

## COURS 2

BOUCLES  
EN C

PARTIE



BOUCLE FOR

# RÉPÉTER LES MÊMES INSTRUCTIONS UN CERTAIN NOMBRE DE FOIS

Exemple : Afficher 100 fois "Je ne parle pas français."



(épisode 377 des Simpson)

# RÉPÉTER UN CERTAIN NOMBRE DE FOIS BOUCLE FOR (VERSION SIMPLE)

Syntaxe	<div>① for ( int i = 1 ; i &lt;= nb-repetitions ; i++ ) { Instructions à répéter ③ } ② ①</div>
---------	--

Effet: Répète les instructions "nb-repetitions" fois

Exemple complet:

(réponse au slide  
précédent)

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

int main() {
    for (int i = 1 ; i <= 100 ; i++) {
        printf("Je ne parle pas français.\n");
    }
    return EXIT_SUCCESS;
}
```

Remarques:

- ① nom de variable que vous voulez (souvent i, j ou k)
- ② nombre, variable ou expression que vous voulez
- ③ pour des questions de visibilité, on indente ces instructions

# RÉPÉTER UN CERTAIN NOMBRE DE FOIS BOUCLE FOR (VERSION SIMPLE)

Syntaxe

```
for ( int i = 1 ; i <= nb-repetitions ; i++ ) {  
    Instructions  
    à  
    répéter  
}
```

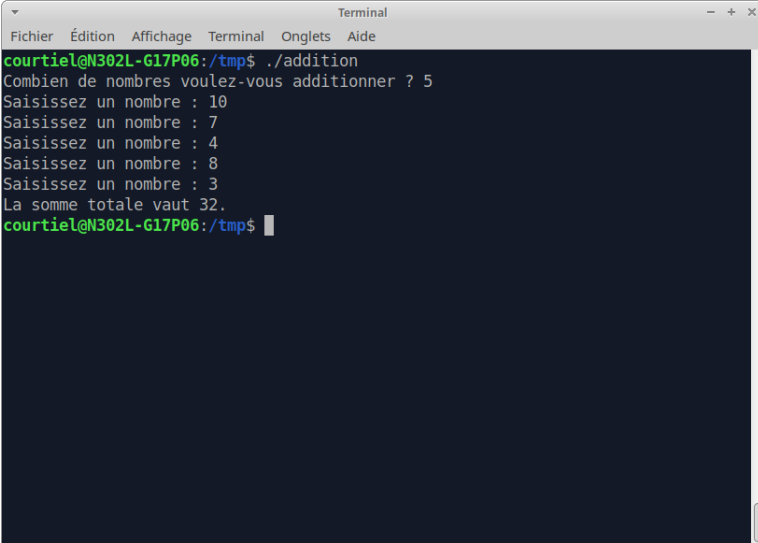
Exemple suivant : Demander un nombre et écrire autant de fois la phrase "Je ne parle pas français."

```
#include <stdio.h>  
#include <stdlib.h>  
  
int main() {  
  
    printf("Saisissez le nombre de lignes à écrire : ");  
    int nombre_lignes;  
    scanf("%d", &nombre_lignes);  
  
    for (int i = 1 ; i <= nombre_lignes ; i++) {  
        printf("Je ne parle pas français.\n");  
    }  
    return EXIT_SUCCESS;  
}
```

# EXEMPLE + ÉVOLUÉ : SOMME DE NOMBRES

On veut un programme qui fait l'addition de nombres saisis par l'utilisateur.

L'utilisateur indiquera au préalable le nombre de nombres qu'il souhaite additionner.

A screenshot of a terminal window titled "Terminal". The window has a menu bar with "Fichier", "Édition", "Affichage", "Terminal", "Onglets", and "Aide". The terminal content shows a user running a program named "addition". The program prompts the user for the number of numbers to add (5), then prompts for each number (10, 7, 4, 8, 3), and finally displays the total sum (32).

