de 0 (1 '0' 0.0001 1.34) est vrai

```
if(32)
   printf("ceci sera toujours affiche\n");
if(0)
   printf("ceci ne sera jamais affiche\n");
```

Exemples :

```
if (i < 10) i++;
```

La variable i ne sera incrémentée que si elle a une valeur inférieure à 10.

```
if (i == 10) i++;
```

== et pas =

La variable i ne sera incrémentée que si elle est égale à 10.

```
if (!recu) printf ("rien reçu\n");
```

Le message "rien reçu" est affiché si recu vaut zéro.

```
if ((!recu) && (i < 10)) i++;
```

i ne sera incrémentée que si recu vaut zéro et i<10.

Si plusieurs instructions, il faut les mettre entre accolades.

```
if ((!recu) && (i < 10) && (n!=0) ){
   i++;
   moy = som/n;
   printf(" la valeur de i =%d  et moy=%f\n", i,moy) ;
 }
else {
   printf ("erreur \n");
    i = i +2;                    // i +=2 ;
}
```

```
if(delta != 0)   =   if(delta)

if(delta == 0)   =   if(!delta)
```

Attention!

- Ne pas confondre = = (opérateur logique d'égalité) et = (opérateur d'affectation)

```
#include <stdio.h>

int     main(void)
{
        int     i = 0;

        if(i = 0)       /* ici affectation */
                printf("i = zero\n");
        else
                printf("Quand i != de zero\n");

        return 0;
}
```

Quand i != de zero

if emboîtés

- else est associé avec le if le plus proche

```
int i = 100;

if(i > 0)
        if(i > 1000)
                printf("i > 1000\n");
        else
                printf("i is reasonable\n");
```

i is reasonable

```
int i = 100;

if(i > 0) {
        if(i > 1000)
                printf(" i > 1000 \n");
} else
        printf("i is negative\n");
```

Les itérations – for

```
for( init ; test; increment)
  {
   /* corps de for */
  }
```

```
int i,j;

for (i = 0; i <3; i++) {
        printf ( "i = %d\n", i);
}
for(j = 5; j > 0; j- -)
        printf("j = %d\n", j);
```

```
i = 0
i = 1
i = 2
j = 5
j = 4
j = 3
j = 2
j = 1
```

```
double angle;

for(angle = 0.0; angle < 3.14159; angle += 0.2)
        printf("sine of %.1lf is %.2lf\n",angle, sin(angle));
```

```
int i, j, k;

for(i = 0, j = 2, k = -1; (i < 20) &&(j==2); i++, k--)
```

```
for(   ;   ;   )
{
   .............;     /* bloc d'instructions */
   .............;
   .............;
}
```

est une boucle infinie (répétition infinie du bloc d'instructions).

(Boucles)

LA BOUCLE TANT QUE ... FAIRE ...

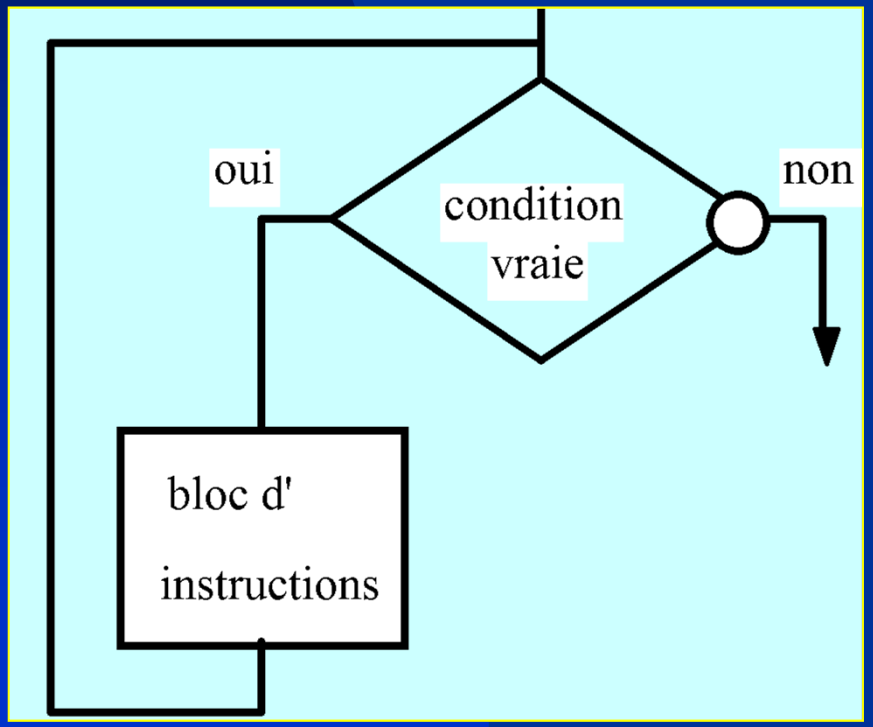
Boucle pré-testée

Il s'agit de l'instruction **while** :

tant que (expression vraie)
faire{BLOC D'INSTRUCTIONS}

tant que, pas jusqu'à ce que!

Organigramme:



Syntaxe en C:

```
while (expression)
{
  .....;        /* bloc d'instructions */
  .....;
  .....;
}
```

Le test se fait **d'abord,** le bloc d'instructions n'est pas forcément exécuté.

