

Boucles *do-while* et *while*

La boucle `for` permet de répéter une partie du programme (des instructions) un certain nombre de fois.

```
for (int i = 0; i < nombreRepetitions; i++){ Instructions }
```

En effet, on ne peut utiliser la boucle `for` que si le nombre de répétitions est connu à l'avance. Alors, comment fait-on dans les situations où on ne connaît pas combien de fois on doit répéter certains traitements (instructions), comme par exemple :

- Tant que le mot de passe entré par l'utilisateur est erroné, relisez le mot de passe.
- Tant que l'entrée de l'utilisateur n'est pas un nombre positif (âge, année de naissance, ...), on lui invite à entrer une nouvelle valeur.

Dans ces cas, on ne peut pas utiliser la boucle `for` et on doit avoir recours à d'autres types de boucle, à savoir, la boucle `do-while` ou la boucle `while`. Considérons cet exemple : tant que l'âge entré par l'utilisateur n'est pas un nombre positif, on veut lui inviter à entrer une nouvelle valeur.

```
int age;
cout << "Entrez votre age: "; cin >> age;
```

On veut répéter les deux instructions précédentes jusqu'à ce que l'utilisateur entre un âge valide (valeur supérieure à zéro). Dans ce cas, combien de fois on doit répéter ces instructions? Une seule fois, 2 fois? 3 fois? ... Est-ce que l'utilisateur va donner un âge valide, dès le début? Après la deuxième tentative? La troisième? Ou quand exactement? En effet, on ne sait pas combien de fois il faudrait répéter ces instructions, puisque on ne peut pas prédire a priori quand l'utilisateur va effectivement saisir un âge valide (strictement supérieur à 0). Donc, vu le fait qu'on ne sait pas combien de fois on doit répéter les instructions, alors il faut avoir recours à la boucle `do-while` au lieu de la boucle `for`.

La boucle `do-while` s'écrit comme suit :

```
int age;

do {
    cout << "Entrez votre age: "; cin >> age;
} while (age <= 0);
```

`While` veut dire tant que en Anglais.

Syntaxe de la boucle `do-while`

```
do {
    // Instructions
} while (Condition);
```

La boucle `while` permet de répéter les instructions entre accolades, *tant que* la condition entre parenthèses est vraie.

Exécutons l'exemple ci-dessus pas à pas :

```
int age;
```

age	...
	?
	...

Quand on arrive au mot-clé `do` rien ne nous empêche d'entrer dans la boucle et exécuter les instructions :

```
cout << "Entrez votre age: "; cin >> age;
```

Supposons que l'utilisateur a entré (-17) comme âge.

age	...
	-17
	...

Après avoir exécuté les instructions de lecture (`cout` et `cin`), on passe pour vérifier la condition se trouvant entre parenthèses : `while (condition);` Si la condition est fausse, alors on sort de la boucle. Sinon, i.e., la condition est vraie, alors, on remonte et on ré-exécute la boucle (à partir du mot-clé `do`). Dans notre cas, la condition `while (age <= 0);` est vraie vu le fait que `age == -17`, alors on remonte et on ré-exécute les instructions `cout` et `cin`.

Supposons maintenant que l'utilisateur a entré (0) comme valeur.

age	...
	0
	...

On révérifie la condition : `while (age <= 0);` condition vraie, alors on ré-exécute la boucle (à partir du mot-clé `do`).

Supposons, cette fois-ci, que l'utilisateur a entré (20) comme valeur.

age	...
	20
	...

Maintenant, la condition, `while (age <= 0);` est devenue fausse, ce qui signifie que l'exécution de la boucle est terminée et l'ordinateur continuera l'exécution des instructions après la boucle.

Remarques

1. Les instructions à l'intérieur de la boucle `do-while` sont toujours exécutées au moins une fois.

```
do {  
    // Instructions  
} while (Condition);
```
2. Comme pour le `if`, la condition peut utiliser des opérateurs logiques (`and`, `or`, `not`). Par exemple :

```
do {  
    // Instructions  
} while ((x > 0) and (y > 0) or (z > 0));
```
3. Les parenthèses qui entourent la condition sont obligatoires.

```
do {  
    // Instructions  
} while (condition);
```
4. N'oubliez pas le point-virgule ";" à la fin de la boucle `do-while`.
5. Qu'affiche la boucle suivante :

```
6. int i = 1;
   do {
       cout << i;
   } while (i > 0);
```

Cette boucle affichera : 11111111111111...