```
courtiel@N302L-G17P06:/tmp$ ./addition
Combien de nombres voulez-vous additionner ? 5
Saisissez un nombre : 10
Saisissez un nombre : 7
Saisissez un nombre : 4
Saisissez un nombre : 8
Saisissez un nombre : 3
La somme totale vaut 32.
courtiel@N302L-G17P06:/tmp$
```

# EXEMPLE + ÉVOLUÉ : SOMME DE NOMBRES

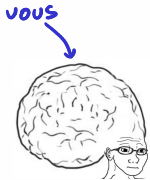
Mais comment résoudre ce problème de programmation?

Principe général souvent vrai	Demandons-nous comment nous, humains, arrivons à résoudre le problème
----------------------------------	--

Ici, comment calculerions-nous

$$10 + 7 + 4 + 8 + 3$$

?

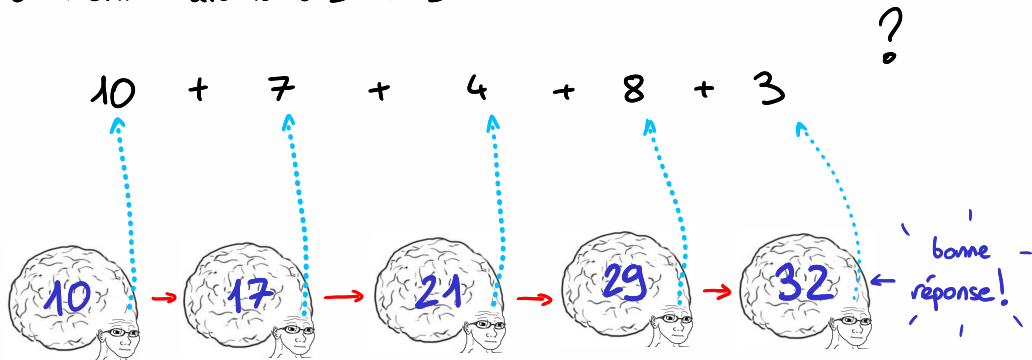


# EXEMPLE + ÉVOLUÉ : SOMME DE NOMBRES

Mais comment résoudre ce problème de programmation?

Principe général souvent vrai	Demandons-nous comment nous, humains, arrivons à résoudre le problème
----------------------------------	--

Ici, comment calculerions-nous



C'est comme si on répétait " nombre en mémoire ← nombre en mémoire + nombre que je vois "  
→ Essayons de le coder.



# EXEMPLE + ÉVOLUÉ : SOMME DE NOMBRES

## La réponse

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

int main() {

    printf("Combien de nombres voulez-vous additionner ? ");
    int nb_termes;
    scanf("%d",&nb_termes);

    int entier_saisi; // Cette variable va successivement contenir chaque entier que l'utilisateur rentrera
    int somme; // Cette variable contiendra la somme des nombres qui ont été saisi jusqu'à présent

    somme = 0; // On initialise cette variable à 0.

    for ( int i = 1 ; i <= nb_termes ; i++ ){

        printf("Saisissez un nombre : ");
        scanf("%d",&entier_saisi);

        somme = somme + entier_saisi;

    }

    printf("La somme totale vaut %d.\n",somme);

    return EXIT_SUCCESS;
}
```

Structure à retenir!

# LA MÉMOIRE LORS D'UNE BOUCLE FOR

Que se passe-t-il quand la boucle suivante est exécutée ?

CODE

```
for (int i = 1 ; i <= 100 ; i++) {  
    // Instructions qui ne modifient pas i  
}
```

EN MÉMOIRE

i     ~~1~~ ~~2~~ ~~3~~ ~~4~~ ... ~~98~~ ~~99~~ 100

- Dans l'ordre :
- ① Une variable  $i$  est déclarée
  - ② Elle est initialisée à 1
  - ③ On exécute les instructions à l'intérieur de la boucle
  - ④ On augmente  $i$  de 1
  - ⑤ Si  $i$  est plus petit ou égal à 100,  
on revient au point ③
- Sinon, l'exécution de la boucle est terminée.

