xLib.h

```
#ifndef XLIB_H
#define XLIB_H

// introducing xLib as a namespace
namespace xLib {

   int doSomething(int);
   const int minimum = 100;
}

// ...

// expanding xLib again
namespace xLib {
   class Foo
   {
   private: int a;
   public: Foo(int);
           void print() const;
   };
}
#endif
```

xLib.cpp

```
#include<iostream>
using std::cout;
using std::endl;
#include "xlib.h"

// expanding xLib again
namespace xLib {
   int doSomething(int m) {
      cout << "doSomething in xLib" << endl;
      return minimum - m;
   }
   Foo::Foo(int a) : a(a) { }
}

// ...

// expanding xLib again
namespace xLib {
  void Foo::print() const { cout << "xLib::Foo::a " << a << endl; }
}
```

yLib.h

```
#ifndef YLIB_H
#define YLIB_H

// introducing yLib as a namespace
namespace yLib {
   int doSomething(int);
}
// ...
// expanding yLib again
namespace yLib {
   const int minimum = 200;
}
// ...
// expanding yLib again
namespace yLib {
   class Foo
   {
   private: int a;
   public: Foo(int);
           void print() const;
   };
}
#endif
```

yLib.cpp

```
#include<iostream>
using std::cout;
using std::endl;
#include "ylib.h"

// expanding yLib again
namespace yLib {
   int doSomething(int m) {
      cout << "doSomething in yLib" << endl;
      return minimum - m;
   }
}

// ...
// expanding yLib again
namespace yLib {
   Foo::Foo(int a) : a(a) { }
   void Foo::print() const { cout << "yLib::Foo::a " << a << endl; }
}
```

xyLib_Test_Driver.cpp

```cpp
#include<iostream>
#include "ylib.h"
#include "xlib.h"

int main() {

   {// a local scope
      using namespace xLib; // the using directive makes all the names available.
      std::cout << doSomething(10) << std::endl;
      std::cout << minimum << std::endl;
   }

   {// another local scope
      using yLib::doSomething; // A using declaration, makes a single name available.
      std::cout << doSomething(20) << std::endl;

      using yLib::minimum; // A using declaration, makes a single name available.
      std::cout << minimum << std::endl;
   }

   xLib::Foo f1(1000); // no ambiguity when you use the scope-resolution operator
   f1.print();

   yLib::Foo f2(2000); // no ambiguity when you use the scope-resolution operator
   f2.print();

   return 0;
}
```

Output

```
doSomething in xLib
90
100
doSomething in yLib
180
200
xLib::Foo::a 1000
yLib::Foo::a 2000
```