Rq: les {} ne sont pas nécessaires lorsque le bloc ne comporte qu'une seule instruction.

Exemple

```
i=1;
while(i<5)
{
  printf("Intérieur %d\n",i);
  i++;
}
printf("Extérieur %d\n",i);
```

itération

```
Intérieur   1
Intérieur   2
Intérieur   3
Intérieur   4
Extérieur   5
```

tant que, pas jusqu'à ce que!

```
int j = 5;

printf("start\n");
while(j == 0)
        printf("j = %d\n", j--);
printf("end\n");
```

start
end

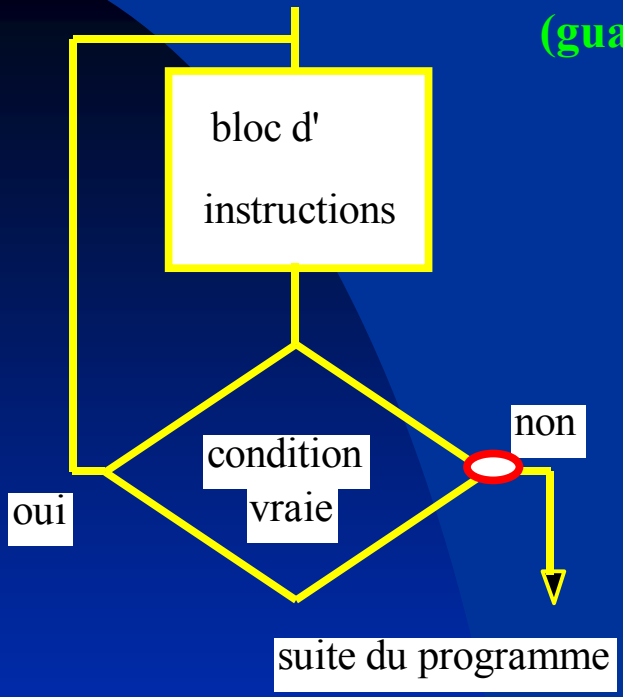
```
i=1;
while(i<5);
{
  printf("Intérieur %d\n",i);
  i++;
}
```

"tant que l'expression est vraie attendre".

(Boucles)

do while = REPETER … tant que

(guarantit l'exécution au moins une fois)




```
do
{
  .....;      /* bloc d'instructions */
  .....;
}
while (expression);
```

```
int j = 5;
do
   printf("j = %i\n", j--);
while(j > 0);
printf("stop\n");
```

```
j = 5
j = 4
j = 3
j = 2
j = 1
stop
```

```
i=1;
do
{
  printf("i=%d ",i);
  i++;
}while(i<0);
printf("stop\n");
```

```
i = 1
stop
```

switch = AU CAS OU ... FAIRE ...

```
switch(variable de type char ou int)     /* au cas où la variable vaut:  */
{
 case  valeur1: .....;          /* cette valeur1(étiquette): exécuter ce bloc d'instructions.*/
                .......;
               break;           /* L'instruction d'échappement break;
                                   permet de quitter la boucle ou l'aiguillage le plus proche.
                                */

 case valeur2:.........;         /* cette valeur2:  exécuter ce bloc d'instructions.*/
                .........;
               break;
        .
        .                       /* etc ...*/
        .
        default:  .......;      /* aucune des valeurs précédentes: exécuter ce bloc
                  .........;        d'instructions, pas de "break" ici.*/
}
```

Le bloc "default" n'est pas obligatoire. valeur1, valeur2, …. doivent être des expressions constantes. L'instruction switch correspond à une cascade d'instructions if ...else

Cette instruction est commode pour les "menus":

```
char choix;
printf("SAISIE TAPER 1\n");
printf("AFFICHAGE TAPER 2\n");
printf("POUR SORTIR TAPER S\n");
printf("\nVOTRE CHOIX: ");
choix = getchar();
switch(choix)
{
 case '1':  .......;
            break;

 case '2':  ......;
            break;

 case 'S':   printf("\nFIN DU PROGRAMME ....");
             break;

 default;  printf("\nCE CHOIX N'EST PAS PREVU ");  /* pas de break ici */
}
```

```
int choix;


scanf(" %d ", &choix);
switch(choix)
{
 case 1: …
```

!

```
float f;

switch(f) {
 case 2:
    ...
```

```
switch(i) {
 case 2 * j:
    ....
```

Instructions d'échappement

Pour rompre le déroulement séquentiel d'une suite d'instructions

Break;

```
int i, j=1;
char a;
for (i = -10; i <= 10; i++){

  while(j!=0)   /* boucle infinie */
  {
    a=getchar();
    if(a= ='x')
     break;
  }
}
```

Si x est tapée au clavier

Continue;

```
for (i = -10; i <= 10; i++)
{
    if (i == 0)
      continue;
            // pour éviter la division par zéro
      printf(" %f", 1 / i);
}
```

return (expression);
permet de sortir de la fonction qui la contient

exit (expression); **La fonction est interrompu.**
expression : un entier indiquant le code de terminaison du processus

goto étiquette

```
#include <stdio.h>
void main()
{
   int i, j;
    for (i=0; i < 10; i++)
     for (j=0; j < 4; j++) {
       if ( (i*j) == 10)
         goto trouve;
         printf("i*j != 10.\n");
       }
   trouve:
   printf("i*j =%d * %d = %d== 10.\n",i,j,i*j);
}
```