CS2720 Practical Software Development

Namespace Tutorial Spring 2011

Instructor: Rex Forsyth
Office: C-558
E-mail: forsyth@cs.uleth.ca
Tel: 329-2496

Tutorial Web Page: http://www.cs.uleth.ca/~forsyth/cs2720/lab/lab.html

Name Spaces

- A namespace is a mechanism in standard C++ that allows logical grouping of names
- Any name declared in the global scope must be unique
- In large-scale development, using libraries from different vendors increases the likelihood of name collision
- C++ namespace mechanism can be used to avoid the *global namespace pollution* problem. To place an item in a namespace, use the **namespace** keyword and enclose the item in braces.

```
namespace howToProgram
{
  class matrix {/* ... */};
  void inverse(matrix &);
}
```

The namespace `howToProgram` is a user-declared namespace

- Each user-declared namespace represents a distinct namespace scope.
- A user-declared namespace may contain other nested namespace definitions as well as declarations or definitions for functions, objects, templates, and types.
- The entities declared within a namespace are called *namespace members* .
- Each name in a namespace must refer to a unique entity within that namespace.
- The full name of a namespace member is its name compounded, or *qualified* by, the name of its namespace. For example, the name of the matrix class declared in namespace `howToProgram` is `howToProgram::matrix`.
- The members of a namespace can be used in a program using their qualified names:

```
howToProgram::matrix someFunc(howToProgram::matrix& m)
{
  // ...
  howToProgram::inverse(m);
  return m;
}
```

- Placing items in a namespace is cumulative.

```
namespace howToProgram
{
  class matrix {/* ... */};
  const double pi = 3.14159;
}

namespace howToProgram
{
  void inverse(matrix &);
  matrix operator+(const matrix &m1, const matrix &m2)
  { /* ... */ }
}
```

Using Namespace Members

- Using Directives: Are used to introduce all the items from a specific namespace into the current scope so that they can be referred to by their unqualified form.

```
using namespace namespace_name;
```

  where `using` and `namespace` are `C++` keywords, and `namespace_name` is a previously defined namespace name.

- Using Declarations: Are used to introduce an individual namespace item into the current scope so that it can be referred to by its unqualified form.

```
namespace howToProgram
{
  namespace matrixLib
  {
    class matrix {/* ... */};
    // ..

   }
}

//using declaration for the namespace member matrix
using howToProgram::matrixLib::matrix;
```

  introduces the name `matrix` into the current scope

- Namespace Aliases: Are used to associate a shorter synonym with a namespace name

```
namespace  UniversityOfLethbridge
{
  /* ... */
}
```

  can be associated with a shorter synonym as follows.

```
namespace UofL = UniversityOfLethbridge
```

Namespace Summary

- Any items that are not explicitly placed in a named namespace are put into the **Global Namespace**. This is what we have been doing so far. The libraries **myio** and **utility** along with all our classes have just been added to the global namespace. These items can then be used at any time without qualification. BUT it increases the possibility of a name collision.

- The C++ Standard Library is in the **std** namespace. So far we have been adding all of these to the global namespace by using the *using directive*
  `using namespace std;`.
  This allows us to use all the items without qualification but adds to the global pollution.

- We can place items into a named namespace using the syntax
  `namespace <name>`
  and enclosing the item in braces. Each placed item is appended to the existing namespace. Many items (function, constants, libraries and classes) may be placed in a namespace.

- We can use items from a namespace in one of three different ways:
  1. Introduce the entire namespace by using the *using directive*
     `using namespace name;`
     This will bring all items in the specified namespace into the global namespace and they can be used without qualification. More pollution!!
  2. Introduce a specific item from the namespace by using the *using declaration*
     `using namespaceName::itemName;`
     Now only this item may be used without qualification. Less pollution.
  3. Do not use the keyword `using`. Just qualify each item with the name of the namespace it is in. e.g `std::cout`. No pollution.
- **DO NOT** put a *using directive* in a header file. If you need some item from another namespace in a header file, use the second or third form above.
- Namespaces should have long descriptive names to avoid possible namespace name collisions. Use the `alias` feature to create a smaller name in the client.

```
#ifndef MYIO_H
#define MYIO_H

#include <fstream>
namespace developedByRex
{
//*************************************************************************
// function to clear the cin stream up to & including the next new line
//*************************************************************************
void clrStream();
     .
     .
     .
}   // end of namespace
```

```
#include <iostream>
#include <fstream>
#include <string>
using std::cin;
using std::string;

#include "myio.h"

namespace developedByRex
{
//*************************************************************************
// function to clear the cin stream up to & including the next new line
//
// Post-condition -- all data up to and including the next new line char
//                   is removed from cin
//*************************************************************************
void clrStream() {

   while (cin.peek() != '\n') cin.ignore();
   cin.ignore();
}
      .
      .
      .

} // end of namespace
```

```
namespace developedByRex
{
   const double PI = atan(1.0)*4.0;
   const double TOLERANCE = 1.0E-8;
      .
      .
      .
//*****************************************************************
// utility function to check equality of floating point numbers
//
// Parameter usage :
//     x,y  -- import the numbers to compared
//
// Post-condition -- function returns true if the numbers are within
//                   TOLERANCE of each other; false otherwise
//*****************************************************************
bool dEqual(double x, double y) { return std::fabs(x-y) < TOLERANCE; }
      .
      .
      .
} // end of namespace
```

Suppose we now write a program which to uses `clrStream` from the **myio** library and `PI` and `dEqual` from the **utility** library both of which are in the *developedByRex* namespace.

```
#include <iostream>
using std::cout;
using std::cin;
using std::endl;
#include "myio.h"
#include "utility.h"
namespace dbr = developedByRex;
using dbr::clrStream;    // introduce clrStream into the global namespace

int main() {
   int age;
   cout << "Enter your age -- ";
   cin >> age;
   clrStream();
   double balance;
   cout << "Enter the balance $";
   cin >> balance;
   clrStream();
   if (dbr::dEqual(balance,1234.56)) cout << "great"; else cout << "Too bad";
   cout << endl;
   cout << "PI is " << dbr::PI << endl;
   return 0;
}
```