Espace de noms

Espace de noms

- Les espaces de noms permettent de regrouper des entités telles que les classes, les objets et les fonctions sous un nom
- Format :

```
namespace ident
{
    entites
}
```

Par exemple

```
namespace esp {   int a, b; }
```

Pour accéder à a et b :

esp::a, esp::b

- Objectif : résoudre des conflits de noms (dans les librairies des développeurs)

Espace de noms

```
namespace a
{
  void afficher();   // premier afficher
 }
namespace b
 {
    void afficher();  // deuxième afficher
 }
a::afficher(); // appelle premier
b::afficher(); // appelle deuxième
```

Espace de noms

- Si vous avez déclaré une classe ou des fonctions dans un espace de nom :

Fichier.h

```
namespace esp
{
    class A  {
        public :
        void f1();
        void f2();
    };
    void Fonction1();
}
```

Espace de noms

Fichier.cpp

```
 namespace esp
 {
        void A::f1()
        {
        }
         void A::f2()
        {
        }
         void Fonction1()
        {
        }
 }          ou
void esp::A::f1()
void esp::Fonction1()
```

iostream.h et iostream

- Avant normalisation C++ - **<iostream.h>** seul fichier d'entête

- Normalisation de C++ - **<iostream>** devient en-tête standard pour les entrées-sorties. Toutes ses définitions font partie de l'espace de nommage standard **std**.

- <iostream.h> toujours présent dans un souci de compatibilité

```
#include <iostream>
int main()
{
    std::cout << “bonjour" << std::endl
}
```

iostream.h et iostream

Pour alléger l'écriture – using namespace

```
#include <iostream>
 using namespace std;
int main()
{
    cout << “bonjour" << endl
}
```

Équivalent à

```
#include <iostream.h>
int main()
{
    std::cout << “bonjour" << std::endl
}
```

Espace de noms : exemples

```
#include <iostream>
using namespace std;
  namespace first   {
        int var = 5;
   }
  namespace second   {
        double var = 3.1416;
   }
 int main ()  {
      cout << first::var << endl;
      cout << second::var << endl;
      return 0;
  }
```

5 3.1416

Espace de noms : exemples

```
#include <iostream>
 using namespace std;
  namespace first  {
        int x = 5;  int y = 10;
   }
  namespace second  {
        double x = 3.1416;   double y = 2.7183;
   }
 int main ()  {
      using first::x;    using second::y;
      cout << x << endl;     cout << y << endl;
      cout << first::y << endl;    cout << second::x << endl;
      return 0;
}                                5     2.7183   10        3.1416
```

Espace de noms : exemples

```
#include <iostream>
 using namespace std;
 namespace first  {
        int x = 5;   int y = 10;
  }
 namespace second   {
        double x = 3.1416;  double y = 2.7183;
 }
 int main ()   {
        using namespace first;
        cout << x << endl;    cout << y << endl;
        cout << second::x << endl;  cout << second::y << endl;
        return 0;
 }                          5     10      3.1416      2.7183
```

Espace de noms : exemples

```
#include <iostream>
 using namespace std;
 namespace first  {
      int x = 5;
 }
 namespace second  {
      double x = 3.1416;
 }
 int main ()
 {
      {
          using namespace first;
          cout << x << endl;
      }
```

Espace de noms : exemples

```
    {
     using namespace second;
     cout << x << endl;
    }
 return 0;
}
```

5 3.1416

Chaines de caractères : string

```
#include <iostream>
#include <string>
using namespace std;
string prenom = “Pierre”;
string nom =“Joyce”;

prenom = “James”; // L’affectation est sur-définie
nom = prenom + “ ” + nom;

cin >> nom;
cout << nom;

if (nom[0] == ‘A’) …
if (nom < prenom) . . .// L ’opérateur ‘ < ‘  est sur-défini
if (nom == prenom) . . . // L ’opérateur ‘ == ‘  est sur-défini

cout << “Longueur = ” << nom.length();
```

Compatibilité et uniformisation

- Librairies de C, librairies de C++
- Avec la nouvelle norme, conservation des anciennes librairies de C++ et placement des nouvelles dans d'autres fichiers

  Sans « .h » → nouvelle norme <iostream>

- Cas de « string.h » et « string »

- Avec la nouvelle normalisation, les anciennes librairies de C perdent leur extension, mais « c » devant pour éviter des conflits

  Exemple :

  #include <stdlib.h> #include <cstdlib>

  devient

  #include <stdio.h> #include <cstdio>

Résumé

- <iostream.h> : librairie des entrées/sorties C++ de l'**ancienne** norme (pas d'espace de noms)
- <iostream> : librairie des entrées/sorties C++ de la **nouvelle** norme (espace de noms std)
- <string.h> : librairie des chaînes de caractères C de l'**ancienne** norme
- <cstring> : librairie des chaînes de caractères C de la **nouvelle** norme (pas d'espace de noms pour les librairies C)
- <string> : librairie des chaînes de caractères C++ de la **nouvelle** norme (espace de noms std)

