Classes and Structs

- For c++ they are nearly the same
- Differences
  - Structs everything is public by default, classes are private by default
- Struct was originally from c
  - could have nothing besides variables, no methods etc
- Usually structs are only used for very simple data structures
  - POD (plain old data)
Classes (Access Modifiers)

- public
  - Accessible to everyone
- protected
  - Accessible to class and inherited classes
- private
  - Only the class (and friends) can access
Classes (Instances)

- An instance is an occurrence
  - Each has their own values for fields

- Declaring an instance
  - Class Student{}; /// our class
  - Student a; /// declare our instance
Classes (Accessing fields)

- Student a; /// declare our instance
- cout << a.name << endl;
Classes (Constructors)

- All classes have a default constructor (which is public)
- Other Constructors can be used (Constructor overloading)
- Constructors with defaults
- Copy constructors
  - Implicit copy constructor is a member-wise copy
Classes (Passing)

- Passing a class in a function creates a copy of the class
  - Uses the copy constructor
  - Changes won't be preserved outside of the function

- To preserve changes, pass by reference or as a pointer
  - `void f(Student& a){}` /// using a reference
  - `void f(Student* pa){}` /// using a pointer
Classes (Inheritance)

- class Person{};
- class Student : public Person{}; /// students are people too!
Classes (using references and pointers)

- When using pointers, often easier to read without dereferencing (*)
  - use the -> symbol
Code Separation

- .h and .cpp files
  - For the rest of this class we will be using separate .h and .cpp for classes
- .h (interface)
- .cpp (implementation)
  - use :: to tie the function in the .cpp to the .h definition
Abstraction and Encapsulation

- Abstraction:
  - making a well defined interface for the class

- Encapsulation (obfuscation):
  - Try to keep implementation details private
  - Don't make a variable/method private or protected unless necessary
Enums

- enum Weekend { SATURDAY, SUNDAY};
- Weekend myRelaxingDay = SUNDAY;
- Internal Representation
  - 0, 1, 2, ...
- enum Months {JAN = 1, FEB, ..., DEC};
  - start the enumeration at 1