Répétition d'une région de code avec les boucles

Les boucles en C++ sont utilisées pour répéter l'exécution d'un bloc de code plusieurs fois. Cela s'appelle itérer le code, ou plus simplement faire une boucle. En C++, il existe plusieurs types de boucles, notamment les boucles `for`, `while` et `do-while`.

La boucle `for` permet d'exécuter un bloc de code un nombre prédéterminé de fois. Sa syntaxe est la suivante:

```
for (initialisation; condition; iteration)
{
    // corps de la boucle
}
```

L'expression d'initialisation est évaluée une seule fois, au début de la boucle. La condition de la boucle est ensuite vérifiée et, si elle est vraie, le corps de la boucle est exécuté. Si la condition est fausse, la boucle se termine et l'exécution se poursuit avec l'instruction qui suit la boucle. Après chaque exécution du corps de la boucle, l'expression d'itération est évaluée, et la condition est vérifiée pour décider si la boucle doit continuer.

La boucle `while` poursuit son exécution tant qu'une condition spécifiée est vraie. La condition est vérifiée au début d'une itération, donc si la condition est fausse au départ, aucune itération de la boucle n'est exécutée.

La boucle `do-while` poursuit son exécution tant qu'une condition donnée est vraie. Elle diffère de la boucle `while` en ce que la boucle `do-while` vérifie la condition à la fin d'une itération. Cela implique qu'au moins une itération de boucle est toujours exécutée.

Boucle `for` :

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

int main() {
    for (int i = 0; i < 5; i++) {
        cout << i << endl;
    }
    return 0;
}
```

Ce code affiche les nombres de 0 à 4, un par ligne. La variable `i` est initialisée à 0, et la condition de la boucle est `i < 5`. À chaque itération, `i` est incrémenté de 1. Lorsque `i` atteint 5, la condition de la boucle devient fausse et la boucle se termine.

Boucle `while`:

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

int main() {
    int i = 0;
    while (i < 5) {
        cout << i << endl;
        i++;
    }
    return 0;
}
```

Ce code affiche également les nombres de 0 à 4, un par ligne. La variable `i` est initialisée à 0 avant la boucle, et la condition de la boucle est `i < 5`. À chaque itération, `i` est incrémenté de 1. Lorsque `i` atteint 5, la condition de la boucle devient fausse et la boucle se termine.

Boucle `do-while`:

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

int main() {
    int i = 0;
    do {
        cout << i << endl;
        i++;
    } while (i < 5);
    return 0;
}
```

Ce code affiche également les nombres de 0 à 4, un par ligne. La variable `i` est initialisée à 0 avant la boucle. À chaque itération, `i` est incrémenté de 1. La condition de la boucle est vérifiée à la fin d'une itération, donc lorsque `i` atteint 5, la condition devient fausse et la boucle se termine.

Boucles imbriqués

Voici un exemple de code en C++ qui utilise des boucles imbriquées pour afficher les tables de multiplication de 1 à 10:

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

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

Ce code utilise une boucle `for` imbriquée dans une autre boucle `for`. La boucle externe itère sur les valeurs de `i` de 1 à 10, et la boucle interne itère sur les valeurs de `j` de 1 à 10. À chaque itération de la boucle interne, le produit de `i` et `j` est affiché, suivi d'un espace. À la fin d'une itération de la boucle externe, un saut de ligne est inséré pour passer à la ligne suivante.

Lorsque ce code est exécuté, il affiche les tables de multiplication de 1 à 10, avec chaque ligne contenant les produits d'un nombre par les nombres de 1 à 10. Par exemple, la première ligne contient les produits de 1 par les nombres de 1 à 10, la deuxième ligne contient les produits de 2 par les nombres de 1 à 10, et ainsi de suite.

Voici un autre exemple de code en C++ qui utilise des boucles imbriquées pour résoudre un problème de programmation un peu plus complexe:

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

int main() {
    int n = 5;
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n - i; j++) {
            cout << " ";
        }
        for (int k = 1; k <= 2 * i - 1; k++) {
            cout << "*";
        }
        cout << endl;
    }
    return 0;
}
```

Ce code utilise deux boucles `for` imbriquées pour afficher un triangle isocèle composé d'étoiles (`*`). La boucle externe itère sur les lignes du triangle, et les deux boucles internes itèrent sur les caractères de chaque ligne. La première boucle interne ajoute des espaces pour aligner les étoiles, et la deuxième boucle interne ajoute les étoiles.

Lorsque ce code est exécuté avec `n` égal à 5, il affiche le triangle suivant:

```
    *
   ***
  *****
 *******
*********
```

La boucle for each

La boucle `foreach` en C++ est une boucle `for` basée sur une plage, introduite avec C++. Cette structure de boucle facilite la traversée d'un ensemble de données itérables en éliminant le processus d'initialisation et en parcourant chaque élément plutôt qu'un itérateur.

Voici un exemple simple de code C++ qui utilise une boucle `foreach` pour parcourir un tableau:

```cpp
#include <iostream>

int main() {
    int myArray[] = {1, 2, 3, 4, 5};
    std::cout << "Tableau: ";
    for (int x : myArray) {
        std::cout << x << " ";
    }
    std::cout << std::endl;
    return 0;
}
```

Dans cet exemple, nous avons un tableau `myArray` qui contient cinq entiers. Nous utilisons une boucle `foreach` pour parcourir les éléments du tableau et les afficher. La variable `x` est utilisée pour stocker la valeur de l'élément courant du tableau.

