COMP151

Namespaces

Motivation

Suppose that you want to use two libraries with a bunch of useful classes and functions, but some names collide:

```
// File: "gnutils.h"
class Stack { ... };
class Some_Class { ... };
void gnome();
int func(int);

// File: "msutils.h"
class Stack { ... };
class Other_Class { ... };
void windows200();
int func(int);
```

Motivation . . .

Even if you do not use `Stack` and `func`, you run into trouble:

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

```
#include "gnutils.h"
#include "msutils.h"

int main() {
    Some_Class sc;
    Other_Class oc;
    . . .
    if (choice == LINUX)
        gnome();
    else if (choice == MSWINDOWS)
        windows2000();
    return 0;
}
```

Solution: namespace

If the library writers would have used namespace, multiple names would not be a problem.

```
// File: "gnutils.h"
namespace gnu {
    class Stack { ... };
    class Some_Class { ... };
    void gnome();
    int func(int);
}

// File: "msutils.h"
namespace microsoft {
    class Stack { ... };
    class Other_Class { ... };
    void windows2000();
    int func(int);
}
```

Namespace Alias & Scope Operator ::

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

```
#include "gnutils.h"
#include "msutils.h"
namespace ms = microsoft;                              // namespace alias

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::gnome();
    else if (choice == MSWINDOWS)
        ms::windows2000();
    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 "gnutils.h"
#include "msutils.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(42);                    // 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 will not cause any ambiguity.

```
#include "gnutils.h"
#include "msutils.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

- Functions and classes of the standard library (`string`, `cout`, `isalpha()`, `...`) and the STL (`vector`, `list`, `for_each`, `swap`, `...`) are all defined in `namespace std`.
- Although the following works, it is considered a bad practice.
- Here, we bring *all* the names that are declared in the three header files into the *global namespace.*

Example: namespace std

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

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

    v.push_back(63);                    // ... push_back some more ints
    it = find(v.begin(), v.end(), 42);
    if (it != v.end())
        cout << "Found 42!" << endl;

    return 0;
}
```

Explicit Use of using Declaration

It is better to introduce only the names you really need, or to qualify the names whenever you use them.

```
#include <iostream>
#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 ints
    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>
#include <vector>
#include <algorithm>

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

    v.push_back(63);                          // ... push_back some more ints
    it = std::find(v.begin(), v.end(), 42);
    if (it != v.end()) std::cout << "Found 42!" << std::endl;
    return 0;
}
```

- 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.

Final Remarks

- A combination of using declarations and explicit scope resolution is also possible; this is mostly a matter of taste.
- In `g++`, the classes and functions of the standard library and the STL are not defined in `namespace std`, but in the global namespace.
- That's why you get away with forgetting using declarations.
- However, this will most likely change in the future, so you better get used to it.
- VC++ already does it the right way.