Object-Oriented Programming and Data Structures

COMP2012: Namespace

Prof. Brian Mak
Prof. C. K. Tang

Department of Computer Science & Engineering
The Hong Kong University of Science and Technology
Hong Kong SAR, China

Motivation

Suppose that you want to use two libraries, each consisting of a bunch of useful classes and functions, but some of them have the same name.

```
/* File: gnu-utils.h */
class Stack { /* incomplete */ };
class Some_Class { /* incomplete */ };
void firefox( ) { /* incomplete */ };
int func(int) { /* incomplete */ };

/* File: ms-utils.h */
class Stack { /* incomplete */ };
class Other_Class { /* incomplete */ };
void internet_explorer( ) { /* incomplete */ };
int func(int) { /* incomplete */ };
```

Motivation ..

Even if you don't use `Stack` and `func`, you run into troubles:

- compiler will complain about multiple definitions of `Stack`;
- linker will complain about multiple definitions of `func`.

```
#include "gnu-utils.h"                          /* File: use-utils.cpp */
#include "ms-utils.h"
enum OS { Linux, MSWindows, MacOS } choice;

int main( )
{
    Some_Class sc;
    Other_Class oc;

    if (choice == Linux)
        firefox( );
    else if (choice == MSWindows)
        internet_explorer( );
    return 0;
}
```

Solution: namespace

If the library writers would have used namespaces, multiple names wouldn't be a problem.

```
/* File: gnu-utils-namespace.h */
namespace gnu
{
    class Stack { /* incomplete */ };
    class Some_Class { /* incomplete*/ };
    void firefox( ) { /* incomplete */ };
    int func(int) { /* incomplete */ };
}

/* File: ms-utils-namespace.h */
namespace microsoft
{
    class Stack { /* incomplete */ };
    class Other_Class { /* incomplete */ };
    void internet_explorer( ) { /* incomplete */ };
    int func(int) { /* incomplete */ };
}
```

Namespace Alias & Scope Operator ::

You refer to names in a namespace with the scope resolution operator.

```
#include "gnu-utils-namespace.h"  /* File: utils-namespace.cpp */
#include "ms-utils-namespace.h"
namespace ms = microsoft;                    // Namespace alias
enum OS { Linux, MSWindows, MacOS } choice;

int main( ) {
    gnu::Some_Class sc; gnu::Stack gnu_stack;
    ms::Other_Class oc; ms::Stack ms_stack;
    int i = ms::func(42);

    if (choice == Linux)
        gnu::firefox( );
    else if (choice == MSWindows)
        ms::internet_explorer( );
    return 0;
}
```

using Declaration

If you get tired of specifying the namespace every time you use a name, you can use a using declaration.

```
#include "gnu-utils-namespace.h"          /* File: utils-using.cpp */
#include "ms-utils-namespace.h"
namespace ms = microsoft;                          // Namespace alias
using gnu::Some_Class; using gnu::Stack;
using ms::Other_Class; using ms::func;

int main( )
{
    Some_Class sc;                          // Refer to gnu::Some_Class
    Other_Class oc;                         // Refer to ms::Other_Class
    Stack gnu_stack;                          // Refer to gnu::Stack
    ms::Stack ms_stack;

    int i = func(2);                                  // Refer to ms::func
    return 0;
}
```

Ambiguity With using Declarations

You can also bring all the names of a namespace into your program at once, but make sure it won't cause any ambiguities.

```
#include "gnu-utils-namespace.h"      /* File: utils-using-err.cpp */
#include "ms-utils-namespace.h"
namespace ms = microsoft;                          // Namespace alias
using namespace gnu;
using namespace ms;

int main( )
{
    Some_Class sc;                    // Refer to gnu::Some_Class
    Other_Class oc;                   // Refer to ms::Other_Class

    Stack S;                                  // Error: ambiguous;
    ms::Stack ms_stack;                                       // OK
    gnu::Stack gnu_stack;                                     // OK
    return 0;
}
```

Namespace std

```
#include <iostream>                                    /* File: using-std.cpp */
#include <vector>
#include <algorithm>
using namespace std;

int main( )
{
    vector<int> v;
    vector< int >::iterator it;

    v.push_back(63);                          // ... push_back some more int's
    it = find( v.begin( ), v.end( ), 42 );

    if ( it != v.end( ) )
        cout << "found 42!" << endl;

    return 0;
}
```

How Should We Declare Namespaces?

- Functions and classes of the standard library (`string`, `cout`, `isalpha()`, ...) and the STL (`vector`, `list`, `foreach`, `swap`, ...) are all defined in namespace std.
- Here, we bring all the names that are declared in the three header files into the global namespace.
- Although the previous program works, it is considered bad practice to declare the namespace std globally.
- It is better to introduce only the names you really need, or to qualify the names whenever you use them.
- Although this takes more typing effort, it is also immediately clear which functions and classes are from the standard (template) library, and which are your own.
- A combination of using declarations and explicit scope resolution is also possible; this is mostly a matter of taste.

Explicit Use of using Declaration

```
#include <iostream>                    /* File: std-individual-using.cpp */
#include <vector>
#include <algorithm>
using std::vector; using std::find;
using std::cout; using std::endl;

int main( )
{
    vector<int> v;
    vector< int >::iterator it;
    v.push_back(63);                   // ... push_back some more int's

    it = find( v.begin( ), v.end( ), 42 );
    if ( it != v.end( ) )
        cout << "found 42!" << endl;
    return 0;
}
```

Explicit Use of namespace Per Object/Function

```
#include <iostream>                    /* File: std-per-obj-using.cpp */
#include <vector>
#include <algorithm>

int main( )
{
    std::vector<int> v;
    std::vector< int >::iterator it;
    v.push_back(63);                   // ... push_back some more int's

    it = std::find( v.begin( ), v.end( ), 42 );
    if ( it != v.end( ) )
        std::cout << "found 42!" << std::endl;
    return 0;
}
```