En effet, si la condition ne devient jamais fausse, les instructions seront exécutées indéfiniment.

Boucle While

```
while (condition){
    // Instructions
}
```

Le principe de cette boucle est similaire à celui de la boucle `do-while`. La différence est que la condition est testée, dès le début, *avant* d'entrer dans la boucle. Si la condition est fausse, les instructions dans la boucle ne seront pas exécutées.

Exemple

```
int i = 0;
while (i < 2){
    cout << "Hello" << endl;
    i++;
}
```

Exécution pas à pas:

```
int i = 0;
```

	...
i	0
	...

Quand on arrive au mot-clé `while`, on est obligé de vérifier la condition avant d'entrer et exécuter les instructions de la boucle. On a `i == 0` donc la condition `(i < 2)` est vérifiée, alors on entre et on exécute les instructions : `cout << "Hello" << endl; i++;`

Ce qui affiche "Hello" à l'écran et incrémente `i` de 1.

	...
i	1
	...

Après exécution du bloc d'instructions, on remonte et on revérifie la condition. On a `i == 1`, donc la condition `(i < 2)` est vérifiée, alors on entre et on exécute les instructions : `cout << "Hello" << endl; i++;`

Ce qui affiche "Hello" à l'écran et incrémente `i` de 1.

	...
i	2
	...

Après, on remonte et on revérifie la condition. On a `i == 2`, donc la condition `(i < 2)` n'est pas vérifiée, alors on sort, cette fois-ci, de la boucle.

Exemple : dans les deux cas (programmes) suivants, la condition est fausse

```
int i = 100;
while (i < 100){
    cout << "Hello!" << endl;
}
```

Cette boucle n'affichera rien.

```
int i = 100;
do {
    cout << "Hello!" << endl;
} while (i < 100);
```

Cette boucle affichera : "Hello!"

Erreurs classiques : Il y a un point-virgule à la fin de la boucle `do-while`, en revanche, il n'y a pas de point-virgule à la fin de la boucle `while`. Par exemple :

```
int i = 1;
while (i < 10);
    i++;
```

Le morceau de code ci-dessus sera interprété comme suit :

```
int i = 1;
while (i < 10)
    ;
i++;
```

Le point-virgule est considéré comme le corps de la boucle `while` et l'instruction `i++` est après la boucle. `i` est inférieur à 10, alors on entre dans la boucle pour ne jamais en ressortir puisque la valeur de `i` ne sera jamais modifiée. Notez que l'exemple, ci-dessus, est donné juste pour illustrer le propos. On devait écrire :

```
int i = 1;
while (i < 10){
    i++;
}
```

Même que les accolades ne sont pas obligatoires dans le cas d'une seule instruction, il est très conseillé de les utiliser.

Exercice : Qu'affiche ce programme?

```
int i = 1;
while (i < 10){
    cout << i << endl;
    i++;
}
```

Le programme affichera : 1, 2, 3, 4, 5, 6, 7, 8, 9

On commence à partir de 1 et on sort de la boucle lorsque `i` prendra la valeur 10.

Quelle boucle dois-je utiliser?

La question qui se pose est :

Quand utiliser la boucle `for`?

Quand utiliser la boucle `do-while`?

Quand utiliser la boucle `while`?

- Si le nombre d'itérations (répétitions) est connu à l'avance, alors on utilise la boucle `for`.

```
for (int i = 0; i < nombreIterations; i++){ Instructions }
```

Par exemple, si on veut calculer la moyenne de tous les étudiants. On connaît combien d'étudiants sont inscrits, alors on utilise une boucle `for`.

- Sinon, quand les instructions doivent être répétées au moins une fois. Par exemple, dans le cas où on doit tout d'abord récupérer une valeur de l'utilisateur, alors on utilise la boucle `do-while`.
- Sinon, on utilise la boucle `while`.

