1

Outline

- In this lesson, we will:
  - Describe the `namespace` keyword
  - Look at the `std` namespace
  - Discuss some of the reasons for using namespaces
  - Describe `std::cout` and its class `std::ostream`
  - Look at how `std::endl` is implemented

2

Namespaces

- Up to this point, we have the `std` namespace,
  - We have seen it with:
    - Types `std::size_t`
    - Class names `std::string`
    - Function names `std::sin`
    - Objects `std::cout`

- Now we will discuss namespaces and how they are defined

3

Namespaces

- Suppose you want to author another sine function,
  but you don't want people calling it unless they really want to

```
namespace ca_uwaterloo_dwharder {
    double sin( double x );

    double sin( double x ) {
        if ( x < 0 ) {
            return -sin( -x );
        } else if ( x <= M_PI_2 ) {
            return ((
                (1.0 - 4.0*M_1_PI)*M_1_PI*x + (3.0*M_1_PI - 1.0)
            )*4.0*M_1_PI*x + 1.0)*x;
        } else if ( x <= M_PI ) {
            return sin( M_PI_2 - x );
        } else if ( x <= 2.0*M_PI ) {
            return -sin( x - M_PI );
        } else {
            return sin( x - 2.0*M_PI );
        }
    }
}
```

4

Namespaces

- Inside your program, you would call this program as follows:

```
int main();

int main() {
    double x{};
    std::cout << "Enter a number: ";
    std::cin >> x;

    std::cout << "sin(" << x << ") = " << std::sin( x ) << std::endl;
    std::cout << "sin(" << x << ") is approximately "
              << ca_uwaterloo_dwharder::sin( x ) << std::endl;

    return 0;
}
```

5

Namespaces

- More-or-less everything in the standard library is in the `std` namespace
  - For example,
    `std::cout`
    `std::endl`
    `std::string`
    `std::size_t`
    `std::sin`
    `std::cos`
  - There are a few seldom exceptions when macros are brought over from the older C libraries:
    `assert(…)`
    `M_PI`

6

Namespaces

- Currently, the standard library does not contain a secant function
  - You could author one:

```
double sec( double x ) {
    return 1.0/std::cos( x );
}
```

- Currently, the standard library does not contain a secant function
  - Suppose there was no `std` namespace
  - Suppose that in a future version of the standard library, a `sec` function is added to the `cmath` library
  - This would either:
    - Break your code
      - The compiler indicating there are two definitions of `sec`
    - It will carefully inspect both, and call the one you didn't mean

7

Namespaces

- Warning: nothing keeps you from adding additional global variables, functions and classes the `std` namespace
  - Nothing, except of course, in the words of John von Neumann, that you would be "in a state of sin"
  - You'd probably not gain significant respect amongst your colleagues, either…

8

Namespaces

- Namespaces ensure that changes in one namespace never effect anything in other namespaces
  - For example, a company may have a global namespace

```
namespace com_cyberdyne {
    // Company-wide classes and functions...
}                                   com_cyberdyne::f(…)
```

  - A project within Cyberdyne could have a namespace within this namespace, or its own namespace

```
namespace com_cyberdyne {
    namespace skynet {
        // Skynet-specific classes and functions...
    }
    // Company-wide classes and functions...
}                                   com_cyberdyne::skynet::f(…)

namespace com_cyberdyne_skynet {
        // Skynet-specific classes and functions...
}                                   com_cyberdyne_skynet::f(…)
```


`cout` and `clog`

- The first *object* we saw in this course was:

```
namespace std {
    // Global variable declaration
    ostream cout;
    ostream clog;
}
```

- Another object in the `iostream` library is `clog`:

```
int main() {
    std::cout << "This is printed to the console"
              << std::endl;
    std::clog << "This can be redirected to a log file"
              << std::endl;

    return 0;
}
```


`std::endl`

- The next identifier from the standard library is `std::endl`
  - This is actually a function:

```
namespace std {
    // Function declaration
    ostream &endl( ostream &out );
}
```

  - It takes an `ostream` object as an argument by reference, and returns (presumably the same object) by reference

- When `std::cout` sees a right-hand operand that is `std::endl`, it actually calls `std::endl( std::cout )`

`std::endl`

- You can author a function like this:

```
std::ostream &bob( std::ostream &out );

std::ostream &bob( std::ostream &out ) {
    out << "Bob!!!";
    return out;
}
```

- You can then use this function:

```
int main() {
    std::cout << bob << std::endl;

    return 0;
}
```

Output:
Bob!!!

std::endl

- This would be weird, but it works:

```
int main() {
    std::endl( std::cout );

    return 0;
}
```

Summary

- Following this lesson, you now
  - Understand the `namespace` keyword
  - Have an idea of how the `std` namespace is implemented and the purpose of prefixing such functions with `std::`
  - Know how to define your own namespace
  - Have a better understanding of `std::cout` and `std::endl`

References

[1] No references?

Colophon

These slides were prepared using the Georgia typeface. Mathematical equations use Times New Roman, and source code is presented using `Consolas`.

The photographs of lilacs in bloom appearing on the title slide and accenting the top of each other slide were taken at the Royal Botanical Gardens on May 27, 2018 by Douglas Wilhelm Harder. Please see

https://www.rbg.ca/

for more information.

Disclaimer

These slides are provided for the ECE 150 *Fundamentals of Programming* course taught at the University of Waterloo. The material in it reflects the authors' best judgment in light of the information available to them at the time of preparation. Any reliance on these course slides by any party for any other purpose are the responsibility of such parties. The authors accept no responsibility for damages, if any, suffered by any party as a result of decisions made or actions based on these course slides for any other purpose than that for which it was intended.