Lecture 15: Member Functions

PIC 10A

Todd Wittman

The Product Class

- Last class we introduced the Product class. It's declaration looks like:

```
class Product {
    public:
        Product ( );
        void read ( );
        bool is_better_than (Product b) const;
        void print ( ) const;
    private:
        string name;
        double price;
        int score;
};
```

Review
- public vs. private
- member functions
- constructor
- accessors
- mutators
- const
- encapsulation
- That weird };

- We defined the Read member function. Print is simple.
- Now let's look at is_better_than.

The Product Class

- To find the "bang for the buck" ratio, compute score/price.
- Return true if the current product has better ratio than a passed product b. Be sure to avoid division by zero!

```
bool Product::is_better_than(Product b) const  {
        if (b.price==0) return false;
        if (price==0) return true;
        return score/price > b.score/b.price;
}
```

- To call this function for Products a & b, use something like:

```
if ( a.is_better_than(b) )   cout<<"Buy a!";
```

- Note how we access the different variables. To get the price of a, just say "price". To get price of b, use "b.price".
- Recall that non-member functions can't get to price directly.

Sec 5.5: Default Constructors

- There's one member function left to define: the constructor Product.
- Recall that in the class declaration, Product() has no return values. This is only for constructors!
- Really all the constructor does is create the blank object. So we'll fill in default values for the private variables.

```
Product::Product ( ) {
        price = 1;
        score = 0;
        name = "No item";
}
```

Calling the Constructor

- Now to create a Product object, we call the constructor.

```
Product my_product;
```

- This sets up a Product with the default values. To set the values, use the Read function

```
my_product.read();
```

- Sometimes we want to set those values as soon as we create the object.
- We could do this by adding some parameters to our constructor.

Sec 5.6: Constructor with Parameters

- We could change the constructor declaration to take some parameters for name, price, & score. *But don't actually call the parameters that! Those names are already taken!*

```
Product (string new_name, double new_price, int new_score);
```

- The definition of the function would just assign these values.

```
Product::Product(string new_name, double new_price, int new_score)  {
        name = new_name;
        price = new_price;
        score = new_score;
}
```

- Example calls to this constructor:

```
Product lotr ("LOTR DVD", 39.99, 10);
Product sw ("Star Wars DVD", 29.99, 3);
```

Multiple Constructors

- Suppose we declared both constructors in our class declaration.

```
Product ( );
Product (string new_name, double new_price, int new_score);
```

- Won't give an error.
- When you create a product, it will call the function whose parameters match. *How cool is that?*

```
Product prod1;    //Creates blank product.
Product prod2 ("Product 2",12.99,6);
```

- You can do this parameter-matching trick with other functions as well. So we could have multiple versions of the same function, each handling different parameter inputs.

Calling our Member Functions

- Repeatedly read in products and report the best one.

```
int main() {
    string response = "y";
    Product next, best;
    while (response == "y") {
        next.read( );
        if ( next.is_better_than (best) )
            best = next;
        cout << "More data? (y/n) ";
        cin >> response;
    }
    cout << "The best value is: ";
    best.print( );
    return 0;
}
```

Putting It All Together

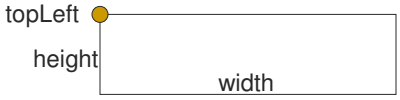
- The general outline of our program looks like this:
  - #include <libraries>
  - using namespace std;
  - class declarations { };
  - class member functions
  - function declarations
  - program functions
  - main routine

Next lecture we'll talk about putting the class information in a separate header file.

But for now we'll put everything into one cpp file like this.

Let's Make A Rectangle Class

- Our graphics library ccc_win.h has classes for Point, Circle, Line, & Message.
- Let's build a Rectangle class.
- A Rectangle can be specified by the top left corner point, the height, and the width.



- We should have member functions which construct, draw, and move the rectangle.
- *Help me write this class right now...*