Exemple

```
int age;
do {
    cout << "Entrez votre age: "; cin >> age;
} while (age <= 0);
```

Il s'agit d'un cas typique où nous avons besoin d'une boucle `do-while`, parce que l'utilisateur a besoin de saisir son âge au moins une fois avant que l'on soit capable de tester s'il a entré une bonne valeur. On pourrait écrire la solution en utilisant une boucle `while` comme suit :

```
int age;
cout << "Entrez votre age: "; cin >> age;
while (age <= 0){
    cout << "Entrez votre age: "; cin >> age;
}
```

Remarquez bien que les instructions de lecture se répètent : `cout << "Entrez votre age: "; cin >> age;`

Dans un programme, il est toujours souhaitable d'éviter les situations où on exprime deux fois (ou même plusieurs fois) de façon indépendante la même chose à des endroits différents. Pourquoi? Parce que, par exemple, si on veut modifier une partie, on pourrait oublier de modifier l'autre. Donc, dans l'exemple précédent, il est préférable d'utiliser une boucle `do-while` au lieu de la boucle `while` :

```
int age;
do {
    cout << "Entrez votre age: "; cin >> age;
} while (age <= 0);
```

Exemple : Reprenons l'exemple des notes qu'on a vu dans la section précédente (boucle `for`).

```
#include<iostream>
using namespace std;
int main(){
    int nombreNotes;
    cout << "Entrez le nombre de notes: "; cin >> nombreNotes;

    double note, somme = 0;
    for (int i = 1; i <= nombreNotes; i++){
        cout << "Note " << i << ": "; cin >> note;
        somme = somme + note;
    }

    if (nombreNotes > 0){ cout << "Moyenne = " << somme / nombreNotes << endl; }
    else { cout << "Error: nombre de notes incorrect!"; }
}
```

On veut répéter les instructions suivantes : `cout << "Entrez le nombre de notes: "; cin >> nombreNotes;` jusqu'à ce que l'utilisateur entre un nombre de notes supérieur à 0. Alors, comment forcer l'utilisateur à entrer un nombre de notes supérieur à zéro? La solution consiste à utiliser une boucle `do-while` comme suit :

```
do {
    cout << "Entrez le nombre de notes: "; cin >> nombreNotes;
} while (nombreNotes <= 0);
```

En utilisant la boucle `do-while`, on sera sûr que l'utilisateur a entré une valeur supérieure à zéro. Ce qui veut dire que le `if` se trouvant à la fin du programme devient inutile :

```
int main(){
    int nombreNotes;
    do {
        cout << "Entrez le nombre de notes: "; cin >> nombreNotes;
    } while (nombreNotes <= 0);

    double note, somme = 0;
    for (int i = 1; i <= nombreNotes; i++){
        cout << "Note " << i << ": "; cin >> note;
        somme = somme + note;
    }

    cout << "Moyenne = " << somme / nombreNotes << endl;
}
```

Maintenant, on veut améliorer le programme de sorte que l'utilisateur soit avisé, par le biais d'un message, au moment où il entre un nombre négatif, comme suit :

```
Entrez le nombre de notes : -2
Error: nombre de notes incorrect!".
Entrez le nombre de notes : 0
Error: nombre de notes incorrect!".
Entrez le nombre de notes : _
```

Pour faire cela, il faut modifier la boucle `do-while` comme suit :

```
int nombreNotes;
do {
    cout << "Entrez le nombre de notes: "; cin >> nombreNotes;
    if (nombreNotes <= 0){ cout << "Error: nombre de notes incorrect!" << endl; }
} while (nombreNotes <= 0);
```

Supposons maintenant qu'on veuille limiter le nombre de notes à 10 (et on veut toujours qu'il soit supérieur à 0) : `nombreNotes > 0` et `nombreNotes <= 10`. Comment formuler la condition?