En résumé, une variable  $i$  va de 1 à 100 en augmentant de 1 à chaque passage

# LA MÉMOIRE LORS D'UNE BOUCLE FOR

Que se passe-t-il quand la boucle suivante est exécutée ?

CODE

```
for (int i = 1 ; i <= 100 ; i++) {  
    // Instructions qui ne modifient pas i  
}
```

On peut utiliser le contenu  
de cette variable dans la boucle !

EN MÉMOIRE

i    1 2 3 4 ... 98 99 100

Exemple :

```
int carre;  
for (int k = 1 ; k <= 9 ; k++) {  
    carre = k * k;  
    printf("Le carré de %d est %d.\n", k, carre);  
}
```

```
Terminal  
Fichier  Édition  Affichage  Terminal  Onglets  Aide  
courtiel@N302L-G17P06:/tmp$ ./carre  
Le carré de 1 est 1.  
Le carré de 2 est 4.  
Le carré de 3 est 9.  
Le carré de 4 est 16.  
Le carré de 5 est 25.  
Le carré de 6 est 36.  
Le carré de 7 est 49.  
Le carré de 8 est 64.  
Le carré de 9 est 81.  
courtiel@N302L-G17P06:/tmp$
```

# STRUCTURE PLUS GÉNÉRALE D'UNE BOUCLE FOR

Boucle qui fait varier  $i$  d'un entier arbitraire  $debut$  à un autre  $fin$  :

CODE

```
for (int  $i = debut$  ;  $i \leq fin$  ;  $\underline{i++}$ ) {  
    // Instructions qui ne modifient pas  $i$  ↑  
}
```

(on peut aussi faire " $i = i + 2$ " ou " $i = i + 3$ " ou " $i--$ " ou...)

EN MÉMOIRE

$i$

<del><math>debut</math></del>	<del><math>debut + 1</math></del>	<del><math>debut + 2</math></del>	...	<del><math>fin - 1</math></del>	$fin$
-------------------------------	-----------------------------------	-----------------------------------	-----	---------------------------------	-------

- Dans l'ordre :
- ① Une variable  $i$  est déclarée
  - ② Elle est initialisée à  $debut$
  - ③ On exécute les instructions à l'intérieur de la boucle
  - ④ On augmente  $i$  de 1
  - ⑤ Si  $i$  est plus petit ou égal à  $fin$ , on revient au point ③
- Sinon, l'exécution de la boucle est terminée.

# STRUCTURE PLUS GÉNÉRALE D'UNE BOUCLE FOR

Boucle qui fait varier  $i$  d'un entier arbitraire **debut** à un autre **fin** :

CODE

```
for (int i = debut ; i <= fin ; i++) {
```

// Instructions qui ne modifient pas  $i$

```
}
```

(on peut aussi faire " $i = i + 2$ " ou " $i = i + 3$ " ou " $i--$ " ou...)

EN MÉMOIRE

$i$

~~debut~~ ~~debut+1~~ ~~debut+2~~ ... ~~fin-1~~ ~~fin~~

Exemple :

```
int carre;  
for (int k = -3; k <= 8; k++) {  
    carre = k * k;  
    printf("Le carré de %d est %d.\n", k, carre);  
}
```



```
Terminal  
Fichier  Édition  Affichage  Terminal  Onglets  Aide  
courtiel@N302L-G17P06:/tmp$ ./carre2  
Le carré de -3 est 9.  
Le carré de -2 est 4.  
Le carré de -1 est 1.  
Le carré de 0 est 0.  
Le carré de 1 est 1.  
Le carré de 2 est 4.  
Le carré de 3 est 9.  
Le carré de 4 est 16.  
Le carré de 5 est 25.  
Le carré de 6 est 36.  
Le carré de 7 est 49.  
Le carré de 8 est 64.  
courtiel@N302L-G17P06:/tmp$
```

PARTIE 

BOUCLE WHILE

# BOUCLE WHILE (TANT QUE)

Si on ne connaît pas le nombre de répétitions qu'on souhaite faire, il faut utiliser une boucle "while"

