Espaces de noms

Namespaces

1. Utilité des espaces de noms

Les espaces de noms (« namespace ») sont utilisés pour :

- **Diviser l'espace global :** un programme peut contenir un ensemble de fonctions, structures et classes. Chaque élément de cet ensemble occupe un espace propre à lui. L'espace restant est nommé l'espace global. Ainsi une variable déclarée en dehors de cet ensemble est dite « variable globale », car elle appartient à l'espace global. Par opposition, une variable déclarée à l'intérieur d'une fonction, on dit qu'elle est locale à cette fonction. On peut diviser à l'infini l'espace global soit à l'aide de fonctions, structures et classes sinon à l'aide « d'espaces de noms »,

- **Résoudre les conflits de noms :** ils permettent en effet d'aider le programmeur à développer de nouveaux composants logiciels sans générer de conflits de noms avec les composants existants. Si l'on veut définir par exemple une nouvelle fonction « rand » permettant de générer des nombres aléatoires entiers, elle va entrer en conflit avec la fonction « rand » définie dans « cstdlib ». Pour pouvoir utiliser notre fonction, il faudra la renommer, en espérant que le nouveau nom ne va pas lui aussi entrer en conflit avec le nom d'une fonction définie par un autre programmeur. Le langage C++ offre « les espaces de noms » pour résoudre les conflits de noms.

2. Déclaration des espaces de noms

- Il faudra utiliser le mot clé réservé « namespace » suivi d'un nom choisi pour représenter l'espace de nom.

```
namespace nom_espace {/* etc. */}
```

```
namespace { int i = 1000; } // espace de nom anonyme
namespace test { int i=2500; } // espace de nom test
namespace UnAutre { // espace de nom UnAutre
        int j=200 ;
        void affiche(){}
        class abc {};
}
```


3. Utilisation des espaces de noms

- Il existe 3 manières pour faire appel au contenu d'un espace de nom :
  - L'opérateur de résolution de portée « : : » :

```
::i; // espace de nom anonyme
test::i;
UnAutre::affiche();
UnAutre::abc zz;
```

- À chaque fois, l'appel est limité au moment de l'utilisation. Si l'on a besoin une seconde fois de la méthode « affiche », il faudra faire appel à elle à l'aide de l'opérateur « :: ».

```
#include <iostream>

namespace { int i = 1000; } // espace de nom anonyme
namespace test { int i=2500; } // espace de nom test

int main() {
    std::cout << "i de l'espace anonyme vaut: " \
         << ::i << endl; // 1000
    std::cout << "i de test vaut: " << test::i \
         << std::endl; //2500
    return 0;
}
```

- La déclaration « using » :

```
using test::i;
using UnAutre::affiche();
using UnAutre::abc;
abc zz;
```

- La variable « i » est déclarée à partir de l'espace nom « test » et elle est connue pour le reste du bloc où elle a été définie, il en est de même pour la fonction « affiche » et la classe « abc ». On peut déclarer par la suite une instance de « abc » sans avoir à préciser où il faudra chercher la classe « abc ».

```
#include <iostream>

namespace test {int i=2500;}
using test::i; // définition de i pour le reste du programme
int main() {
    std::cout << ::i; // 2500
    return 0;
}
```


- Mais, attention à cette erreur où nous avons affaire à une double définition

```
#include <iostream>

int i = 1000; // première définition de i (globale)
namespace test {int i=2500;}
using test::i; // 2e définition de i or i a été déjà définie.
```

- La directive « using » :

```
using namespace test ;
using namespace UnAutre ;
using namespace std;
```

- La directive « using » est une forme radicale permettant de signifier au compilateur que tous les membres d'un espace de nom donné doivent être accessibles.

- Par exemple, les membres de std (« cin », « cout » etc.) sont maintenant accessibles grâce à la directive « using namespace std ». Il en est de même pour la variable « j », la fonction « affiche » et la classe « abc » de l'espace nom « UnAutre ».

```
#include <iostream>

namespace test {int i=2500;int j=200;}

// utilisation de tout le contenu de test
using namespace test;

// utilisation de tout le contenu de std
using namespace std;

int main() {
    cout << "i de test vaut: " << test::i << endl; //2500
    cout << "le même i " << ::i << endl; //2500
    cout << "encore le i " << i << endl; //2500

    return 0;
}
```


4. Quelques particularités des espaces de noms

- Il est possible d'imbriquer les espaces de noms :

```
namespace ABC {
    int i; // ABC::i
    namespace DEF {
        int j; // ABC::DEF::j
    }

}
```

- Il est possible de définir des « alias » d'espaces de noms.

```
namespace UnTresLongNom {}
namespace xyz = UnTresLongNom;
```

  - Il est conseillé d'utiliser un nom long pour l'espace de nom afin d'éviter d'éventuels conflits de noms, et déclarer un alias à cet espace de nom pour faciliter la vie du programmeur.