Introduction to Object Oriented Programming

Learning about Classes

Classes

- We are now moving into the concept of **OOP** → “Object Oriented Programming”
- Classes are very similar to structs
  - Structs are collections of data
  - Classes are simply more powerful

Why bother using a class?

- Structs can’t “do” anything
- Classes can have functionality built in
- Example… mystring.length()
  - mystring is a string object
  - mystring has an internal member variable that tracks the length
  - length() is a member function

Basic Example

- Suppose that we create a Point class
  - It contains an X value and a Y value
  - We can create member functions to move the point, display the value, or perform other manipulations
- See code example in Canvas

Classes v Structs

- Structs
  - No functionality
  - Typically used to hold a collection of variables
- Classes
  - Can still hold variables (just like a struct)
  - You can implement custom functions inside a class
    - Remember the car analogy from several lectures ago?
    - We could do something like **`mycar.estimatevalue()`**

Vocabulary

- Struct: an object without any member functions; collection of data items of diverse types
- Class: an object with both member variables and member functions
- Object: instance of the class
- Member Variable: variable that belongs to a particular struct/class
- Member Function: function that belongs to a particular class

Introducing Encapsulation

- Hide the details of your class from others
  - Makes your class easier to maintain
  - Helps avoid broken code

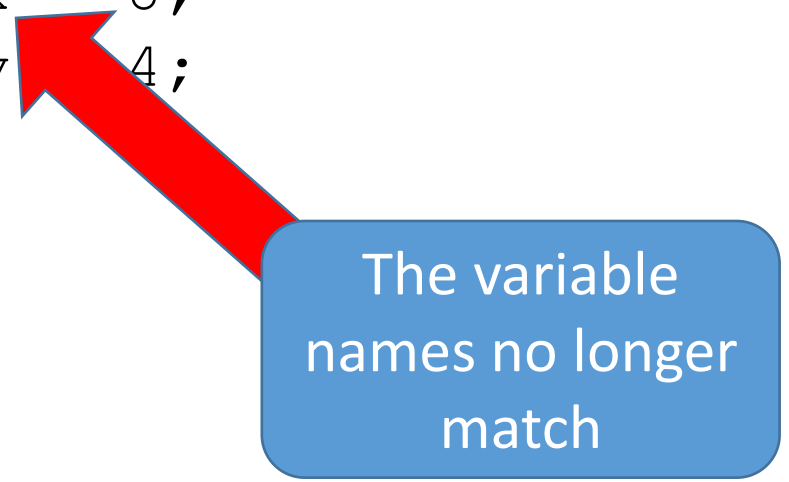
- Consider the Point class
  - What if we change **int x;** → **int x_position;**
  - That's a problem for anyone who was using our Point class

Example of Broken Code

```
class Point {
public:
    int x_position;
    int y_position;

    void move_left(int);
};
```

```
int main() {
    Point p1, p2;
    p1.x = 8;
    p1.y = 4;
}
```



How to Implement Encapsulation?

- Introduce the concept of **accessor** functions
  - Functions that retrieve values
  - E.g. implement **get_x()** and **get_y()**
  - Now there's a layer of separation between the implementation (your code) and the interface (how people interact with your code)
  - Details such as internal variable names no longer matter

- **mutator** functions are used to set values
  - Examples include **set_x()** and **set_y()**

Accessor and Mutator Functions

- Use a consistent naming scheme
- Examples
  - get_grade(), get_location(), get_name()
  - set_grade(), set_location(), set_name()

- Accessors are commonly known as “getters”
- Mutators are commonly known as “setters”

How do we enforce our plan?

- C++ includes the concept of access specifiers
- For now, we will introduce two specifiers:
  - Public: these variables and functions are available to any code that includes the header file
  - Private: can only be accessed or modified by code within the same class
- See demo using the point class

Why are accessors and mutators critical?

- In combination with access specifiers, accessors and mutators allow us to control access
- Especially useful when you want to have “read-only” member variables
  - Users can retrieve the variable using a public “getter” function
  - They cannot modify a private value unless you provide a “setter”

How secure are access specifiers?

- This is not meant to prevent people from looking at your source code
- A programmer could still open your .cpp file and look at the names of “private” variables
- The concept of public and private members is enforced by the compiler
- You will receive a compile-time error if you try to access unauthorized variables or functions

Classes vs Structs

- Structs
  - No functionality
  - Members are public by default
- Classes
  - Functionality
  - Members are private by default

Separating Your Code Into Files (again)

- Classes are typically written with their own header (.h) and implementation (.cpp) files
- Point.h
  - Contains the class definition with the member function prototypes and member variables
- Point.cpp
  - Holds the corresponding function definitions

Understanding the Concept of an Object

- By default, each object has its a personal copy of each member variable
  - This is a crucial observation!
- Consider the Point class
  - If you create Points p1 and p2, they are independent
  - E.g Modifying the X location of p1 will not change the X location of p2