CS162, Week 2 Wednesday

Classes

Oct 3 2018
Vocab

• **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
• **Encapsulation:** the details of implementation of a class are hidden from the programmer who uses the class
Classes v Structs

• Structs
  • No functionality
  • Default public

• Classes
  • Functionality
  • Default private
Getting around Privacy

• Classes have member variables and functionality
• Default private
  • Traditionally member variables are private with member functions being public
  • Use accessor and mutator functions to work with private member variables
• Has a constructor
  • Sets up the object with appropriate values
  • Is automatically called when the object gets created
  • 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
- Is 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:
    - `Point::Point(int a, int b) {
      x = a; b = y;
    }`
    - `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 created
  - `next_point = Point(3,3);`
Classes in Classes

• Classes can have other objects

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

• Calling the constructor of an object in the constructor of the host
  • Line::Line():start_point(1,1), end_point(2,2)
  • 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)
Passing Objects

• Can be passed the same way as any other variable
• Traditionally pass by reference
  • Generally more efficient
  • Pass by value makes two copies -> requires the copy constructor at least once
  • Pass by reference only uses the one variable, no copies
  • Changes to references persist, can cause problems
Const

• 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

- 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
- Must be initialized outside of class
  - int Point::count = 0;
  - The person writing the class does this
  - Static variables can’t be initialized twice
Static

- Can have static functions
  - They can not use non static things
  - Does not use calling object

```cpp
static int getCount();

int Point::getCount() {
    count++;
    return count;
}
```

Function call: Point::getCount();