CSCI 2270: Data Structures Summer 2012

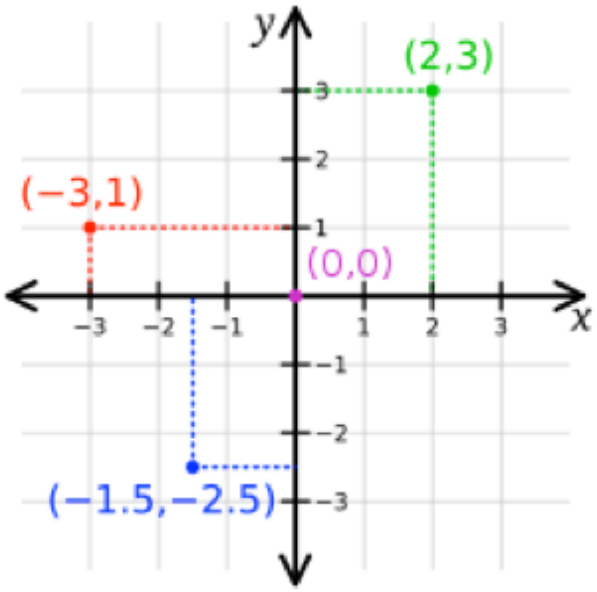
Psuedocode Due Wednesday 7/11/2012 (Midnight)
Programming Assignment Due Monday 07/16/2012 (8 AM (MST))
Identical to an assignment written by Professor Mishra
Marginally updated by Ashok Basawapatna sometime in July

I put this up early so don't worry if not everything makes sense, By Wednesday after class you should have enough information to do the whole assignment. Since you have so much less time for this assignment I put hints in italicized parenthesis. Also we will work on this all day Thursday in CSEL so you can ask questions there.
--Ashok

Make sure you look at least look at page 64 and 65 of the text

Goal:
In this assignment, you will design and implement two classes called ***point*** and ***line***. There are four goals of this assignment: (1) Write a class that meets a *precise* specification; (2) Understand how to write a class that is separated into a header file and an implementation file; (3) Implement a class using another class; and (4) Gain experience in using a test program to track down bugs in a class implementation (there are two test programs provided with this homework, test1 is for point and test2 is for line).

Point class:
The *point* class implements a Cartesian coordinate system that specifies each point uniquely in a plane by a pair of numerical coordinates, which are the signed distances from the point to two fixed perpendicular directed lines, measured in the same unit of length. For example, the following figure shows four points, (2,3), (-3,1), (0,0), (-1.5,-2.5) in a Cartesian coordinate system:



The *point* class provides member functions for the following operations: *(We will never actually draw these points and lines)*

Constructor: Two constructor functions: one that initializes a *point* object to (*0*, *0*) and another that initializes a *point* object to (*x*, *y*) where *x* and y are provided by the programmer using the class. *(recall the idea of "default arguments" in constructors)*

Copy Constructor: Initializes a *point* object to be a copy of an existing *point* object.

Get *x* and *y* coordinates of a point: *double get_x( ); double get_y( );*

Set *x* and *y* coordinates of a point: *void set_x (double x); void set_y (double y);*

Translation: Translating a point is equivalent to adding a fixed pair of numbers (*X*,*Y*) to the Cartesian coordinates of the point. That is, if the original coordinates of a point are (*x*,*y*), after the translation they will be (*x* + *X*, *y* + *Y*).
void translate(double X, double Y);

Scaling: Scaling a point is equivalent to multiplying the Cartesian coordinates of the point by a positive number *m*. If (*x*,*y*) are the coordinates of a point, the corresponding point after scaling has coordinates (*mx, my*).
void scale(double m);

Reflection: If (*x*, *y*) are the Cartesian coordinates of a point, then (*x*, *-y*) are the coordinates of its reflection across the X axis and (*-x*, *y*) are the coordinates of its reflection across the Y axis.
void reflect_x( ); void reflect_y( );

Rotation: To rotate a point counterclockwise around the origin by some angle *r* radians is equivalent to ($x \cos r - y \sin r$, $x \sin r + y \cos r$).
void rotate(double r); (remember to include math.h)

Operators overloaded: assignment operator (=) and comparison operator (==). In addition, the *point* class provides two friend functions to overload the input and output operators. Input format: two real numbers Output format: (x, y). *(i.e. for == if both points are equal return true, otherwise return false. Also remember for assignment that the object to the left of the equals is the calling object).*

Line class:
The *line* class is implemented using the *point* class. A *line* object is defined by (any) two distinct point objects through which the line passes. It provides the following member functions: *(remember to include point.h, -- our line class will just have two private variables which both points and all our member functions will act on those two private variables)*

Three constructor functions: (1) initializes the line to be same as X axis; (2) initializes the line to be passing through (0, 0) and a programmer provided point object; and (3) initializes the line to be passing through two programmer-provided point objects.

Copy constructor: Initializes a *line* object to be a copy of an existing *line* object.

Slope of the line: Slope of a *line* object passing through two different points, ($x1$, $y1$) and ($x2$, $y2$) is $(y2 - y1)/(x2 - x1)$ provided $x1 \neq x2$. Slope is ∞ if $x1 = x2$. *double slope( );*

Translation: Translating a line is equivalent to adding a fixed pair of numbers (X,Y) to the Cartesian coordinates of the points through which it passes. If the original line passes through points *p1* and *p2*, the translated line will pass through points obtained by translating *p1* and *p2* by (X,Y).
void translate(double X, double Y); (use the translate function you wrote for your point class to your advantage….)

Reflection: Reflection of a line across the X axis is the line obtained by reflecting the points it passes through across X axis. Similarly, reflection across the Y axis is the line obtained by reflecting the points it passes through across Y axis.
void reflect_x( ); void reflect_y( );

Operators overloaded: assignment operator (=) and comparison operator (==). In addition, the *line* class provides two friend functions to overload the input and output operators.

Input format: *x1 y1 x2 y2* Output format: The output operator prints out the linear equation of the line in one of the following three forms (m, a, b and c are real numbers): $y = mx + c$, $y = b$, $x = a$ *(ie: if our x's are equal we have a vertical line (x=a) and if our y's are equal we have a horizontal line (y=b)).*

Files you need to write/submit:

1. `point.h` and `line.h`: Header files for *point* and *line* classes.
2. `point.cxx` and `line.cxx`: Implementation files for *point* and *line* classes.
3. `README`: This is a text file that contains your name and current status of your program (whether the program compiles, known bugs, etc.).

Please submit a single zip file containing all these files. Please do not include any object file. All submissions must be done via moodle.

Important Suggestions:

- As always, start working on this assignment immediately. Don't wait until the last minute.
- Do not start coding until you have a good understanding of the assignment.

First design, implement and test *point* class. Once you have your *point* class working correctly, start working on your *line* class. Set aside significant amount of time to test your program. Test your program thoroughly. Two sample test programs are provided on moodle to test your classes. Start with these programs. Keep in mind that these test programs do not test your classes completely. You will have to extend these test programs for further testing.

No late submissions will be accepted unless there is a valid excuse. Reasons such as "mis-read the submission date/time", "complete forgot about the due date", or "moodle nor responding" while trying to submit at the last minute are not valid excuses. Remember that you may submit your assignment several times before the deadline. Moodle automatically overwrites your previous submission with the latest one.