Classes

More than a Glorified Struct

Review

- Classes have member variables and functionality
- Contents are private by default
  - Traditionally member variables are private with member functions being public
  - Use accessor and mutator functions to work with private member variables

Picking up from Last Time…

- Started to define a Point
  - Declared some member variables
  - Added some member functions such as `move_right()` and `set_position()`

- New Question… how do we initialize the member variables?
  - Calling each individual mutator function is cumbersome
  - There's got to be a better way

Introducing… a Constructor

- A specially defined function
- Automatically called when the object gets created
- Sets up the object with appropriate values
- If a constructor is not provided by the programmer, one will be automatically generated but will not initialize any values

More Details on Constructors

- Must have the same name as the class
- Not allowed to return anything
- May have parameters
  - If no parameters provided, referred to as default constructor
  - If parameters are provided it can be defined in a couple ways:
    - Option one:
```
Point::Point(int a, int b) {
        x = a; y = b;
}
```
    - Option two: `Point::Point(int a, int b):x(a),y(b) {}`
- Can't be called using the dot operator
- Can be called after the object is declared

```
next_point = Point(3,3);
```

Classes in Classes

- Classes can contain other objects

```
class Line {
    private:
        Point start_point, end_point;
};
```

- Calling the constructor of an object in the constructor of the host
  - Option one: `Line::Line():start_point(1,1), end_point(2,2)`
  - Option two:

```
Line::Line(int s_x, int s_y, int e_x, int e_y):
start_point(s_x,s_y), end_point(e_x, e_y)
```

Destructor

- Special function which is called automatically when the object is destroyed
- Think of this as the “opposite” of the constructor
- Generally used to clean up dynamic memory usage, file I/O handles, database connections, etc

Passing Objects

- Can be passed the same way as any other variable
- Traditionally pass by reference or pointer
  - Generally more efficient
  - Pass by value creates a copy -> all of the memory needs to be copied
  - Pass by reference only uses the one variable, no copies
  - Can be problematic since changes to references persist

const – a special keyword

- To prevent changes to an object being passed, put const in the parameter listing

```
bool isGreater(const Point& a, const Point& b);
```

- If a function isn't supposed to change anything, put a const at the end

```
void print() const;

void Point::print() const {/* definition code goes here*/}
```

  - Will cause an error if the code in print changes anything
- If you use const for one parameter of a particular type, then you should use it for every other parameter of that type that is not changed by the function call
- Const can't be a member variable of a class

static – another keyword

- Creates a variable shared by all the objects of the same class

```
static int count;
```

- Allowed to be private
- Permits objects of the same class to communicate
- Important detail: must be initialized outside of class

```
int Point::count = 0;
```

  - The person writing the class does this
  - Static variables can’t be initialized twice

Other uses of **static**

- Can have static functions
  - They cannot use non-static things
  - Are not attached to an object

```
static int getCount();

int Point::getCount() { //note: no static keyword
   count++;
   return count;
}
```

Function call: `Point::getCount();`