Les variables en C++

Introduction

Nous allons rentrer un bout de code dans Visual Studio pour comprendre comment fonctionne la mémoire d'un ordinateur au niveau binaire.

Comment un ordinateur comprend notre représentation des chiffres et des nombres.

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

string floatToBinary(int num) {
    bitset<8> bits(*reinterpret_cast<unsigned long*>(&num));
    return bits.to_string();
}

int main() {
    int num = 10;
    string binary = floatToBinary(num);
    cout << "L'état binaire de " << num << " est: " << binary << endl;
    return 0;
}
```

C'est pourquoi le langage C a été inventé :)

La RAM (Random Access Memory) est une mémoire volatile qui stocke les données et les programmes en cours d'exécution sur un ordinateur. Chaque emplacement de mémoire dans la RAM a une adresse unique, représentée en hexadécimal.

```
+-----------------+----------------+
| Adresse mémoire | Valeur         |
+-----------------+----------------+
| 0x00000000      |                |
| 0x00000001      |                |
| 0x00000002      |                |
| 0x00000003      |                |
| ...             |                |
| 0x0000XXXX      | variable: 5    |
| ...             |                |
+-----------------+----------------+
```

Dans cet exemple, nous avons déclaré une variable `variable` de type `int` et lui avons affecté la valeur `5`. Cette variable est stockée dans la RAM à une adresse mémoire spécifique, que nous avons représentée par `0x0000XXXX`. La valeur `5` est stockée à cette adresse sous forme binaire, accompagnée de son étiquette `variable`.

Il est important de noter que l'adresse mémoire exacte où la variable est stockée peut varier d'une exécution à l'autre du programme. De plus, la taille en mémoire d'un `int` peut également varier selon l'architecture de l'ordinateur (32 bits ou 64 bits)

Le nom de la variable n'est pas stocké dans la mémoire RAM. Les noms de variables sont utilisés par les programmeurs pour faciliter la lecture et la compréhension du code. Lorsque le code est compilé, les noms de variables sont remplacés par des adresses mémoire qui indiquent où les données sont stockées dans la RAM.

Par exemple, si nous déclarons une variable `int variable = 5;` dans notre code, le compilateur va remplacer le nom `variable` par une adresse mémoire spécifique (par exemple, `0x0000XXXX`) lors de la compilation. La valeur `5` sera ensuite stockée à cette adresse mémoire dans la RAM.

En résumé, les noms de variables ne sont pas stockés dans la mémoire RAM, mais sont remplacés par des adresses mémoire lors de la compilation du code.

Déclarer, typer, afficher une variable

Les variables en C++ sont des conteneurs pour stocker des valeurs de données. En C++, il existe différents types de variables (définis avec différents mots-clés), par exemple[1]:

- `int` - stocke des entiers (nombres entiers), sans décimales, tels que 123 ou -123
- `usigned int` - stocke des entiers positifs (nombres entiers), sans décimales, tels que 0 ou 123
- `double` - stocke des nombres à virgule flottante, avec des décimales, tels que 19,99 ou -19,99
- `char` - stocke des caractères uniques, tels que 'a' ou 'B'. Les valeurs de char sont entourées de guillemets simples
- `string` - stocke du texte, tel que "Hello World". Les valeurs de chaîne sont entourées de guillemets doubles
- `bool` - stocke des valeurs avec deux états: vrai ou faux

Pour créer une variable, vous devez spécifier le type et lui attribuer une valeur. La syntaxe est la suivante[1]:

```
type nomVariable = valeur;
```

- ## Les Types

Le signe égal est utilisé pour attribuer des valeurs à la variable.

Par exemple, pour créer une variable appelée `myNum` de type `int` et lui attribuer la valeur 15, vous pouvez utiliser le code suivant[1]:

