Programmation C++ (débutant)/Instructions for, while et do...while

Le cours du chapitre 4 : le for, while et do...while

La notion de boucles

Dans ce chapitre, nous allons étudier les structures de contrôle permettant d'effectuer des boucles : le for, le while et le do…while. On parle de boucles lorsqu'on répète l'exécution d'une série d'instructions à l'intérieur d'un programme. La notion de boucle est une des notions à la base de toute l'algorithmique.

Le for

Le for est une structure de contrôle qui permet de répéter un certain nombre de fois une partie d'un programme.

Syntaxe :

```
for( instruction1 ; condition ; instruction2 )
     instruction3 ;
```

Sémantique du for :

1. on exécute l'instruction1
2. on teste la condition :
   - si elle est vraie, on exécute l'instruction3, puis l'instrution2 puis on revient au 2.
   - si elle est fausse on passe à l'instruction suivante.

L'instruction3 peut être une suite d'instructions entre accolades.

Exemple 1 : utilisation du for

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

int main()
{
    int i;
    for(i=0;i<10;i=i+1)
        cout<<"BONJOUR"<<endl;
    return 0;
}
```

- Dans ce programme, il y a une boucle de type for :
  - L'instruction 1 est : i=0
  - L'instruction 2 est : i=i+1
  - Le corps du for comporte une seule instruction : cout<<"BONJOUR"<<endl;
- **Exécution pas à pas de l'exemple 1**
  - i vaut 0
  - Le test i<10 est vrai ==> on exécute le corps du for avec i=0
  - On exécute i=i+1 ==> i vaut 1
  - Le test i<10 est vrai ==> on exécute le corps du for avec i=1

- On exécute i=i+1 ==> i vaut 2
- Le test i<10 est vrai ==> on exécute le corps du for avec i=2
- On exécute i=i+1 ==> i vaut 3
- Le test i<10 est vrai ==> on exécute le corps du for avec i=3
- On exécute i=i+1 ==> i vaut 4
- Le test i<10 est vrai ==> on exécute le corps du for avec i=4
- On exécute i=i+1 ==> i vaut 5
- Le test i<10 est vrai ==> on exécute le corps du for avec i=5
- On exécute i=i+1 ==> i vaut 6
- Le test i<10 est vrai ==> on exécute le corps du for avec i=6
- On exécute i=i+1 ==> i vaut 7
- Le test i<10 est vrai ==> on exécute le corps du for avec i=7
- On exécute i=i+1 ==> i vaut 8
- Le test i<10 est vrai ==> on exécute le corps du for avec i=8
- On exécute i=i+1 ==> i vaut 9
- Le test i<10 est vrai ==> on exécute le corps du for avec i=9
- On exécute i=i+1 ==> i vaut 10
- Le test i<10 est faux ==> on sort du for avec i=10

- **Résumé de l'exemple 1**
  - On a exécuté 10 fois le corps du for.
  - Le programme affiche donc 10 fois BONJOUR à l'écran.</pre>

Exemple 2 : un deuxième exemple de for

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

int main()
{
    int i;
    for (i=0; i<10; i=i+1)
        cout<<"La valeur de i est : "<<i<<endl;
    cout<<"La valeur finale de i est : "<<i<<endl;
    return 0;
}
```

- **Dans ce programme, il y a une boucle de type for :**
  - L'instruction 1 est : i=0
  - La condition est : i<10
  - L'instruction 2 est : i=i+1
  - Le corps du for comporte une seule instruction : cout<<"La valeur finale de i est : "<<i<<endl; ** On a exécuté 10 fois le corps du for : **La première fois avec i valant 0. **La dernière fois avec i valant 9. ** On quitte le for avec i valant 10. *'"Exécution de l'exemple 2"' La valeur de i est : 0 La valeur de i est : 1 La valeur de i est : 2 La valeur de i est : 3 La valeur de i est : 4 La valeur de i est : 5 La valeur de i est : 6 La valeur de i est : 7 La valeur de i est : 8 La valeur de i est : 9 La valeur finale de i est : 10 === Exemple 3 : encore un exemple de for ! === UNIQ-source-3-5df81f2d7198fec5-QINU * "'On a une boucle de type for "' **on va exécuter le corps de la boucle la première fois avec i valant 8, la dernière fois avec i valant 18. ** Lorsqu'on quitte le for, i vaut 19. *'"Exécution de l'exemple 3"' La valeur de i est : 8 La valeur de i est : 9 La valeur de i est : 10 La valeur de i

