Outline

- Using declaration
- Using directive
- Name qualification
- Nested namespaces
- Unnamed namespace
Namespaces

- Namespace is an organization unit in your code eg. std namespace
- Allows functions with the same name to co-exist
  - Eg. could have your own version of cout that is different from std::cout

Using directive

- Example of a using directive
  ```
  using namespace std;
  ```
- Pulls everything into the global namespace when you use it
- Scoping rules: check local namespace, then enclosing namespace, then all the way up to the global scope
- If not in global scope, declare an error
Using directive

- Note: we will not use using directives
- Considered bad practice. Why?
  Potential for a lot of namespace conflicts

Using Declarations

```cpp
#include <iostream>
using std::cout;
using std::endl;

int main(int argc, char **argv)
{
    cout << "Hello world" << endl;
    return 0;
}
```

These are “using declarations”.
Notice how we use cout and endl
Using Declarations

```cpp
#include <iostream>

int main(int argc, char **argv)
{
    std::cout << "Hello world" << std::endl;
    return 0;
}
```

Without “using declarations” we need to fully qualify cout and endl. By “fully qualify”, we mean we preface cout and endl with std::

Summary

- Use “using declarations” or fully qualify things from namespaces
- Don’t use the “using directive”


Defining your own namespace

```cpp
#include <iostream>

namespace mymath{
    int square(int x) {
        return x*x;
    }
}

int main(int argc, char **argv)
{
    std::cout << mymath::square(10) << std::endl;
    return 0;
}
```

This creates the mymath namespace and puts the square() function in it

To use square() you must add using mymath::square() or qualify square with mymath::

- If you have multiple files with stuff inside a mymath namespace, this “stuff” all belongs to the same mymath namespace
Nested Namespaces

```cpp
#include <iostream>

namespace mymath{
    namespace stuff {
        int square(int x) {
            return x*x;
        }
    }
}

int main(int argc, char **argv)
{
    std::cout << mymath::stuff::square(10) << std::endl;
    return 0;
}
```

To use `square`, you now need to refer to it as `mymath::stuff::square()`.

Unnamed Namespace

```cpp
namespace {
    const int secret_code = 42;
}
```

- This keeps everything local to the file and is not visible outside the file.
- No need to qualify the namespace for the variable `secret_code`
- Good for encapsulation (can change implementation details but keep the interface the same)