```
int myNum = 15;
std::cout << myNum;
```

- **Déclaration**

Vous pouvez également déclarer une variable sans lui attribuer de valeur et lui attribuer une valeur plus tard[1]:

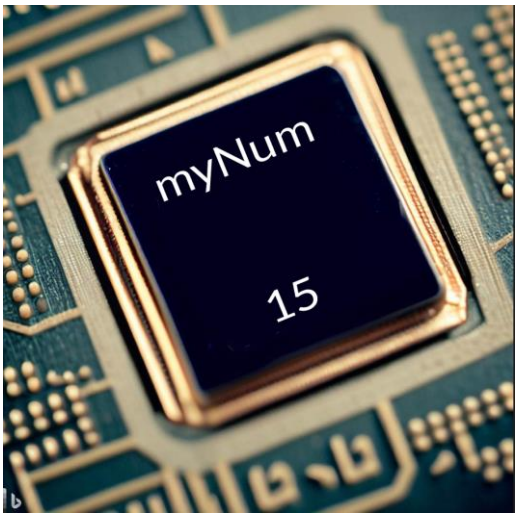
```
int myNum;
myNum = 15;
std::cout << myNum;
```

Copier

Notez que si vous attribuez une nouvelle valeur à une variable existante, elle écrasera la valeur précédente[1].

En résumé, les variables en C++ sont des conteneurs pour stocker des valeurs de données. Il existe différents types de variables pour stocker différents types de données. Pour créer une variable, vous devez spécifier le type et lui attribuer une valeur. Vous pouvez également modifier la valeur d'une variable existante en lui attribuant une nouvelle valeur.

```
+------------------+
|                  |
|   Mémoire RAM    |
|                  |
+------------------+
|                  |
|  Adresse mémoire |
|  unique de       |
|  myNum           |
|                  |
+------------------+
|                  |
|  Valeur de       |
|  myNum           |
|  (par exemple, 5)|
```



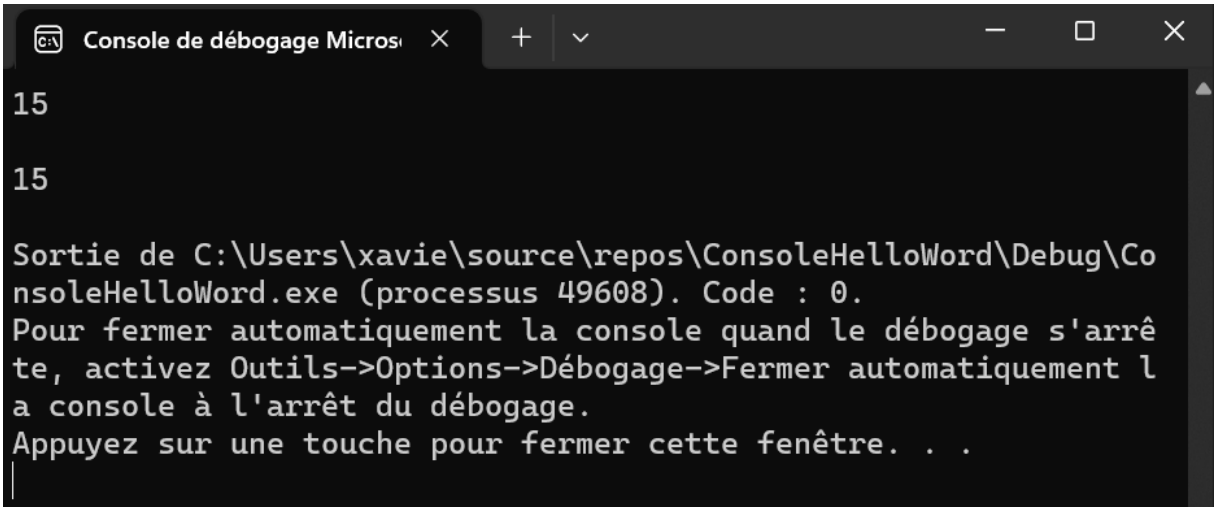
Exemple

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

int main()
{
   int myNum; // déclarer  une variable
   myNum = 15; // initialiser une variable
  cout << myNum << endl << endl; // Afficher la variable

  int myNum2 = 15;  // déclarer et initialiser une variable
  cout << myNum2 << endl; // Afficher la variable

}
```



Déclaration de tous les types de variable

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

int main() {
    int myInt = 5;
    double myDouble = 3.14;
    char myChar = 'a';
    string myString = "Hello";
```

```
    bool myBool = true;

    cout << "int: " << myInt << endl;
    cout << "double: " << myDouble << endl;
    cout << "char: " << myChar << endl;
    cout << "string: " << myString;

    return 0;
}
```

Ce programme déclare cinq variables de différents types (`int`, `double`, `char`, `string` et `bool`) et leur attribue des valeurs. Ensuite, il utilise l'objet `cout` pour afficher chaque variable ligne par ligne, en utilisant l'opérateur d'insertion `<<` pour insérer les valeurs des variables dans le flux de sortie. Le programme affiche les valeurs des variables dans l'ordre dans lequel elles ont été déclarées.

Les références en C++

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

int main() {
    int x = 5;
    int& y = x;

    cout << "x: " << x << endl;
    cout << "y: " << y << endl;

    y = 10;

    cout << "x: " << x << endl;
    cout << "y: " << y << endl;

    return 0;
}
```

Dans ce code, nous déclarons une variable `x` de type `int` et lui attribuons la valeur `5`. Ensuite, nous déclarons une référence `y` à la variable `x` en utilisant la syntaxe `int& y = x;`. Une référence est un alias pour une variable existante, ce qui signifie que lorsque nous utilisons la référence `y`, nous manipulons en fait la variable `x`.

