C++ Classes
STL Containers

Today’s Topics

- C++ Style Guide
- Moving a Function to its own file
- Include guards
- Classes in C++
- std::string
- STL Containers
- Testing Using Catch2

Google Style Guide

https://google.github.io/styleguide/cppguide.html#Variable_Names

*Just a warning: I may violate these style rules all over the place because I was a rebel when I was young. (and the only one reading most of my code)

Let’s Extend Hello, World!

Let’s move AddOne() out of main.cpp

#include “addOne.h”

Error Messages

There are two types that you will encounter:

1. Compile Errors - won't compile

```
std::cout
```

2. Link Errors - compiles but won't link

```
extern int extern_int;
extern_int = this_int;
```

Include Guards vs “#pragma once”

An “include guard” looks like this:

```
#ifndef HELLOCS126_ADDONE_H
#define HELLOCS126_ADDONE_H
//  this won’t get included more than once
int AddOne(int my_int);
#endif //HELLOCS126_ADDONE_H
```

#pragma once

is a non-standard but widely supported preprocessor directive.

It serves the same purpose as include guards, but with a few advantages, including:

- less code,
- avoidance of name clashes, and
- sometimes improvement in compilation speed

BUT…

not necessarily available in all compilers and its

implementation is tricky and might not always be reliable

Note: “using namespace std;” is bad practice?

Why?

Because “std” has MANY named items. If you have a naming collision, you may not know it. It could cause you headaches.

namespace

Shall we play that game again?

scratch_5.cpp

Let’s extend Hello World!

Let’s create a class: HelloCOE

CLion creates BOTH .cpp and .h

What goes in each?

Let’s extend Hello World!

Let’s create a class: HelloCOE

CLion creates BOTH .cpp and .h

What goes in each?

- Header (.h)
- Source (.cpp)

Strings in C++ (string class)

```
#include <string>

std::string myStr = “this is my string!”;
```

*if you #include <iostream>, <string> is included too. This would be one of those cases where guards come into use.

http://www.cplusplus.com/reference/string/string/

Let’s extend Hello World!

Let’s create a class: HelloCOE

What is meant by: “undefined behavior”

String operator[] beyond string.length()

Compare:

    operator[] vs string.at()

Classes in C++ (HelloCOE.cpp)

- `public` - accessible from outside the class
- `private` - NOT accessible (or viewed) from outside the class
- `protected` - NOT accessible (or viewed) from outside the class but accessible from inherited classes

Members of classes are PRIVATE by default in C++

STL (Standard Template Library)

What is a template?

Simply put, templates are a way to design a form that tells the compiler how to re-use some code for you.

*We will look at these much more closely in a later lecture

Template

Form (tutorialspoint):

```
template <class type> ret-type func-name(parameter list) {
   // body of function
}
```

Example:

```
template <typename T> T MyMax(T x, T y)
   {
      return (x > y)? x: y;
   }
```

Template Example

```
template <typename T>
    T MyMax(T x, T y)
    {
       return (x > y)? x: y;
    }

Use case examples:
    intMax   = MyMax<int>(3, 7);
    floatMax = MyMax<float>(3.0, 7.0);
```

Template Example

```
template <typename T>
    T MyMax(T x, T y)
    {
        return (x > y)? x: y;
    }

int_max = MyMax<int>(3, 7);
```

Remember the compiler pre-process? Here, you are telling the compiler:

> “Replace every “T” with “int”!”

“Replace every “T” with “int”!”

```
template <typename T>
    T MyMax(T x, T y)
    {
       return (x > y)? x: y;
    }
```

Becomes:

```
    int MyMax(int x, int y)
    {
       return (x > y)? x: y;
    }
```

STL (Standard Template Library)

So...just from pure deduction, what would a “template library” be?

STL (Standard Template Library)

The C++ standard template library is...

A library full of standard templates that allow us to extend functionality to a variety of types without needing to write new code for each type. Instead, we let the compiler do it for ut.

STL (Standard Template Library)

STL has four components (flavors)

- Algorithms
- Containers
- Functions
- Iterators

STL (Standard Template Library)

For now, we are only going to concern ourselves with:

- Algorithms
- **Containers**
- Functions
- Iterators

Containers

For now, we will only look at the simplest of them:

- Array
- Vector
- List

array

This is something with which we are all familiar:

```
std::string myArray[5];

std::string myArray[5] = {“this”, “is”, “my”, “string”,
“buddy”};
```

array

How do we get rid of these pesky “std::”?

```
using std::string;

string myArray[5];

string myArray[5] = {“this”, “is”, “my”, “string”,
“buddy”};
```

vector

A vector of strings:

```
    #include <vector>

    using std::vector;

    using std::string;

    vector<string> msg;
```

What’s different from an array?

- Dynamic size - grows as needed
- Append - `msg.push_back(“another”);`

list

This is getting boring.

```
    #include <list>

    using std::list;

    using std::string;

    list<string> listOfStrings;
```

What’s the purpose? Advantage?

The claim-to-fame for list is speed of insertion and deletion in the middle of the list.

C++ vector vs Java ArrayList

There ARE differences that matter if you are running multiple threads and want to use the same data.

In this class, you won’t need to know the difference.

Testing With Catch2

1. Download: https://github.com/catchorg/Catch2
2. Tutorial: https://github.com/catchorg/Catch2/blob/devel/docs/tutorial.md#top
3. Amalgamated Header: https://raw.githubusercontent.com/catchorg/Catch2/devel/extras/catch_amalgamated.hpp (This is the same as what the preprocessor does with a succession of #includes.)

Want to watch a video instead?
https://www.youtube.com/watch?v=Ob5_XZrFQH0
(start around the 6 minute mark)