CS 162
Intro to Computer Science II

Lecture 11

OOP

Accessors vs. Mutators

this keyword, const

2/7/24

Oregon State University

Odds and Ends

- Assignment 2 text file got updated last Thursday (Feb 1st)
  - Redownload the zip file to get the latest text file
  - How to error handling the age? Check here: https://classes.engr.oregonstate.edu/engr/winter2023/engr103-010/demo/week8/error.cpp

- Midterm Exam date update:
  - Previous: Monday of week 6 (2/12)
  - Now: Friday of week 6 (2/16)

Today's Topics:

- Intro to OOP
- Accessor vs. Mutator functions
- `this` keyword (pointer)
- Separate class files

C++

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 can have collections of data and perform operations

→ classes are simply more powerful

Why use 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 in string class

Class in real world…

- Write a program to model a university
- Use specific classes to represent the real-world objects that are part of a university
  - Students
  - Instructors
  - Courses
- Each of these classes would have its own member variables (attributes) and member functions, and they would interact by sending messages to each other via member functions

Class in real world…

- Our Student class might look like this:
  - Member variables (attributes):
    - Name
    - ID
    - GPA
    - Current courses
    - Completed courses
    - Etc.
  - Member functions:
    - Register for course
    - Submit assignment
    - Receive grade

Class in real world…

- Our Instructor class might look like this:
  - Member variables (attributes):
    - Name
    - ID
    - Office location
    - Current courses
    - Past courses
    - Salary
    - etc.
  - Member functions:
    - Assign grade
    - Assign homework
    - Assign exam
    - etc.

Class in real world…

- Our Course class might look like this:
  - Member variables (attributes):
    - Title
    - CRN Number
    - Instructor
    - Enrolled students
    - Assignments
    - Student grades
    - etc.
  - Member functions:
    - Enroll student
    - Drop student
    - Add assignment
    - Add grade
    - etc.

Why classes are good?

- Straightforward
  - Have a good understanding of these things and how they interact
- Reusable
  - All the relevant member variables and member functions in a single package (self-encapsulated)

Today's Topics:

- Intro to OOP
- Accessor vs. Mutator functions
- `this` keyword (pointer)
- Separate class files

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

- Demo…

C++ Access Specifiers

- C++ includes the concept of access specifiers (modifiers)
- For now, we will introduce two:
  - **`public`**: these variables and functions are accessible and modifiable to any part of the program
  - **`private`**: can only be accessed or modified by code within the same class

- Why would we want to make something private?

Introducing Encapsulation

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

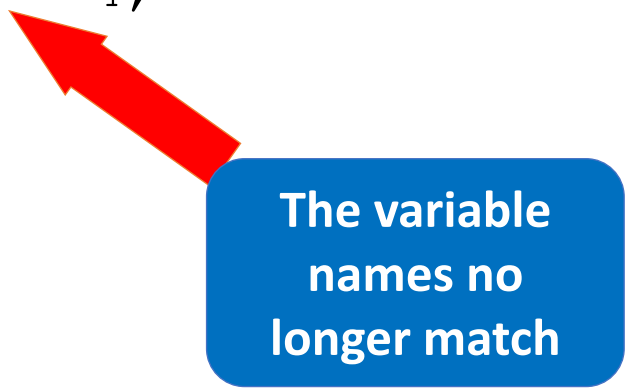
- Consider the Point class
  - What if we change `int x;` → `int x_position;`
  - That's a problem for anyone who was using our Point class

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 avoid this problem?

- Make x and y private!
  - So they cannot be modified outside of our class

```
class Point {
private:
        int x;
        int y;
public:
        void move_left(int);
};
```

- Demo…
- Wait a second… Once x and y are private, how to we access or modify the state of an object now?

How to Implement Encapsulation?

- Introduce the concept of **accessor** functions
  - Functions that are used to **get** (or **access**) values of an object from outside (or inside) the class
  - E.g. implement `get_x()` and `get_y()`
  - Now there's a layer of abstraction 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 (or mutate) values of an object from outside (or inside) the class
  - E.g. `set_x()` and `set_y()`

Accessor and Mutator Functions

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

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

- Demo…

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
  - Convention: No functionality
  - Default **public**
- Classes
  - Functionality
  - Default **private**
  - **Convention:**
    - **member variables: private**
    - **member functions: public**

Vocab

- **Struct**: a type definition without any member functions; collection of data items of diverse types
- **Class**: a type definition 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
- **Encapsulation**: the details of implementation of a class are hidden from the programmer who uses the class

Review

- Abstraction vs. Encapsulation
  - Abstraction: hide unwanted details while giving out most essential details
    - i.e. 10,000 feet view
  - Encapsulation: hide the code and data into a single unit
  - In short, abstraction hides details at the **design** level, while encapsulation hides details at the **implementation** level
- Classes have member variables and functionality
- Contents are private by default
  - Traditionally member variables are private with member functions being public
  - Use accessors and mutators to work with private member variables
    - `get_grade(), get_location(), get_name()`
    - `set_grade(), set_location(), set_name()`
- Classes are typically written with their own header (.h) and implementation (.cpp) files

`this` Keyword

- Can be used inside any class functions as a **pointer** to the object with which the function was called
  - **"`this`" always points to the object being operated on**
- Using `this` can be helpful
  - Make sure we're referring to the data members of a class, not to other variables that might be in scope.
  - E.g. when a function parameter has the same name as one of its data members

```
void Point::set_x(int x){
        this->x = x;
}
```

- Demo…

Const

- To prevent changes to an object being passed, put `const` the parameter listing
  - E.g. `bool is_greater (const Point& a, const Point& b);`
- If a function isn't supposed to change anything, put a const at the end
  - e.g. `void print() const;`
  - `void Point::print() const {/* definition */}`
  - Will cause an error if the code in print changes anything
- If using const member variable, it has to be initialized in constructor(s) using initialization list
  - E.g. `Point::Point():z(5){} //where z is defined as a const int`
- Demo…