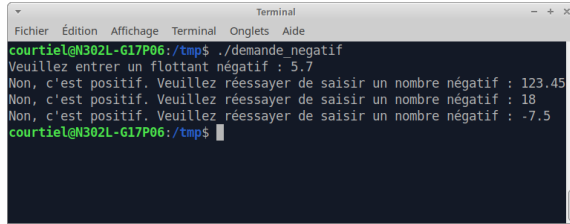
Exemple : On veut écrire un programme qui demande un flottant négatif à l'utilisateur. S'il rentre un flottant positif, on lui redemandera de saisir un nombre jusqu'à qu'il soit négatif.

## CODE

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

int main() {
    float x;
    printf("Veuillez entrer un flottant négatif : ");
    scanf("%f",&x);
    while (x > 0) {
        printf("Non, c'est positif. ");
        printf("Veuillez réessayer de saisir un nombre négatif : ");
        scanf("%f",&x);
    }
    return EXIT_SUCCESS;
}
```

## RÉSULTAT



```
Terminal
Fichier  Édition  Affichage  Terminal  Onglets  Aide
courtiel@N302L-G17P06:/tmp$ ./demande_negatif
Veuillez entrer un flottant négatif : 5.7
Non, c'est positif. Veuillez réessayer de saisir un nombre négatif : 123.45
Non, c'est positif. Veuillez réessayer de saisir un nombre négatif : 18
Non, c'est positif. Veuillez réessayer de saisir un nombre négatif : -7.5
courtiel@N302L-G17P06:/tmp$
```

# BOUCLE WHILE (TANT QUE)

Si on ne connaît pas le nombre de répétitions qu'on souhaite faire, il faut utiliser une boucle "while"

Syntaxe

```
while ( condition ) {  
    Instructions qui  
    se répètent  
}
```

Effets:

- ① On teste si la condition est vraie.
- ② Si oui, on exécute les instructions et on revient au point ①
- ③ Si non, la boucle est finie.



Attention aux boucles INFINIES!  $\infty$

Il faut que les instructions dans la boucle puissent influencer sur la condition.

Par ex, si la condition est  $x > 0$ , il faut qu'au moins une instruction dans la boucle modifie  $x$ .



# COMMENT NE PAS SE TROMPER DE CONDITION ?

Souvent il est plus facile d'écrire une condition d'arrêt mais quand on utilise `while`, c'est une "condition de poursuite"

## ASTUCE

Trouver la condition d'arrêt, puis prendre sa négation

Ex: Demander à l'utilisateur un entier  $n$  jusqu'à que  $n$  soit un multiple de 5 strictement positif

Condition d'arrêt :  $(n > 0) \ \&\& \ (n \% 5 == 0)$

Condition de poursuite :  $!(n > 0) \ \&\& \ (n \% 5 == 0)$   
soit  $(n \leq 0) \ || \ (n \% 5 != 0)$

## SOLUTION 1

```
int n = -1;
while ( !( n > 0 && n % 5 == 0 ) ) {
    printf("Entrez un entier >0 divisible par 5 :");
    scanf("%d", &n);
}
```


## SOLUTION 2

```
int n = -1;
while ( n <= 0 || n % 5 != 0 ) {
    printf("Entrez un entier >0 divisible par 5 :");
    scanf("%d", &n);
}
```


# ÉQUIVALENCE ENTRE FOR ET WHILE

En C, on peut émuler une boucle **for** avec une boucle **while**  
(et inversement)

Par exemple,

```
for (A ; B ; C) {  
      
}
```

équivalent  
à

```
A ;  
while ( B ) {  
      
    C ;  
}
```

Toutefois on n'utilisera pas une boucle **while**  
quand on connaît le nombre d'itérations -  
Toujours une boucle **for**.

Deux raisons :

- 1 - Une boucle **for** est plus lisible qu'une fausse boucle **while**.
- 2 - Il est très facile de faire une boucle infinie avec une boucle **while**