Nous affichons ensuite les valeurs de `x` et de `y` en utilisant l'objet `cout`. Comme `y` est une référence à `x`, les deux valeurs affichées sont les mêmes.

Ensuite, nous modifions la valeur de `y` en lui attribuant la valeur `10`. Comme `y` est une référence à `x`, cela modifie également la valeur de `x`. Nous affichons à nouveau les valeurs de `x` et de `y`, et nous pouvons voir que les deux valeurs ont changé.

En résumé, une référence en C++ est un alias pour une variable existante. Lorsque vous utilisez une référence, vous manipulez en fait la variable à laquelle elle fait référence. Dans l'exemple ci-dessus, nous avons utilisé une référence pour modifier la valeur d'une variable et avons vu comment cela affectait également la variable originale.

le code commenté ligne par ligne

```
#include <iostream> // Inclut la bibliothèque d'entrée/sortie standard
using namespace std; // Indique au compilateur d'utiliser l'espace de noms std


int main() { // Définit la fonction principale du programme
    int x = 5; // Déclare une variable x de type int et lui attribue la valeur 5
    int& y = x; // Déclare une référence y à la variable x


    cout << "x: " << x << endl; // Affiche la valeur de x
    cout << "y: " << y << endl; // Affiche la valeur de y (qui est une référence à
x)


    y = 10; // Modifie la valeur de y (et donc de x) en lui attribuant la valeur
10


    cout << "x: " << x << endl; // Affiche à nouveau la valeur de x (qui a été
modifiée)
    cout << "y: " << y << endl; // Affiche à nouveau la valeur de y (qui a été
modifiée)


    return 0; // Termine la fonction main et renvoie 0 pour indiquer que le
programme s'est terminé correctement
}
```

Les avantages d'utiliser des références

Les références en C++ ont plusieurs avantages par rapport aux pointeurs que l'on verra en fin de cours . Voici quelques exemples:

- **Éviter les copies lors du passage d'arguments** : Lorsque vous passez un argument à une fonction par référence, vous évitez de créer une copie de cet argument, ce qui peut être coûteux en termes de performances pour les grands objets[1].
- **Modifier les paramètres passés dans une fonction** : Lorsque vous passez un argument à une fonction par référence, vous pouvez modifier la valeur de cet argument à l'intérieur de la fonction[1].

- **Syntaxe plus propre par rapport aux pointeurs** : Les références ont une syntaxe plus simple et plus propre que les pointeurs, ce qui peut rendre votre code plus facile à lire et à comprendre[1].
- **Éviter une copie de grandes structures** : Si vous avez une grande structure de données que vous souhaitez passer à une fonction, il peut être coûteux en termes de performances de créer une copie de cette structure. En passant la structure par référence, vous évitez cette copie.
- **Dans For Each Loop pour modifier tous les objets** : Vous pouvez utiliser des références dans une boucle For Each pour modifier tous les éléments d'un conteneur.
- **Dans For Each Loop pour éviter la copie d'objets :** Vous pouvez également utiliser des références dans une boucle For Each pour éviter de créer des copies inutiles des éléments d'un conteneur.

En résumé, les références en C++ ont plusieurs avantages par rapport aux pointeurs, notamment en termes de performances et de lisibilité du code. Elles peuvent être utilisées pour éviter les copies inutiles, modifier les paramètres passés à une fonction et simplifier la syntaxe du code.

Les risques avec les références

Les références en C++ sont considérées comme plus sûres que les pointeurs car elles ne peuvent pas être nulles et ne peuvent pas être réaffectées une fois initialisées. Cependant, il existe certains risques associés à l'utilisation de références. Par exemple, si une référence est initialisée avec un objet temporaire, cet objet sera détruit à la fin de l'instruction et la référence deviendra invalide.

