Boucles en C

Boucle for : syntaxe

Permet de répéter plusieurs fois un bloc d'instructions associé à une variable de boucle

```
for (init ; condition_cont ; modification) {
instruction_for_1;
…
instruction_for_n;
}
```

`init:`

→ initialise la variable de boucle (qui a été déclarée avant)

`condition_cont:`

→ teste la valeur de la variable de boucle
(condition d'entrée (de continuation) de la boucle)

→ détermine si la valeur de fin de boucle est atteinte ou non

`Modification:`

→ modifie la valeur de la variable de boucle

→ attribue une valeur pour le prochain tour de boucle

→ intervient dans le bloc après : `instruction_for_n;`

Affichage des 4 premiers entiers naturels non nuls

```
int i;
for (i=1; i < 5 ; i = i + 1 ) {
printf("Valeur de i : %d \n ", i);
}
printf("Boucle finie \n");
```

Valeur de i : 1

Valeur de i : 2

Valeur de i : 3

Valeur de i : 4

Boucle finie

Boucle for : méthodologie

- Déclaration de la variable de boucle
  - i = la valeur à afficher
- Écriture du traitement à répéter en utilisant la variable de boucle *(instructions qui vont être dans la boucle)*
  - Affichage de la valeur de I
- Identification de la valeur initiale de la variable de boucle *(init)*
  - i commence à 1
- Identification de la valeur finale de la variable de boucle
  - i s'arrête à 4
- Calcul de la valeur suivante de la variable de boucle
  - i augmente de 1

Affichage des lettres entre 'a' et 'd'

```
char car;
for (car='a'; car <= 'd' ; car = car + 1 ) {
printf("Valeur de car : %c \n ", car);
}
printf("Boucle finie \n");
```

Valeur de car : 'a'

Valeur de car : 'b'

Valeur de car : 'c'

Valeur de car : 'd'

Boucle finie

Notation

- **Incrémentation et décrémentation d'une variable**
  - ↪ `var = var+1` **peut s'écrire** `var++`
  - ↪ `var = var-1` **peut s'écrire** `var--`

```
int i;
for (i=1; i < 5; i++) {
printf("Valeur de i : %d \n", i);
}
printf("Boucle finie \n");
```

Affichage des multiples de 3 inférieurs à 10

```
int i;
printf("Multiples de 3\n");
for (i=3; i < 10 ; i = i + 3 ) {
printf("%d \n ",i);
}
printf("Boucle finie \n");
```

```
-----------------------------------
Multiples de 3
3
6
9
Boucle finie
```

Directive `#define`

- Nommer des valeurs (sans déclarer de variables)
- Abstraction : utiliser le nom plutôt que la valeur

- AVANTAGES :
  - Facilite la modification et la mise au point du programme
  - réutilisation simple de la valeur
  - modification de la valeur à un seul endroit
  - Meilleur style de programmation

- `#define NOM valeur`
  - Le préprocesseur (avant compilation) remplace toutes les apparitions de NOM par valeur
  - Par convention les noms seront en majuscules

Affichage des N premiers entiers naturels non nuls

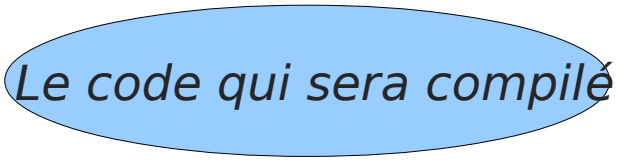
Même algorithme que affichage des entiers de 1 à 4, de 1 à 10 ou …

Ecriture d'un algorithme général, indépendant du nombre d'entiers à afficher

```
#include <stdio.h>
#define N 7
int main() {
int i;
for (i=1 ; i < N ; i++) {
printf("Valeur de i : %d \n",i);
}
return 0;
}
```

Préprocesseur (avant compilation)

```
int main() {
int i;
for (i=1 ; i < 7 ; i++) {
printf("Valeur de i : %d \n",i);
}
return 0;
}
```



Attention : `valeur ? variable`

```
#include <stdio.h>
#define N 3
int main() {
N = N - 1;
return 0;
}
```

Boucles conditionnées

Ecriture d'un programme qui :

Demande la saisie d'un entier n

Puis

Si n est négatif affiche par ordre décroissant les entiers entre -1 et n
Si n est positif affiche par ordre croissant les entiers entre 1 et n
Sinon affiche 0

Boucles conditionnées

```
#include <stdio.h>
int main(){
int n;
scanf("%d", &n);
if (n < 0) {
...
} else {
    if (n > 0) {
        ...
    }
    else { /* n = 0 */
        ...
    }
  }
return 0
}
```