est : 11 La valeur de i est : 12 La valeur de i est : 13 La valeur de i est : 14 La valeur de i est : 15 La valeur de i est : 16 La valeur de i est : 17 La valeur de i est : 18 La valeur finale de i est : 19 === Exemple 4 : un for décroissant === UNIQ-source-4-5df81f2d7198fec5-QINU *'"Explication du for"' ** Cette fois-ci, à chaque étape on effectue i--, c'est-à-dire on décrémente i de 1. ** On va donc exécuter le corps du for la première fois avec i valant 10, la dernière fois avec i valant 4. ** Lorsqu'on quitte le for, i vaut 3. *'"Exécution de l'exemple 4"' La valeur de i est : 10 La valeur de i est : 9 La valeur de i est : 8 La valeur de i est : 7 La valeur de i est : 6 La valeur de i est : 5 La valeur de i est : 4 La valeur finale de i est : 3 === Exemple 5 : un for de 2 en 2 === UNIQ-source-5-5df81f2d7198fec5-QINU *'"Explication du for"' ** Cette fois-ci, à chaque étape, on ajoute 2 à i. ** i va donc valoir successivement 10,12, 14, 16 et 18. ** Lorsque i vaut 20, la condition devient fausse et on quitte la for. *'"Exécution de l'exemple 5"' La valeur de i est : 10 La valeur de i est : 12 La valeur de i est : 14 La valeur de i est : 16 La valeur de i est : 18 La valeur finale de i est : 20 === Exemple 6 : un for qui ne s'arrête jamais === UNIQ-source-6-5df81f2d7198fec5-QINU *'"Explications du for"' ** La valeur initiale de i est 10. ** A chaque étape, on exécute i++. ** La valeur de i sera toujours plus grande que 3. ** La condition i>3 est toujours vraie ==> le for ne s'arrête jamais.

- Le programmeur voulait certainement écrire i=i-1; au lieu de i++.

- **Exécution de l'exemple 6**

```
La valeur de i est : 10
La valeur de i est : 11
La valeur de i est : 12
La valeur de i est : 13
La valeur de i est : 14
...
```

En fait, le type `int` étant de taille fixe (16 bits ou 32 bits selon le compilateur) et signé, la boucle se termine quand la valeur de la variable i dépasse la limite des positifs (+32767 ou +2147483647) :

Si le type `int` utilise 16 bits :

```
...
La valeur de i est : 32767
La valeur finale de i est : -32768
```

Si le type `int` utilise 32 bits :

```
...
La valeur de i est : 2147483647
La valeur finale de i est : -2147483648
```

Le while

Syntaxe

```
while ( condition )
    instruction;
```

Sémantique du while

1. On teste la condition :
   - si elle est vraie, on exécute l'instruction puis on recommence au 1).
   - si elle est fausse, on passe à l'instruction suivante.

L'instruction peut être une suite d'instructions entre accolades.

Exemple 7 : un exemple de while

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

int main()
{
    int i=0;
    while(i<10)
    {
        cout<<"La valeur de i vaut : "<<i<<endl;
        i++;
    }
    cout<<"La valeur finale de i vaut : "<<i<<endl;
    return 0;
}
```

- **Explications**
  - La variable i est initialisée à 0.
  - A chaque étape, à la fin du corps du while, on incrémente i de 1.
  - On exécute donc le corps du while la première fois avec i valant 0, la dernière fois avec i valant 9.
  - Lorsqu'on sort du while i vaut 10.
- **Exécution de l'exemple 7**