Voici un exemple simple de code C++ qui illustre ce cas :

```
#include <iostream>

int main() {
    const int& x = 3 + 4;
    std::cout << x << std::endl;
    return 0;
}
```

Dans cet exemple, nous avons une référence `x` qui est initialisée avec l'objet temporaire résultant de l'expression `3 + 4`. Cet objet temporaire est détruit à la fin de l'instruction, ce qui rend la référence `x` invalide.

Il est important de noter que cette situation peut entraîner des comportements imprévisibles et des erreurs difficiles à déboguer. Pour éviter cela, il est recommandé d'éviter d'initialiser des références avec des objets temporaires.

De même, si une référence est initialisée avec une variable locale dans une fonction, cette variable sera détruite lorsque la fonction se termine et la référence deviendra invalide.

Voici un exemple simple de code C++ qui illustre ce cas:

```
#include <iostream>

int& foo() {
    int x = 42;
    return x;
}

int main() {
    int& y = foo();
    std::cout << y << std::endl;
    return 0;
}
```

Dans cet exemple, nous avons une fonction `foo` qui déclare une variable locale `x` et renvoie une référence à cette variable. Lorsque la fonction `foo` se termine, la variable `x` est détruite et la référence renvoyée devient invalide.

Il est également important de noter que les références ne sont pas des objets et n'occupent pas nécessairement de l'espace mémoire. Cela signifie que vous ne pouvez pas prendre l'adresse d'une référence ou utiliser des pointeurs pour manipuler des références.

Voici un exemple simple de code C++ qui illustre que vous ne pouvez pas prendre l'adresse d'une référence ou utiliser des pointeurs pour manipuler des références:

```
#include <iostream>

int main() {
    int x = 42;
    int& y = x;

    // Erreur: impossible de prendre l'adresse d'une référence
    // int* p = &y;

    // Erreur: impossible d'utiliser des pointeurs pour manipuler des références
    // int* p = &x;
    // int& z = *p;

    std::cout << y << std::endl;
    return 0;
}
```

Dans cet exemple, nous avons une variable `x` et une référence `y` qui est initialisée avec `x`. Si nous essayons de prendre l'adresse de la référence `y` en utilisant l'opérateur "address-of" (&), nous obtenons une erreur de compilation. De même, si nous essayons d'utiliser des pointeurs pour manipuler des références, nous obtenons également une erreur de compilation.

De plus, les références ne peuvent pas être utilisées dans les tableaux ou les structures de données dynamiques, car elles doivent être initialisées lors de leur déclaration et ne peuvent pas être réaffectées par la suite.

Voici un exemple simple de code C++ qui illustre cette limitation des références:

```
#include <iostream>
#include <vector>

int main() {
    std::vector<int> myVector = {1, 2, 3};

    // Erreur: impossible d'utiliser des références dans un tableau
    // std::vector<int&> myRefVector = {myVector[0], myVector[1], myVector[2]};

    // Erreur: impossible de réaffecter une référence
    // int x = 42;
    // int& y = x;
    // y = myVector[0];

    std::cout << myVector[0] << std::endl;
    return 0;
}
```

Dans cet exemple, nous avons un vecteur `myVector` qui contient trois entiers. Si nous essayons de créer un vecteur de références `myRefVector` en utilisant les éléments de `myVector`, nous obtenons une erreur de compilation. De même, si nous essayons de réaffecter une référence `y` en utilisant un élément de `myVector`, nous obtenons également une erreur de compilation.

En résumé, bien que les références en C++ soient considérées comme plus sûres que les pointeurs, il existe certains risques associés à leur utilisation. Il est important de comprendre ces risques et de les prendre en compte lors de la conception de programmes en C++.

Pour résumer

Une variable est un conteneur pour stocker des données en mémoire. Dans votre code, vous avez déclaré une variable `x` de type `int` et lui avez attribué la valeur `5`.

Les noms de variables peuvent contenir des lettres, des chiffres et des tirets bas, mais doivent commencer par une lettre et ne peuvent pas contenir d'espaces ou d'accents.

C++ est sensible à la casse, donc les noms de variables sont sensibles à la casse.

Vous avez également déclaré une référence `y` à la variable `x`, qui permet d'appeler `x` par un autre nom. Les références sont des alias pour des variables existantes.

Vous avez utilisé `cout` pour afficher les valeurs de `x` et de `y`, puis avez modifié la valeur de `y` (et donc de `x`) en lui attribuant la valeur `10`.

Vous avez ensuite affiché à nouveau les valeurs de `x` et de `y` pour montrer comment les deux ont été modifiées.