Ecriture des conditions

Boucles conditionnées

```
#include <stdio.h>
int main(){
int n;
scanf("%d", &n);
if (n < 0) {
  for (i=n ; i >= -1 ; i--) {
      printf("%d \n", i);
  }
} else {
     if (n > 0) {
         for (i=1 ; i <= n ; i++) {
             printf("%d \n", i);
         }
     }
    else { /* n = 0 */
       printf("0 \n");
    }
return 0 ;
}
```

Ecriture des boucles

Condition dans une boucle

Ecriture d'un programme qui affiche les multiples de 3 ou 5 compris entre 0 et 20

- prendre tous les nombres entre 0 et 20
- pour chacun d'eux tester s'il est divisible par 3 ou 5. Si oui, l'afficher

Algorithme indépendant des valeurs 3, 5 et 20

Condition dans une boucle

```
#include <stdio.h>
#define N 3
#define M 5
#define MAX 20
int main(){
int nb;
printf("Multiples de %d ou %d \n", M, N);
for (nb=0; nb <= MAX; nb++){
...
}
return 0;
}
```

Ecriture de la boucle

Condition dans une boucle

```
#include <stdio.h>
#define N 3
#define M 5
#define MAX 20
int main(){
int nb;
printf("Multiples de %d ou %d \n", M, N);
for (nb=0; nb <= MAX; nb++){
    if ( ((nb % N) == 0) || ((nb % M) == 0) ) {
        printf("%d \n", nb);
    }
}
return 0;
}
```

Ecriture de l'affichage conditionné

Exécution
Multiples de 3 ou 5
3
5
6
9
10
12
15
18

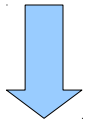
Condition dans une boucle

Attention à l'alternative !

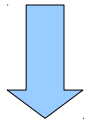
```
for(i=0;i<N;i=i+1) {
  if (condition) {
    instruction1;
  } else {
    instruction2;
  }
}
```

VS

```
for(i=0;i<N;i=i+1) {
  if (condition) {
    instruction1;
  }
}
instruction2;
```



Effectue instruction2 à chaque fois qu'on entre dans la boucle et que la condition du if est fausse



Effectue instruction2 Seulement à la fin de la boucle

Boucles for imbriquées

Ecriture d'un programme qui affiche les tables de multiplication de 1 à 5

- prendre tous les nombres entre 1 et 5
- pour chacun d'eux afficher la table de multiplication

- Afficher la table de N
  - Prendre tous les nombres de 1 à 10
  - Afficher la valeur i*N

Affichage de N tables de multiplications

```
#include <stdio.h>
#define N 5
#define MAX 10
int main(){
 int j, i;
 for (i=1 ; i <= N ; i++) {
        printf("Table de %d \n", i);
        for (j=1 ;j <= MAX ;j++){
            printf("%d ",i*j);
        }
        printf("\n") ;
   }
return 0;
}
```

Table de i

N tables

Boucles for imbriquées : exécution

```
i = 1 ( i <= 5)
j varie de 1 à 10

i = 2 ( i <= 5)
j varie de 1 à 10

i = 3 ( i <= 5)
j varie de 1 à 10

i = 4 ( i <= 5)
j varie de 1 à 10

i = 5 ( i <= 5)
j varie de 1 à 10

i = 6 ( i > 5)
```

```
Table de 1
1 2 3 4 5 6 7 8 9 10

Table de 2
2 4 6 8 10 12 14 16 18 20

Table de 3
3 6 9 12 15 18 21 24 27 30

Table de 4
4 8 12 16 20 24 28 32 36 40

Table de 5
5 10 15 20 25 30 35 40 45 50
```

Les boucles while et do-while

- On ne connait pas toujours le nombre d'opérations nécessaires pour obtenir un résultat
  - Données saisies par un utilisateur, par exemple un code d'accès, avec risque d'erreur de saisie
  - Recherche d'une valeur particulière dans un ensemble de données
  - Etc

- La condition de continuation ne porte plus (uniquement) sur un compteur

La boucle while

- La condition de continuation est évaluée avant d'entrer dans la boucle
  - Elle porte sur une ou plusieurs variables initialisées avant la boucle
  - Si la condition est fausse initialement, le programme n'exécute aucune instruction de la boucle
  - Attention : si la condition de continuation ne devient jamais fausse, le programme ne sort jamais de la boucle
    - dans la boucle, une ou plusieurs instructions agissent sur les variables de la condition de continuation pour la faire évoluer vers la condition d'arrêt

Syntaxe de la boucle while

- Avec une seule instruction dans la boucle :

```
while (expression_booléenne )
        instruction ;
```

- Avec un bloc d'instructions :

```
while ( expression_booléenne ) {
        instruction_while_1 ;
                …
        instruction_while_n ;
                }
```

Exemple de la boucle while

Le lièvre et la tortue

- Initialement, le lièvre est à la position `pos_lievre` et la tortue à la position `pos_tortue`

- Chaque seconde, le lièvre parcourt une distance `D_LIEVRE` et la tortue une distance `D_TORTUE`

- Après combien de secondes le lièvre est-il devant la tortue ?

