Advanced C++ Programming: CS179

Lecture 13-1: C++ Versions

C++ 98, 11, 17

- C++ is constantly adding new features and improving old ones
- The core group of people overseeing this try to put out an update every 3 years, but some updates are more important than others
- 99% of what is added is backwards compatible with not only older C++ versions, but with C itself
- 11 has some neat stuff I'll go into today, but 17 is over my head.

Features We Know

```
for( auto iter = X.begin()...

Rock *X = nullptr;

Rock::Rock () {}
Rock::Rock(int) : Rock() {}
```

- There are some features from 11 that we've been using without realizing what version they were from
- auto - the ability to have the compiler fill in the type of a variable for you by looking at what is on the right side of the = sign
- nullptr - A null pointer that knows it is a pointer, instead of NULL, which is just 0
- Chaining constructors - making your special constructor call the default as well

More Common v11 features

```
unordered_map
        <string, int> ages;

for( auto X : ages ){}
```

- unordered_map - A regular map lets you store "key-value" pairs. An unordered map gives you faster lookups but loses the ability to loop small to large
- Smart pointers - We did a week on this
- foreach - Shorthand for looping through a container
  - Interchangeably called a range-based for loop
- Regular expressions - We did a week on this too

Major v11 Features

```
// Normal sort with <
sort(vec.begin(), vec.end())

// Sort with functor
sort(vec.begin(), vec.end,
Student::ByAge());

// Sort with lambda
sort(vec.begin(), vec.end(),
[&](int a, int b)
        {return a < b;});
```

- Lambda - The ability to write a function inside an argument list
  - I hate this one *so bad*, but you can't say you know 11 without knowing this
  - The entire point of functions is to remove duplicated code. Now we're copying an entire function in to multiple places. It's the exact opposite of proper structure
    - But then again, I'm old
      - Just looking at this makes me mad
    - Oh wait, the inventor of C++ says they suck too
- One interesting part - since the lambda is right here, the [&] lets it use local variables without passing them in.

Move

...wha?

- We know pass by value, pass by pointer, and pass by const reference
- Move means instead of using = to make yourself a copy of the other object, you use move to become that object.
  - If I don't need 2 (so why copy), move lets me steal the contents of the other one directly
    - If you have an array, I don't want a copy, I just want a pointer to your array and then remove your pointer to it
    - Basically stealing

Move pt2

```
vector<int>X;
vector<int>Y;
X = Y; // Copy.  Both have same data

X = move(Y);// X has the data, and Y has nothing

Lamp( Lamp && rhs ) {
mBulb = rhs.mBulb;
rhs.mBulb = nullptr;
}
```

- lvalue means left of the =, and rvalue means right of the =
- Normally, the lvalue is what changes
  - Can have a reference to the left, but only a const reference to the right
- Move lets you change the rvalue too
  - This is the how. The why was before - we don't want a copy, we want to steal your properties
- The syntax for a right-side-reference is &&
  - To use in the MoveConstructor method
  - Yay for reusing symbols. * and & both have 5 now

Big 5

- If your class has any dynamic memory at all, you had to write the Big 3.
  - CopyCon, CopyAssign, Destruct
- With the addition of Move, there are 5
  - CopyCon, CopyAssign, Destruct, MoveCon, MoveAssign

Deleting Methods

```
class Rock {
Rock & operator=
        (const & rhs) = delete;
Rock(const &other)
        = delete;
```

- A common trick for preventing other classes from constructing or copying your object is to make the default constructor private
- That still lets objects of your own class do it though
- If you reeeally don't want someone to construct or copy you, you can delete the methods entirely

Random Numbers

```
#include <random>
#include <functional>

normal_distribution<int>
        pick(0,99);// Bell curve
linear_congruential_engine
        engine;
auto newRand = bind
        ( pick, engine );
int number = newRand();
```

- rand() picks a number between 0 and INT_MAX and mods it by the value you say
- This means it is not evenly distributed unless you pick a number that INT_MAX is divisible by
  - If rand gave between 0 and 9, and I asked for a number between 0 and 3, the chances would be 0 1 2 3 0 1 2 3 0 1. 2 and 3 are now less likely
- Now you pick a distribution and an algorithm
  - The distribution part is awesome. Not just linear anymore. 20 choices

Reference

- Again, since we are off the end of the book, here are some online resources I found useful
  - https://www.stroustrup.com/C++11FAQ.html
  - https://cppdepend.com/blog/?p=319
  - https://smartbear.com/blog/develop/the-biggest-changes-in-c11-and-why-you-should-care/
  - https://en.cppreference.com/w/cpp/11
  - https://en.wikipedia.org/wiki/C%2B%2B11

End

Why do I keep trying to split these into 2 when this is a once a week class?