Comment sortir d'une boucle avec break

En C++, vous pouvez utiliser l'instruction `break` pour sortir d'une boucle prématurément. Lorsque l'exécution atteint une instruction `break` à l'intérieur d'une boucle, elle sort immédiatement de la boucle et continue avec l'instruction suivant la boucle.

Voici un exemple de code qui utilise `break` pour sortir d'une boucle `for` :

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

int main() {
    for (int i = 0; i < 10; i++) {
        if (i == 5) {
            break;
        }
        cout << i << endl;
    }
    return 0;
}
```

Dans cet exemple, la boucle `for` itère sur les valeurs de `i` de 0 à 9. Cependant, lorsque `i` atteint la valeur 5, l'exécution atteint l'instruction `break` et sort immédiatement de la boucle. Par conséquent, les nombres de 0 à 4 sont affichés, mais pas les nombres de 5 à 9.

Ce qu'il faut éviter avec les boucles

Il y a plusieurs choses à éviter lors de l'utilisation de boucles en programmation, notamment en C++. Voici quelques exemples:

- **Boucles infinies** : Une boucle infinie est une boucle qui ne se termine jamais. Cela peut se produire si la condition de la boucle est toujours vraie ou si la boucle ne modifie pas les variables utilisées dans la condition de manière à ce que la condition devienne fausse. Les boucles infinies peuvent entraîner des erreurs d'exécution et des blocages du programme.
- **Modification incorrecte des variables de boucle** : Il est important de modifier correctement les variables utilisées dans la condition de la boucle pour éviter les boucles infinies ou les erreurs logiques. Par exemple, si vous utilisez une boucle `for` pour parcourir un tableau, vous devez vous assurer que la variable d'itération est incrémentée ou décrémentée correctement à chaque itération pour éviter de dépasser les limites du tableau.
- **Boucles imbriquées inefficaces** : Lorsque vous utilisez des boucles imbriquées, il est important de réfléchir à l'efficacité de votre code. Les boucles imbriquées peuvent rapidement augmenter la complexité temporelle de votre programme, ce qui peut entraîner des temps d'exécution très longs pour les grandes entrées. Essayez d'optimiser vos boucles imbriquées en réduisant le nombre d'itérations ou en utilisant des algorithmes plus efficaces.
- **Mauvaise gestion des erreurs** : Lorsque vous utilisez des boucles pour lire des données à partir de l'entrée standard ou d'un fichier, il est important de gérer correctement les erreurs. Par exemple, si vous lisez des nombres entiers à partir de l'entrée standard et que l'utilisateur entre une chaîne de caractères non numérique, votre programme peut se bloquer ou se comporter de manière imprévisible. Vous devez vérifier les erreurs d'entrée et gérer ces situations de manière appropriée

Quelle est la différence entre les boucles for et while?

La principale différence entre les boucles `for` et `while` en C++ réside dans leur syntaxe et leur utilisation. La boucle `for` est généralement utilisée lorsqu'on connaît à l'avance le nombre d'itérations à effectuer, tandis que la boucle `while` est utilisée lorsqu'on ne connaît pas le nombre d'itérations à l'avance et que la boucle doit se poursuivre jusqu'à ce qu'une condition spécifique soit remplie.

La syntaxe d'une boucle `for` en C++ est la suivante:

```
for (initialisation; condition; iteration)
{
    // corps de la boucle
}
```

L'expression d'initialisation est évaluée une seule fois, au début de la boucle. La condition de la boucle est ensuite vérifiée et, si elle est vraie, le corps de la boucle est exécuté. Si la condition est fausse, la boucle se termine et l'exécution se poursuit avec l'instruction qui suit la boucle. Après chaque exécution du corps de la boucle, l'expression d'itération est évaluée, et la condition est vérifiée pour décider si la boucle doit continuer.

La syntaxe d'une boucle `while` en C++ est la suivante :

```
while (condition)
{
    // corps de la boucle
}
```

La condition de la boucle `while` est vérifiée au début de chaque itération. Si elle est vraie, le corps de la boucle est exécuté. Si elle est fausse, la boucle se termine et l'exécution se poursuit avec l'instruction qui suit la boucle.

Résumé

Les boucles en programmation permettent de répéter un bloc d'instructions plusieurs fois. Il existe trois types de boucles en C++: `while`, `do-while` et `for`.

La boucle `for` est souvent utilisée lorsque le nombre d'itérations est connu à l'avance. Elle permet d'exécuter un bloc d'instructions un nombre prédéterminé de fois.

Les boucles `while` et `do-while`, en revanche, sont utilisées lorsque le nombre d'itérations n'est pas connu à l'avance. Ces boucles continuent de s'exécuter tant qu'une condition spécifiée est vraie. La différence entre les deux est que la boucle `do-while` vérifie la condition à la fin de chaque itération, garantissant ainsi qu'au moins une itération sera effectuée, tandis que la boucle `while` vérifie la condition au début de chaque itération.

En résumé, les boucles permettent de répéter des instructions plusieurs fois en programmation. Les boucles `for`, `while` et `do-while` offrent différentes options pour contrôler le nombre d'itérations et la condition de sortie de la boucle.