4.3c — Using statements

BY ALEX ON NOVEMBER 9TH, 2016 | LAST MODIFIED BY ALEX ON MARCH 13TH, 2018

If you're using the standard library a lot, typing "std::" before everything you use from the standard library can become repetitive. C++ provides some alternatives to simplify things, called "using statements".

The using declaration

One way to simplify things is to utilize a **using declaration** statement.

Here's our Hello world program again, with a using declaration on line 5:

```
1  #include <iostream>
2
3  int main()
4  {
5     using std::cout; // this using declaration tells the compiler that cout should resolve to std::cout
6     cout << "Hello world!"; // so no std:: prefix is needed here!
7     return 0;
8  }
```

The using declaration "using std::cout;" tells the compiler that we're going to be using the object "cout" from the std namespace. So whenever it sees "cout", it will assume that we mean "std::cout". If there's a naming conflict between std::cout and some other use of cout, std::cout will be preferred. Therefore on line 6, we can type "cout" instead of "std::cout".

This doesn't save much effort in this trivial example, but if you are using cout a lot inside of a function, a using declaration can make your code more readable. Note that you will need a separate using declaration for each name you use (e.g. one for std::cout, one for std::cin, and one for std::endl).

Although this method is less explicit than using the "std::" prefix, it's generally considered safe and acceptable.

The using directive

Another way to simplify things is to use a **using directive** statement. Here's our Hello world program again, with a using directive on line 5:

```
1  #include <iostream>
2
3  int main()
4  {
5     using namespace std; // this using directive tells the compiler that we're using everything in the st
6  d namespace!
7     cout << "Hello world!"; // so no std:: prefix is needed here!
8     return 0;
   }
```

The using directive "using namespace std;" tells the compiler that we want to use *everything* in the std namespace, so if the compiler finds a name it doesn't recognize, it will check the std namespace. Consequently, when the compiler encounters "cout" (which it won't recognize), it'll look in the std namespace and find it there. If there's a naming conflict between std::cout and some other use of cout, the compiler will flag it as an error (rather than preferring one instance over the other).

For illustrative purposes, let's take a look at an example where a using directive causes ambiguity:

```
#include <iostream>

namespace a
{
    int x(10);
}

namespace b
{
    int x(20);
}

int main()
{
    using namespace a;
    using namespace b;

    std::cout << x << '\n';

    return 0;
}
```

In the above example, the compiler is unable to determine whether the x in main refers to a::x or b::x. In this case, it will fail to compile with an “ambiguous symbol” error. We could resolve this by removing one of the using statements, or using an explicit a:: or b:: prefix with x.

Here’s another more subtle example:

```
#include <iostream> // imports the declaration of std::cout

int cout() // declares our own "cout" function
{
    return 5;
}

int main()
{
    using namespace std; // makes std::cout accessible as "cout"
    cout << "Hello, world!"; // uh oh!  Which cout do we want here?  The one in the std namespace or the one we defined above?

    return 0;
}
```

In the above example, the compiler is unable to determine whether we meant std::cout or the cout function we’ve defined, and again will fail to compile with an “ambiguous symbol” error. Although this example is trivial, if we had explicitly prefixed std::cout like this:

```
    std::cout << "Hello, world!"; // tell the compiler we mean std::cout
```

or used a using declaration instead of a using directive:

```
    using std::cout; // tell the compiler that cout means std::cout
    cout << "Hello, world!"; // so this means std::cout
```

then our program wouldn’t have any issues in the first place.

Limiting the scope of using declarations and directives

If a using declaration or directive is used within a block, the using statement applies only within that block (it follows normal scoping rules). This is a good thing, as it reduces the chances for naming collisions to occur just within that block. However, many new programmers put using directives into the global scope. This pulls all of the names from the namespace directly into the global scope, greatly increasing the chance for naming collisions to occur. This is considered bad practice.

Rule: Avoid “using” statements outside of a function (in the global scope).

There’s some debate about whether using directives are okay to use within functions, where their impact is limited. However, our take is that using declarations are the better choice if using statements are desired, and that using directives should be avoided entirely.

Suggestion: We recommend you avoid “using directives” entirely.

Cancelling or replacing a using statement

Once a using statement has been declared, there’s no way to cancel or replace it with a different using statement within the scope in which it was declared.

```
int main()
{
    using namespace Foo;

    // there's no way to cancel the "using namespace Foo" here!
    // there's also no way to replace "using namespace Foo" with a different using statement

    return 0;
} // using namespace Foo ends here
```

The best you can do is intentionally limit the scope of the using statement from the outset using the block scoping rules.

```
int main()
{
    {
        using namespace Foo;
        // calls to Foo:: stuff here
    } // using namespace Foo expires

    {
        using namespace Goo;
        // calls to Goo:: stuff here
    } // using namespace Goo expires

    return 0;
}
```

Of course, all of this headache can be avoided by explicitly using the scope resolution operator (::) in the first place.

4.4 -- Implicit type conversion (coercion)

Index

4.3b -- Namespaces

Share this:

Facebook Twitter Google Pinterest

C++ TUTORIAL | PRINT THIS POST

89 comments to 4.3c — Using statements

« Older Comments 1 2

Suprith
July 9, 2018 at 11:37 pm · Reply

#include <iostream> // imports the declaration of std::cout

int cout() // declares our own "cout" function
{
    return 5;
}