Les exceptions

La gestion des erreurs

- Idée fondamentale
  - En face d'un problème qu'elle ne sait pas résoudre
    - une fonction lance une exception : **throw**
    - une fonction de niveau supérieure pourra l'intercepter : **catch**
  - Une exception est un **objet d'un certain type** pouvant véhiculer des informations

```
class ErreurIndice {    // Classe Exception
   public:       int  hors;
                 ErreurIndice(int i) { hors = i; } };
class Vecteur {
      int   dim;     double  *x;
   public:
      Vecteur(int d) { x = new double[dim = d]; };
      ~Vecteur()     { delete x; };
      double& operator[](int i) {
         if ((i < 0) || (i >= dim)) { ErreurIndice e(i); throw(e); }
            return(x[i]); } };
```

La gestion des exceptions

- Comment intercepter une exception ?
  - Le programmeur peut définir **des zones dans lesquelles les exceptions sont interceptées**
    - **Une telle zone débute par le mot clef try**
    - **Elle est suivie d'un bloc d'instructions qui doit lui même être suivi d 'un ou plusieurs gestionnaires d'exceptions**
  - **Un gestionnaire d'exceptions**
    - **débute par le mot clef catch suivi**
    - **d'un type d'exception entre parenthèses puis**
    - **d'un bloc d'instructions à exécuter si l'exception que le gestionnaire reçoit est du type spécifié dans son en-tête**

La gestion des exceptions

- Comment intercepter une exception ?
  - **Exemple de zone d'interception et de gestionnaire**

```
void main() {
   try {  // Zone dans laquelle les exceptions sont interceptées
      Vecteur v(3);
      v[10] = 5;     }
   catch(ErreurIndice e) { // Gestionnaire d'exceptions
      cerr << "Dépassement d'indices : " << e.hors;
     exit(-1); }}
```

La mise en œuvre des gestionnaires

- Choix du gestionnaire
  - Un gestionnaire d'exceptions peut être suivi d'un ou plusieurs gestionnaires d'exceptions
  - Les gestionnaires sont examinés en séquence
    - Le premier dont le type correspond à l'exception lancée l'intercepte
    - Un gestionnaire par défaut peut être placée en fin de liste **catch(. . .)**
  - Dans le cas où une interception lancée n'est pas interceptée (pas de bloc try), l'exécution du programme avorte
    - **la fonction terminate est appelée, fonction qui appelle à son tour la fonction abort**
    - **on peut écrire sa propre fonction terminate en fournissant son adresse à la fonction set_terminate**

La mise en œuvre des gestionnaires

- Traitement d'une exception
  - Lorsqu'une exception est lancée, tous les objets définis localement dans le bloc try sont détruits par invocation des destructeurs respectifs
  - Une fois qu'un gestionnaire a intercepté une exception et terminé l'exécution du bloc d'instructions associés, **l'exécution du programme continue avec la première instruction située après le dernier gestionnaire du bloc try en question**

Un exemple complet

- Définitions des différentes classes d'exceptions

```
#include <iostream>
#include <new>

using namespace std;

class ErreurIndice {
   public:
      int   hors;
      ErreurIndice(int i) { hors = i; }
};


class ErreurDimension {
   public:
      int   dim;
      ErreurDimension(int i) { dim = i; }
};
```

Un exemple complet

- Utilisation des différentes classes d'exceptions

```
class Vecteur {
      int          dim;
      double  *x;
   public:
      Vecteur(int d) {
          if (d < 0) { ErreurDimension e(d); throw(e); }
         x = new double[dim = d]; };
      ~Vecteur() { delete x; };
      double& operator[](int i) {
          if ((i < 0) || (i >= dim))
                { ErreurIndice e(i); throw(e); }
          return(x[i]); }
};
```

Un exemple complet

- Gestion des exceptions

```
void main() {
    try {
        Vecteur v(3);
        v[1] = 3;
        Vecteur w(-2);
        w[-1] = 8;            // va provoquer une exception Dépassement d'indices
        // Vecteur z(-5);  // va provoquer une exception erreur de dimension
    }
    catch(ErreurIndice e) { // Gestionnaire d'exceptions
        cerr << "Dépassement d'indices : " << e.hors << endl;  }
    catch(ErreurDimension e) { // Gestionnaire d'exceptions
        cerr << "Dimension incorrecte : " << e.dim << endl;  }
    catch(bad_alloc a) {   // En cas de mémoire saturée
        cerr << "Plus de mémoire disponible  "; exit(-1); }
    catch(...) {
        cerr << "Exception non prévue  "; exit(-1); }

    cout << "On continue ici en cas d'indices ou dimension incorrecte\n"; }
```

```
Dimension incorrecte : -2
On continue ici en cas de . . .
```