```
La valeur de i est : 0
La valeur de i est : 1
La valeur de i est : 2
La valeur de i est : 3
La valeur de i est : 4
La valeur de i est : 5
La valeur de i est : 6
La valeur de i est : 7
La valeur de i est : 8
La valeur de i est : 9
La valeur finale de i est : 10
```

Exemple 8 : valider une donnée saisie au clavier

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

int main()
{
    int i;
    cout <<"Tapez une valeur entre 0 et 20 bornes incluses : ";
    cin>>i;
    while (i<0 || i>20)
    {
        cout <<"ERREUR ! ";
```

```
        cout <<"Tapez une valeur entre 0 et 20 bornes incluses : ";
        cin >> i;
    }
    return 0;
}
```

- **Explications**
  - On saisit une première fois la valeur de i par un cin.
  - Tant que la valeur saisie ne sera pas comprise entre 0 et 20 (bornes incluses), on affiche ERREUR et on redemande une nouvelle valeur.
  - La condition « la valeur de i n'est pas comprise entre 0 et 20 » s'écrit (i<0 || i>20).
  - Ne pas confondre le ET et le OU logique !
- **Exécution de l'exemple 8**

```
Tapez une valeur entre 0 et 20 bornes incluses : 30
ERREUR ! Tapez une valeur entre 0 et 20 bornes incluses : -2
ERREUR ! Tapez une valeur entre 0 et 20 bornes incluses : 10
```

Conseil

On utilisera le `for` lorsqu'on connaît le nombre d'étapes à réaliser. On utilisera un `while` dans le cas contraire. Dans tous les cas, il faut toujours vérifier si on exécute notre boucle le bon nombre de fois. Il faudra vérifier dans tous les cas que notre boucle n'est pas infinie.

Le do...while

Syntaxe

```
do {
    instruction;
}
while ( condition );
```

Sémantique du do … while

1. on exécute l'instruction.
2. on évalue la condition.
3. si elle est vraie, on recommence au 1.
4. si elle est fausse, on passe à l'instruction suivante.

Exemple 9 : un exemple de do...while

```
#include <iostream>
using namesapce std;

int main()
{
    int i=0;
    do {
        cout<<"La valeur de i vaut : "<<i<<endl;
        i=i+1;
```

```
    } while(i<10);
    cout<<"La valeur finale de i est "<<i<<endl;
    return 0;
}
```

Exécution de l'exemple 9

```
La valeur de i vaut : 0
La valeur de i vaut : 1
La valeur de i vaut : 2
La valeur de i vaut : 3
La valeur de i vaut : 4
La valeur de i vaut : 5
La valeur de i vaut : 6
La valeur de i vaut : 7
La valeur de i vaut : 8
La valeur de i vaut : 9
La valeur finale de i est : 10
```

Récapitulatif des structures de contrôle

Le if et le if … else ne permettent pas de répéter un bout de programme.
Le switch est un if …else amélioré et ne permet pas non plus de répéter un bout de programme.
Le for, le while et le do … while permettent de construire des boucles répétitives.
Attention aux boucles infinies !

Utilisation des structures de contrôle

Au delà de ces exemples pédagogiques, la difficulté d'utilisation de ces structures provient de la difficulté à partir d'un problème donné d'imaginer une solution sous forme d'un algorithme puis de trouver les structures de contrôle adaptées pour implémenter cet algorithme. Beaucoup de pratique et de travail sont nécessaires pour mettre en œuvre ce processus mental complexe.

Exercices sur les structures de contrôle

EXERCICE 1

Ecrire un programme qui demande à l'utilisateur de taper un entier et qui affiche GAGNE si l'entier est entre 56 et 78 bornes incluses PERDU sinon.

EXERCICE 2

Ecrire un programme qui affiche tous les entiers de 8 jusqu'à 23 (bornes incluses) en utilisant un for.

EXERCICE 3

Même exercice mais en utilisant un while.

EXERCICE 4

Ecrire un programme qui demande à l'utilisateur de taper 10 entiers et qui affiche leur somme.