  - Si `(pos_lievre > pos_tortue)` initialement, il n'y a pas de calcul à effectuer

    → on ne passe pas dans la boucle

  - Sinon, on ajoute la distance parcourue chaque seconde jusqu'à ce que `(pos_lievre > pos_tortue)`

    → on ne connait pas le nombre d'itérations

    → boucle while

La boucle

- Chaque seconde (→ répétition) :
  - le lièvre avance : `pos_lievre = pos_lievre + D_LIEVRE ;`
  - la tortue avance : `pos_tortue = pos_tortue + D_TORTUE ;`
  - le chronomètre tourne : `cpt++;`

- Condition de continuation : le lièvre n'est pas devant la tortue :

  `pos_lievre <= pos_tortue;`

- Initialement :
  - `pos_lievre = 0;`
  - `pos_tortue = 10;`
  - `cpt = 0;`

La lièvre et la tortue

```
#include <stdio.h>
#define D_LIEVRE 3
#define D_TORTUE 1
#define P_LIEVRE 0
#define P_TORTUE 10
int main() {
 int pos_lievre = P_LIEVRE;
 int pos_tortue = P_TORTUE;
 int cpt = 0;
   while (pos_lievre <= pos_tortue) {
     pos_lievre = pos_lievre + D_LIEVRE ;
     pos_tortue = pos_tortue + D_TORTUE ;
     cpt++;
   }
 printf("A l'instant %d, le lièvre est devant la tortue\n", cpt);
 return 0;
}
```

La boucle do-while

- La condition de continuation est évaluée à la sortie de la boucle
  - même si la condition est fausse initialement, le programme passe au moins une fois dans la boucle
  - les variables de la condition peuvent être initialisées dans la boucle

- Comme dans la boucle while :
  - si la condition de continuation ne devient jamais fausse, le programme ne sort jamais de la boucle
  - dans la boucle, une ou plusieurs instructions agissent sur les variables de la condition de continuation pour la faire évoluer vers la condition d'arrêt

Syntaxe de la boucle do-while

- Avec une seule instruction dans la boucle

```
do
    instruction ;
while (expression_booléenne) ;
```

- Avec un bloc d'instructions :

```
do {
    instruction_1 ;
    ...
    instruction_n ;
} while (expression_booléenne) ;
```

Exemple de boucle do-while

- Demande de saisie d'un entier entre 1 et 10, sans limiter le nombre de tentatives.
  - répétition des saisies → boucle
  - condition de continuation :
    - la valeur saisie est inférieure à 1 ou supérieure à 10
- Il faut effectuer une saisie avant de pouvoir évaluer la condition :

  → boucle do-while

Saisie d'une valeur entre 1 et 10

```
#include <stdio.h>
int main() {
 int val;
 do {
    printf("Entrez une valeur entre 1 et 10 : \n");
    scanf("%d", &val);
    } while ((val < 1) || (val > 10)) ;
 printf("la valeur choisie est : %d \n", val);
 return 0;
}
```

Boucle do-while avec condition complexe

- Demande de saisie d'un entier entre 1 et 10, en limitant le nombre de tentatives à 3.
  - condition de continuation :
    - la valeur saisie est inférieure à 1 ou supérieure à 10

    et
    - le nombre de tentatives est inférieur à 3

- Comment sait-on à la sortie si la saisie a été réussie ou non ?

La boucle

```
#include <stdio.h>
#define N 3
int main() {
 int val;
 int i=0;
 do {
    printf("Entrez une valeur entre 1 et 10 : \n");
    scanf("%d", &val);
    i++;
    } while ((i < N) && ((val < 1) || (val > 10))) ;
```

Le test à la sortie de boucle

Après 3 tentatives, i = 3, que la dernière saisie ait été réussie ou non

→ on est obligé de tester val

```
if ((val < 1) || (val > 10)) {
printf("Echec : 3 saisies incorrectes \n");
}
else {
printf("la valeur choisie est : %d, val \n");
}
return 0 ;
}
```

Les boucles for / while / do-while

- La boucle for :
  - L'initialisation, la condition et la modification porte sur la même variable
  - On sait combien de fois le corps de la boucle sera exécuté

- La boucle while :
  - Le nombre d'itérations n'est pas connu, il est possible qu'on n'exécute jamais les instructions de la boucle

- La boucle do-while
  - Le nombre d'itérations n'est pas connu, mais on passe toujours au moins une fois dans la boucle
    - → souvent utilisée pour les saisies avec scanf

- Une boucle for peut étre ré-écrite en boucle while
  - Utiliser un do-while ou un for/while est réellement différent