On pose la question : "on répète la boucle tant que?". Dans notre exemple, on répète la boucle, tant que le nombre de notes est incorrect. Plus précisément, on répète la boucle tant que le nombre de notes est inférieur ou égal à zéro *ou* supérieur à 10.

```
int nombreNotes;
do {
    cout << "Entrez le nombre de notes: "; cin >> nombreNotes;
    if (nombreNotes <= 0){ cout << "Error: nombre de notes incorrect!" << endl; }
} while (nombreNotes <= 0 or nombreNotes > 10);
```

Si vous avez bien remarqué la condition `(nombreNotes <= 0)` n'apparaît pas seulement dans le `while` (comme condition de continuation) mais aussi dans le `if`, donc, on doit aussi modifier la condition du `if` :

```
int nombreNotes;
do {
    cout << "Entrez le nombre de notes: "; cin >> nombreNotes;
    if (nombreNotes <= 0 or nombreNotes > 10){
        cout << "Error: nombre de notes incorrect!" << endl;
    }
} while (nombreNotes <= 0 or nombreNotes > 10);
```

Il est souhaitable dans un programme d'éviter les situations où on répète des portions de code. Pourquoi? Parce que comme on vient de voir, en modifiant une partie du programme, on pourrait oublier de modifier d'autres parties. Pour éviter cette répétition, une bonne pratique consiste à stocker le résultat de l'évaluation de la condition dans une variable afin de ne pas répéter son évaluation.

```
int nombreNotes;
bool saisie_invalide;
do {
    cout << "Entrez le nombre de notes: "; cin >> nombreNotes;
    saisie_invalide = (nombreNotes <= 0 or nombreNotes > 10);
    if (saisie_invalide){ cout << "Error: nombre de notes incorrect!" << endl; }
} while (saisie_invalide);
```

Remarque : l'instruction : `if (saisie_invalide){...}` est équivalente à :

`if (saisie_invalide == true){...}`

De même, l'instruction : `if (!saisie_invalide){...}` est équivalente à :

`if (not saisie_invalide){...}` qui est aussi équivalente aussi à :

`if (saisie_invalide == false){...}`

Exercice : écrire un programme C++ permettant d'inverser un nombre entier saisi au clavier. Par exemple : 5679 → 9765.

```
int main(){
    int n;
    cout << "n = "; cin >> n;

    int x = 0;
    while (n > 0){
        x = x * 10 + n % 10;
        n = n / 10;
    }
    cout << "x = " << x << endl;
}
```

Exercice : écrire un programme C++ qui permet de déterminer le chiffre le plus grand dans un nombre entier saisi au clavier. Par exemple :

5679 → 9

-6566 → 6

```
#include<cmath>
int main(){
    int n;
    cout << "n = "; cin >> n;

    int max = 0;
    n = abs(n);
    while (n > 0){
        int chiffre = n % 10;
        if (max < chiffre){ max = chiffre; }
        n = n / 10;
    }
    cout << "max = " << max << endl;
}
```

Remarque : la fonction `abs(n)` retourne la valeur absolue de la variable `n`.

Exercice - Deviner un nombre : supposons qu'on veuille écrire un programme qui demande à l'utilisateur de deviner un nombre. Pour simplifier, supposons que le nombre à deviner est toujours 5.

Le programme peut s'écrire ainsi :

```
int main(){
    int secret = 5;
    int guess;

    do{
        cout << "Deviner un nombre compris entre 0 et 5: "; cin >> guess;
    } while(condition);

    cout << "Bravo!";
}
```

Comment formuler la condition? Formulons la condition en langue naturelle : la boucle doit être répétée tant que l'utilisateur n'a pas trouvé le nombre à deviner, c'est-à-dire, tant que le nombre entré est différent du nombre à deviner. La condition est donc :

```
int main(){
    int secret = 5;
    int guess;

    do{
        cout << "Deviner un nombre compris entre 0 et 5: "; cin >> guess;
    } while(secret != guess);

    cout << "Bravo!";
}
```

Après exécution, on aura quelque chose comme :

```
Deviner un nombre compris entre 0 et 5: 1
Deviner un nombre compris entre 0 et 5: 4
Deviner un nombre compris entre 0 et 5: 2
Deviner un nombre compris entre 0 et 5: 5
Bravo!
```