EXERCICE 5

Ecrire un programme qui demande à l'utilisateur de taper 10 entiers et qui affiche le plus petit de ces entiers.

EXERCICE 6

Ecrire un programme qui demande à l'utilisateur de taper un entier N et qui calcule la somme des cubes de 5^3 à N^3.

EXERCICE 7

Ecrire un programme qui demande à l'utilisateur de taper un entier N et qui calcule u(N) défini par :
u(0)=3
u(n+1)=3.u(n)+4

EXERCICE 8

Ecrire un programme qui demande à l'utilisateur de taper un entier N et qui calcule u(N) défini par :
u(0)=1
u(1)=1
u(n+1)=u(n)+u(n-1)

EXERCICE 9

Ecrire un programme qui demande à l'utilisateur de taper un entier N entre 0 et 20 bornes incluses et qui affiche N+17. Si on tape une valeur erronée, il faut afficher "erreur" et demander de saisir à nouveau l'entier.

EXERCICE 10

Ecrire un programme qui permet de faire des opérations sur un entier (valeur initiale à 0). Le programme affiche la valeur de l'entier puis affiche le menu suivant :
1. Ajouter 1
2. Multiplier par 2
3. Soustraire 4
4. Quitter

Le programme demande alors de taper un entier entre 1 et 4. Si l'utilisateur tape une valeur entre 1 et 3, on effectue l'opération, on affiche la nouvelle valeur de l'entier puis on réaffiche le menu et ainsi de suite jusqu'à ce qu'on tape 4. Lorsqu'on tape 4, le programme se termine.

EXERCICE 11

Ecrire un programme qui demande à l'utilisateur de taper des entiers strictement positifs et qui affiche leur moyenne. Lorsqu'on tape une valeur négative, le programme affiche ERREUR et demande de retaper une valeur. Lorsqu'on tape 0, cela signifie que le dernier entier a été tapé. On affiche alors la moyenne. Si le nombre d'entiers tapés est égal à 0, on affiche PAS DE MOYENNE.

EXERCICE 12

Ecrire un programme qui demande à l'utilisateur de taper un entier N et qui calcule u(N) défini par :
u(0)=3
u(1)=2
u(n)=n.u(n-1)+(n+1).u(n-2)+n

EXERCICE 13

Ecrire un programme qui demande de saisir 10 entiers et qui affiche le nombre d'occurrences de la note la plus haute.

EXERCICE 14

Ecrire un programme qui demande de saisir un entier N et qui affiche N!.

EXERCICE 15

Ecrire un programme qui demande de saisir un entier et qui indique si cet entier est premier ou non.

EXERCICE 16

Ecrire un programme qui demande à l'utilisateur de saisir un entier N et qui affiche le nombre de nombres premiers inférieurs ou égaux à N.

EXERCICE 17

Ecrire un programme qui demande à l'utilisateur de saisir un entier N et qui affiche le N-ième nombre premier.

EXERCICE 18

Ecrire un programme qui demande à l'utilisateur de saisir un entier N et qui affiche la figure suivante.

```
N=1
*
N=2
**
*
N=3
***
**
*
```

et ainsi de suite

EXERCICE 19

Ecrire un programme qui demande à l'utilisateur de saisir un entier N et qui affiche la figure suivante.

```
N=1
*
N=2
**
 *
N=3
***
 **
  *
```

et ainsi de suite.

EXERCICE 20

On considère la suite hongroise : u(0)=a (a entier)
si u(n) pair alors u(n+1)=u(n)/2 sinon u(n+1)=3*u(n)+1

Pour toutes les valeurs a, il existe un entier N tel que u(N)=1 (conjecture admise).

a) Ecrire un programme qui demande à l'utilisateur de taper a et qui affiche toutes les valeurs de u(n) de n=1 à n=N.

b) Ecrire un programme qui demande à l'utilisateur de taper un entier M puis qui cherche la valeur de a comprise entre 2 et M qui maximise la valeur de N. On appelle A cette valeur. Le programme doit afficher la valeur A et la valeur N correspondante.