Université de Lyon – Université Claude Bernard Lyon 1
Master d'Informatique

Programmation Générique et le langage C++

Norme ISO

Raphaëlle Chaine
raphaelle.chaine@liris.cnrs.fr
2012-2013

Namespace

- Utilisation de différents modules et bibliothèques dans un programme
- Problème dit de « pollution de l'espace de noms » :
  Un même identificateur peut être utilisé par plusieurs modules ou bibliothèques
  – Risque d'ambiguité

- Concept d'espace de noms en C++ :
  donner un nom à un espace de déclaration

```
namespace mon_module
{//déclarations usuelles
  extern double taux;
  double conversion(double);
}
```


- Pour se référer à des identificateurs définis dans un espace de noms, on utilise l'opérateur :: de résolution de portée

```
double mon_module::taux=6.5; //définition
std::cout << mon_module::conversion(1);
```

On dit aussi que l'on se réfère à l'identificateur taux declaré dans la portée de mon_module

- A l'interieur de mon_module, on utilise directement le nom taux

```
double mon_module::conversion(double a)
{
 return a*taux; // mon_module::taux
}
```


- L'espace des déclarations globales d'un programme est aussi un espace de noms dit portée globale

  ::x fait référence à l'identificateur x de la portée globale

- using_declaration :
  permet de faire entrer (connaître) un identificateur dans la portée courante

```
using mon_module::taux;
std::cout << taux;
```

- exemple :
  Si on fait entrer taux dans la portée globale alors ::taux mon_module::taux deviennent des écritures équivalentes
- Attention : il ne doit pas y avoir d'autre taux dans la portée courante

206

- using_directive
  using namespace mon_module;
  permet de rendre visibles les noms de mon_module

```
using namespace mon_module;
std::cout << conversion(3);
```

- Risques d'ambiguïtés si plusieurs espaces de noms comportant des identificateurs identiques sont rendus visibles

207

- using namespace mon_module;
  //espace de nom déclarant taux
  using namespace son_module;
  //espace de nom déclarant taux
  std::cout << taux; //appel ambigu
- Faire appel à l'opérateur de résolution de portée

208

- Danger des using-directives :
  Le compilateur signale
  - les ambiguïtés entre identifiants des espaces de noms rendus visibles,
  - mais pas les surcharges de fonctions à travers les espaces de noms!

209

```
using namespace mon_module;
int conversion(int s){return s*taux;}
int main()
{ int taux=9;
  std::cout<< taux << " " <<::taux
           << " " << mon_module::taux
          << " " << conversion(1) << " "
           << conversion(1.5) << std::endl;
  return(0);
}
// 9  6.5  6.5  6  9.75
```

210

- Le fonctionnement d'un programme satisfaisant peut changer après introduction (totalement indépendante) d'une nouvelle fonction dans un des espaces de noms rendu visible
- il est parfois plus prudent d'utiliser une using_declaration qu'une using_directive!

211

- Les espaces de noms
  - peuvent être emboîtés
  - peuvent être étendus

```
namespace mon_module
{ ...
   namespace sous_module
   { ...
   }
}
namespace mon_module
{ ...
}
```

212