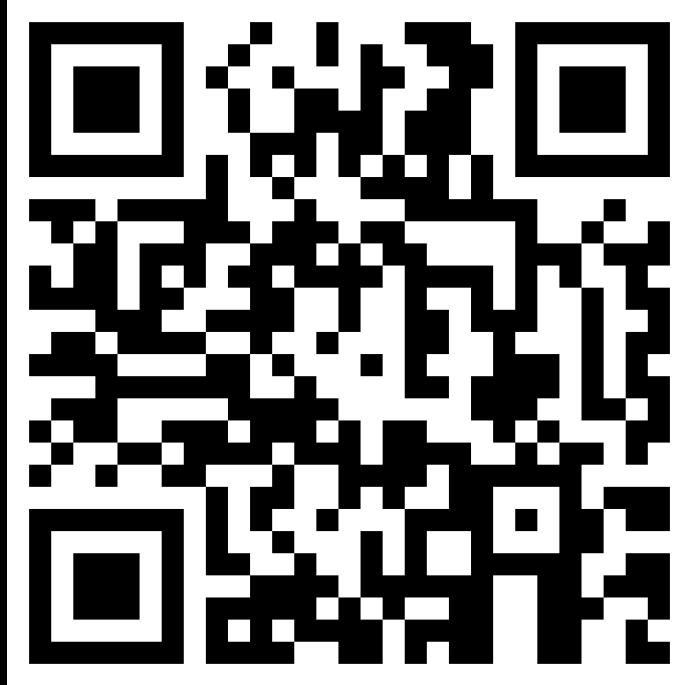


MTRNSoc C++ Workshop

By Dan

2022/10/18

Attendance



<https://forms.office.com/r/juxYn10TbY>

Overview

1. Constructors.
2. Vectors
3. Pointers & Iterators.
4. STL Algorithms.
5. Sets
6. Maps

Some theory and some questions.

Constructors

Constructors are functions which instantiate an object.

```
class Vector;  
  
int main() {  
    Vector v; // Instantiate an object called v of type Vector.  
}
```

Different Types of Constructors

```
class Vector {  
    Vector();           // Default constructor.  
    Vector(int);        // User-defined constructor.  
    Vector(Vector const&); // Copy constructor.  
    Vector(Vector&&);    // Move constructor.  
    Vector& operator=(Vector const&); // Copy assignment.  
    Vector& operator=(Vector&&);    // Copy assignment.  
};
```

Vectors

Vector is an array that can change size.

```
std::vector<int> v1;      // Size of 0.  
std::vector<int> v2(3);   // Size of 3.  
std::vector<int> v3(4);   // Size of 4.
```

If you don't resize, you will get a segmentation fault.

```
// Bad.  
std::vector<int> v4;  
v4[0]; // Seg fault.  
  
// Good.  
std::vector<int> v5;  
v5.resize(1);  
v5[0]; // No seg fault.
```

Vectors

Can also create vectors with an initialiser list.

```
std::vector<int> v6{0}; // [0].  
std::vector<int> v7{1, 2, 3}; // [1, 2, 3].
```

Pointers

Pointers are variables whose value is an address.

```
int var{42};  
int* ptr{&var};  
std::cout << ptr << std::endl;    // Prints the address.  
std::cout << *ptr << std::endl;   // Prints the value AT the address.
```

address:	0x0	0x4
	-----	-----
value:	0x4	42
	-----	-----
	^	^
	int*	int

Pointer Arithmetic

We can look up other addresses by offsetting.

```
int* ptr{new int[3]};  
*ptr = 42;           // Base address.  
*(ptr + 1) = 43;    // Offsets by sizeof(int) * 1.  
*(ptr + 2) = 44;    // Offsets by sizeof(int) * 2.  
std::cout << *ptr << std::endl;  
std::cout << *(ptr + 1) << std::endl;  
std::cout << *(ptr + 2) << std::endl;
```

Or more concisely...

```
*ptr = ptr[0];  
*(ptr + 1) = ptr[1];  
*(ptr + 2) = ptr[2];
```

Iterators

Iterators are *light* class wrappers of pointers.

Lets us iterate over an STL container.

```
std::vector<int> vec(3);  
*vec.begin() = 42;      // Base address.  
*(vec.begin() + 1) = 43; // Offsets by sizeof(int) * 1.  
*(vec.begin() + 2) = 44; // Offsets by sizeof(int) * 2.
```

Or more concisely...

```
std::vector<int> vec(3);  
vec[0] = 42; // Base address.  
vec[1] = 43; // Offsets by sizeof(int) * 1.  
vec[2] = 44; // Offsets by sizeof(int) * 2.
```

Sets

A useful STL container if we only want unique values.

```
std::set<std::string> s;  
s.insert("abc");  
s.insert("abc");  
s.erase("abc");
```

Maps

A useful STL container if we want a relationship between two types of values.

```
std::string str("abbccc");  
std::map<char, int> count;  
count['a'] = 1; // Access via 'a' to assign 1.  
count['b'];     // Inserts a key 'b'.
```

We can iterate through maps.

```
for (auto const& entry : count) {  
    std::cout << entry.first << std::endl; // Key.  
    std::cout << entry.second << std::endl; // Value.  
}
```

STL Algorithms

Very convenient functions for common procedures on STL containers.

```
std::vector<int> v(100);  
for (auto& i : v) {  
    i = 42;  
}
```

Can be condensed into...

```
std::fill(v.begin(), v.end(), 42);
```

<https://en.cppreference.com/w/cpp/algorithm/fill>

Problem Solving Session

Have a go at `inverted_map.exercise.cpp` and
`discard_smallest.exercise.cpp`.