Avec le programme précédent, l'utilisateur pourra proposer des valeurs indéfiniment. Supposons qu'on veuille limiter le nombre d'essais à 3. Pour ce faire, on doit ajouter une variable qui va compter le nombre d'essais.


```
int main(){
    int secret = 5;
    int guess;
    int nombreEssais = 0;

    do{
        cout << "Deviner un nombre compris entre 0 et 5: "; cin >> guess;
        nombreEssais++;
    } while(condition?);

    cout << "Bravo!";
}
```

Comment modifier la condition pour que la boucle s'arrête quand le nombre d'essais dépasse 3? Commencer par formuler la condition en langue naturelle : la boucle est répétée tant que l'utilisateur n'a pas trouvé le nombre à deviner et qu'il lui reste des essais. Plus précisément, la boucle est répétée tant que le nombre à deviner est différent du nombre entré par l'utilisateur et que le nombre d'essais est inférieur à 3.

```
#include<iostream>
using namespace std;
int main(){
    int secret = 5;
    int guess;
    int nombreEssais = 0;

    do{
        cout << "Deviner un nombre compris entre 0 et 5: "; cin >> guess;
        nombreEssais++;
    } while(secret != guess and nombreEssais < 3);
    cout << "Bravo!";
}
```

Si vous avez remarqué, le programme ci-dessus affiche toujours "Bravo!" et ignore que l'utilisateur a désormais deux façon pour sortir de la boucle. Il peut en sortir parce qu'il a trouvé le nombre à deviner (`secret != guess`) ou alors parce qu'il a dépassé le nombre d'essais (`nombreEssais < 3`).

```
Deviner un nombre compris entre 0 et 5: 1
Deviner un nombre compris entre 0 et 5: 4
Deviner un nombre compris entre 0 et 5: 2
Bravo!
```

Donc, même si l'utilisateur n'a pas trouvé le nombre, le programme affiche "Bravo!". Pour résoudre ce problème et informer l'utilisateur pour quelle raison on est sorti de la boucle, on doit compléter le programme comme suit. Après la boucle `do-while`, on ajoute un instruction `if-else` :

```
int main(){
    int secret = 5;
    int guess;
    int nombreEssais = 0;

    do{
        cout << "Deviner un nombre compris entre 0 et 5: "; cin >> guess;
        nombreEssais++;
    } while(secret != guess and nombreEssais < 3);

    if (secret == guess){ cout << "Bravo!"; }
    else { cout << "Game Over!"; }
}
```

Attention, si on avait utilisé `(nombreEssais < 3)` comme condition :


```
if (nombreEssais < 3){ cout << "Bravo!"; }
else { cout << "Game Over!"; }
```

Alors, le programme afficherait "Game over!" quand l'utilisateur trouve le nombre au troisième essai.

```
Deviner un nombre compris entre 0 et 5: 1
Deviner un nombre compris entre 0 et 5: 2
Deviner un nombre compris entre 0 et 5: 5
Game Over!
```

Question : que se passe-t-il si on inverse les deux conditions?

```
do{
    cout << "Deviner un nombre compris entre 0 et 5: "; cin >> guess;
    nombreEssais++;
} while(secret != guess and nombreEssais < 3);
```



Amélioration du programme

Nous pouvons améliorer le programme davantage en générant le nombre secret aléatoirement. Voici comment on peut générer un nombre aléatoire compris entre 1 et 5 en C++ :

```
#include<iostream>
#include<cstdlib>
#include<ctime>
using namespace std;

int main(){
    srand(time(NULL));
    // Generate a random number between 1 and 5
    int secret = rand() % 5 + 1;

    cout << "Nombre secret = " << secret << endl;
}
```

Le programme complet s'écrit ainsi :

```
#include<iostream>
#include<cstdlib>
#include<ctime>
using namespace std;

int main(){
    // Generate a random number between 1 and 5
    srand(time(NULL));
    int secret = rand() % 5 + 1;

    int guess;
    int nombreEssais = 0;
    do{
        cout << "Deviner un nombre compris entre 0 et 5: "; cin >> guess;
        nombreEssais++;
    } while(nombreEssais < 3 and secret != guess);

    if (secret == guess){ cout << "Bravo!"; }
    else { cout << "Game Over!"; }
}
```