Last updated 20 November, 2014

Assignment 9 (100 points):

Please see Submission Checklist (below) for submission requirements.

A class is just a collection of data (like a struct) and functions that can act on that data.

A class defines a data type, we create and use an "instance" of a class (also called an object) just like the way we have used the string class already.

(10 points) Programming Styles and Convention.
http://classes.engr.oregonstate.edu/~jessjo/CS161/OSU_IntroCodeStandards_v1.1.pdf

(30) Remember to submit your report (just over the project, not the exercises)!
Which includes:

- Understanding: a description of what you understand is being asked of you for the assignment.
- Design: an outline in **pseudocode or a drawing** as to how your code will be implemented. The design section is to be done before you start coding.
- Testing: a description of the tests you will implement to ensure your code works along with the actual testing. The testing section is to be done before you start coding. **(Make a table with three columns named Input, Expected Output and Actual Output. Discuss any discrepancies.)**
- Reflection: a discussion of the problems you encountered and how you solved them.

All data members (member variables) in every class in this assignment must be private, so you will need to write get and/or set methods as needed.
Remember to deallocate all dynamically allocated memory!

(20) Exercise components:

1. (10) Write a class called *Point* that contains two doubles that represent its x- and y-coordinates. It should have a constructor that takes two double parameters and uses them to initialize its coordinates. It should also contain a method (member function) called *distanceTo* that takes as a parameter a constant reference to another Point and returns the distance from the Point that was passed as a parameter to the Point that we called the method of. You will need to use sqrt(). For example the following should print out 5:

```
Point p1(0, 0);
Point p2(3, 4);
std::cout << p1.distanceTo(p2) << std::endl;
```

   Next, write a class called *LineSegment* that contains two Points that represent its two endpoints. It should have a constructor that takes two Point parameters and uses them to initialize its endpoints. It should contain a method called *length* that returns the length of the LineSegment – by using the distanceTo method on its endpoints – and a method called *slope* that returns the slope of the LineSegment (if the LineSegment is vertical, you can go ahead and return the value you get when dividing doubles by zero, which is infinity).

   In your main(), ask the user for the coordinates for a first point and then the coordinates for a second point. You should create two corresponding Point objects and use those to create a LineSegment object. You should then print out that LineSegment's length and slope (be sure to label your outputs). Ask the user if they would like to repeat. If the LineSegment is vertical, don't print out infinity for its slope – state that the line segment is vertical (use isinf() to check for infinity).

   Input validation: none

   Discuss: What intrinsic differences are there between structs and classes? Are there differences in how programmer's tend to use them?

   File must be called: geom.cpp

2. (10) Same as exercise 1 from last week, but with classes. Item should be a class, with a constructor that takes parameters with which to initialize its data members (name, price, quantity). You will write a *ShoppingCart* class which contains a vector of Item as a data member. It should have a constructor that

creates a ShoppingCart object with an initially empty vector. It should have a function called *addItem* that takes an Item parameter and adds it to the vector. It should have a function called *listItems* that lists the name, price and quantity for all Items in the cart. It should have a function called *totalPrice* that returns the total cost of all Items in the ShoppingCart. The input/output for the user should be the same as last week.

Input validation: Same as last week.

Discuss: Compare and contrast last week's version of this exercise vs. this week's version.

File must be called: shopCart2.cpp

(40 points) Project Component:
Same as last week's project, but with classes. Date should be a class, with a constructor that takes parameters with which to initialize its data members (day, month, year). The constructor should also verify that the parameters specify a valid date (as discussed last week).

Car should be a class. It should have two Constructors, one that takes parameters with which to initialize all its data members (make, model, year, datePurchased, purchasePrice, isSold, dateSold, salePrice) and one that takes parameters with which to initialize all of those except the last two. It should have a function called getProfit that returns the profit on that car if it has been sold, or the special value NAN (not a number, declared in math.h) if it hasn't been sold yet.

You will write a CarLot class that contains a vector of Car as a data member. It should have a constructor that creates a CarLot with an initially empty vector. It should have a function called *addCar* that takes a Car parameter and adds it to the vector. It should have a function called *listCurrentInv* that lists the data for all Cars that have not yet been sold, and then prints out the count of how many cars that is. It should have a function called *getMonthProfit* that takes as parameters a month and year and **returns** the total profit for sales in that month.

The input/output for the user should be the same as last week.

Input validation: Same as last week.

Discuss: Compare and contrast last week's version of this exercise vs. this week's version.

File must be called: carLot2.cpp

Remember to submit your **report** and **source files** to TEACH before the end of Sunday.

Submission Checklist:

makefile

Exercises (20 pts):
geom.cpp
shopCart2.cpp

Project (40 pts):
carLot2.cpp

Report, ***in PDF format***, must address these four sections (30 pts):

Understanding
Design
Testing
Reflection

The implementation part of the assignment is the .cpp file you submit.