CSU NUPAC Tutorials
2019

Gabriel Palacino

As usual I’m borrowing a lot of the material from Sam Meehan (University of Washington)
Classes

- Classes are like structures on steroids: “data-structure-function-thingy”
  - In fact, structures are explicitly a specific form of a class
- C++ philosophy centers around class construction and usage
  - Understanding this now will benefit you greatly → slow me down
- There is too much to cover - the selection here is what I think is most essential
Classes

- Some basic concepts from structs/functions transfer over
  - Definition/prototyping come before the main function → use your header!
- Some basic concepts DO NOT transfer over
  - Simple variable access → access specification
Classes

- Some basic concepts from structs/functions transfer over
  - Definition/prototyping come before the main function ➔ use your header!
- Some basic concepts DO NOT transfer over
  - Simple variable access ➔ access specification

```
//my::class::header
#include <iostream>

using namespace std;

class Rectangle {
public:
    int width, height,
    int type;

    void setDimensions(int a, int b);
    int getArea();
    int getType();
};
```
Classes

- Some basic concepts from structs/functions transfer over
  - Definition/prototyping come before the main function → use your header!

- Some basic concepts DO NOT transfer over
  - Simple variable access → access specification

We are going to make a class

The class is called “Rectangle”

This is where we store stuff in memory

These are the tools we can use from the class
Classes

- Some basic concepts from structs/functions transfer over
  - Definition/prototyping come before the main function ➔ use your header!

- Some basic concepts DO NOT transfer over
  - Simple variable access ➔ access specification

We are going to make a class
The class is called “Rectangle”

This is where we store stuff in memory

These are the tools we can use from the class

**WARNING**: 
“.” accessor operator now:
(1) accesses members
(2) executes functions
Classes

- Some basic concepts from structs/functions transfer over
  - Definition/prototyping come before the main function ➔ use your header!
- Some basic concepts **DO NOT** transfer over
  - Simple variable access ➔ access specification
Access Specification

- Members of a class exist in three ways
  - Public: Anybody can access it as long as you play by the rules (free and open community)
  - Private: Only the immediate instance of the class can internally access it ("my prrrreeeciouuuuuussss")
  - Protected: Only are allowed access to it if you derive from it ("in the Corleone family")
Public Access

- Determines “who” can see/use a given member of a class - reference here
- Public: Anybody has access to it (“struct members”)
  - You can directly touch that place in memory

Everything in this “block” is public

The main() function can change it like a struct
Private Access

- Determines “who” can see/use a given member of a class - reference here
- **Private**: Only the class itself has access to that spot in memory
  - **IMPORTANT**: We have functions at our disposal now
  - This is the more appropriate way to do things (in my opinion)

Everything in this “block” is private

Must use methods and functions of the class itself to access/modify it
Class Factorization

- Organization is key: interface/implementaiton
  - Class declaration → header
  - Method prototyping → header
  - Method implementation/definition → source

Specify getWidth() “belongs to” the Rectangle class namespace
Class Factorization

- Definition of helper functions allow
  - Functions = public → can be accessed by main()
  - Members = private → only accessible by class (e.g. its functions)

- Benefit: getDescription() function
  - The “user” doesn’t need to know exactly what it does, only that it gives some sort of a description
Initialization

  - This is tiring and can become cumbersome when code gets large
  - What happens if you forget to actually set a member? ➔ ill-defined state “???????”
Initialization - constructors

- **Solution:** constructors
  - A special function which combines [1] and [2] during instantiation
  - Allocates members/memory belonging to the object
Initialization - constructors

- Solution: constructors
  - A special function which combines [1] and [2] during instantiation
  - Allocates memory belonging to the object
- Somewhat “forces” you to set things up properly

The only way to trace your what your code is *actually* doing is by printing stuff to the terminal (#infinitescrollcapture)
**Initialization - constructors**

- **Solution**: *constructors*
  - A special function which combines [1] and [2] during instantiation
  - Allocates memory belonging to the object
- "forces" you to set things up properly for an instance of a class
  - Code won’t compile otherwise
Initialization - and destructors

- Solution++: destructors
  - Most necessary when using more complex memory allocations
  - Example: when allocating pointers (dynamic memory) in your class

- “Leave it like you found it”
  - If you don’t, you will create memory leaks
Initialization - and destructors

- **Solution++**: destructors
  - Most necessary when using more complex memory allocations
  - Example: when allocating pointers (dynamic memory) in your class

- “Leave it like you found it”
  - If you don’t, you will create memory leaks

**Deleting a pointer**: “delete pointer;”
Classes are Objects

- Classes are just allocated blocks of memory (like integers, like arrays, like structs ...)
  - The functions are a bit of a more complicated story - read this if you are curious
  - The functions are a bit of a more complicated story - I prefer to parameterize my ignorance

- We can point to them just like any other object/variable
  - We need to be more careful about the “leave it like you found it” rule

Allocate a “new” version of the object in memory

Access things using “->” instead of “.”

clean up after yourself
Classes are Objects

- Classes are just allocated blocks of memory (like integers, like arrays, like structs ...)
  - The functions are a bit of a more complicated story - read this if you are curious
  - The functions are a bit of a more complicated story - I prefer to parameterize my ignorance
- We can point to them just like any other object/variable
  - We need to be more careful about the “leave it like you found it” rule

myRect as an object

myRect as a pointer to an object
Homework 2 - Part 1

- Create a two classes
  - A 3D vector class (euclidean) with functions to access its members and its norm
  - A Lorentz four-vector class with functions to access its members and its norm using the $\eta(+- - -)$ metric with functions
  - Implement constructors and destructors.

Due midnight of Tuesday, February 19