CS 115 Exam 3, Spring 2011

Your name: _____

Rules

- You may use one handwritten 8.5 x 11" cheat sheet (front and back). This is the only resource you may consult during this exam.
- Explain/show work if you want to receive partial credit for wrong answers.

Grade (instructor use only)

	Your Score	**Max Score**
Problem 1		10
Problem 2		15
Problem 3		15
Problem 4		15
Problem 5		25
Problem 6		20
Total		100

Problem 1: 10 points.

Assume that `Pixel` has been defined as follows:

```
struct Pixel {
     float red, green, blue;
};
```

Write the definition of a function called `PaintItBlack` that meets the following description:

- Parameters: a single `Pixel` object, passed by reference
- Turns the pixel black (`red`, `green`, and `blue` values set to 0)
- Returns nothing

Problem 2: 15 points.

Using the bubble sort below, which works identically to the code from your labs, show the contents of the 8-element array on the next page after each iteration of `BubbleSort's` *outer* loop.

If the array does not change from one iteration to the next, you can write "SAME" for the next iteration.

```
const int SIZE = 8;

void BubbleSort(string arr[]) {
   // show the contents of the array after this loop
   for (int i=0; i < SIZE-1; i++) {
      for (int j=0; j < SIZE-1; j++) {
         if (arr[j] > arr[j+1]) {
            Swap(arr[j], arr[j+1]);
         }
      }
      cout << "After the i=" << i << " loop:\n";
      PrintArray(arr);
   }
}

void Swap(string& a, string&b) {
   string tmp = a;
   a = b;
   b = tmp;
}

void PrintArray(string arr[]) {
   for (int i=0; i < SIZE; i++) {
      cout << arr[i] << endl;
   }
}
```

INITIAL VALUES		AFTER i=0 ITERATION		AFTER i=1 ITERATION
Mercury				
Venus				
Earth				
Mars				
Jupiter				
Saturn				
Uranus				
Neptune				

AFTER i=2	AFTER i=3	AFTER i=4	AFTER i=5	AFTER i=6

Problem 3: 15 points.

After the array has been sorted:
ARRAY CONTENTS

Earth	Jupiter	Mars	Mercury	Neptune	Saturn	Uranus	Venus

Answer the following questions about binary search on this array. The code for binary search is on the next page.

(a) Fill out the following table tracing a binary search for *Pluto* in this array. You should fill out one row per iteration of the loop. If there are more rows than iterations, leave the extra rows blank.

In the *Compare To:* column, you should give the VALUE (the name of the planet) that will be compared to Pluto.

Old value of `first`	**Old value of `last`**	**Compare to:**
0	7	

(b) Fill out the following table tracing a binary search for *Earth* in this array.

Old value of `first`	**Old value of `last`**	**Compare to:**
0	7	

Code for Problem 3.

```
int BinSearch(string arr[], string searchStr) {
   int first=0;
   int last = SIZE-1;
   int middle;

   while (first <= last) {
      middle = (first+last)/2;
      cout << "First = " << first << "; last = " << last
           << "; comparing to " << arr[middle] << endl;
      if (arr[middle] == searchStr) {
         return middle;
      }
      if (arr[middle] < searchStr) {
         first = middle+1;
      }
      else {
         last = middle - 1;
      }
   }
   return -1;
}
```

Problem 4: 15 points.

For this problem, you must write a **class definition** for a class named **`Point`** that contains the elements listed below.

Note that data members should be private and member functions should be public.

You do not need to write the definitions of the member functions – just the prototypes. You will define the functions in the next problem.

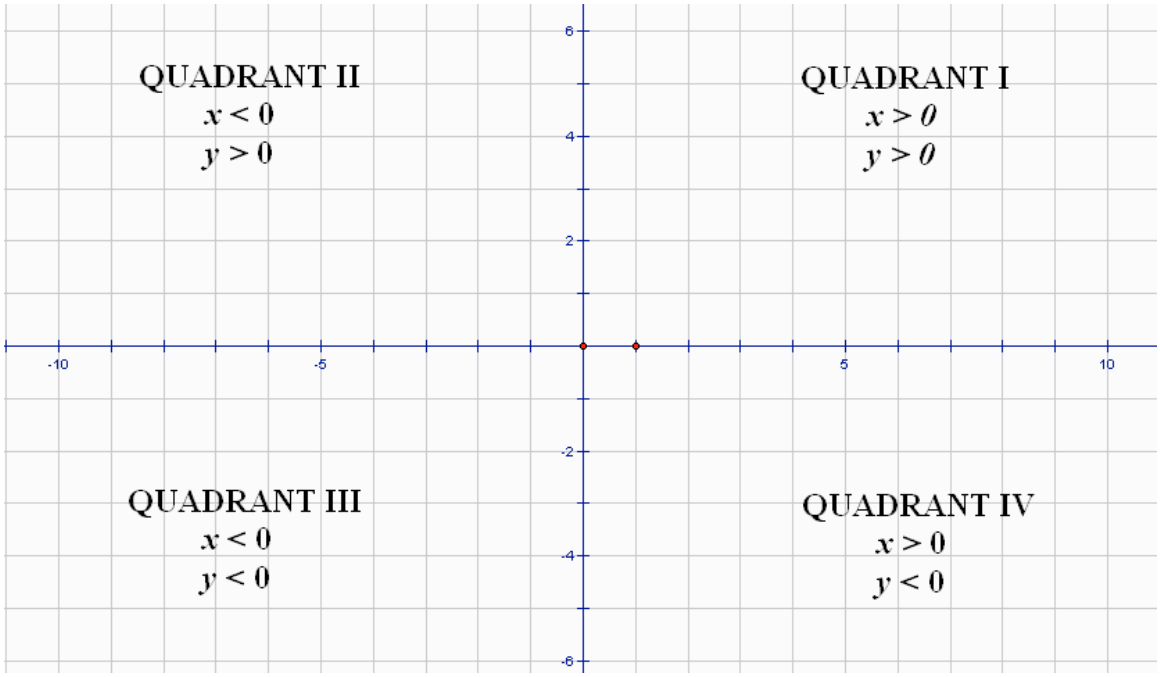
- The *x* and *y* coordinates of the point

- Prototype for a default constructor
- Prototype for a copy constructor
- Prototype for a function called `Set`. This function will take two numeric values as inputs. It will not return anything.
- Prototype for a function called `GetX`. This function will return the x-coordinate of the point.
- Prototype for a function called `GetY`. This function will return the y-coordinate of the point.
- Prototype for a function called `GetQuadrant`. This function will return the quadrant (1, 2, 3, or 4) that the point is in on the xy-plane.
- Prototype for a function called `GetDistance`. This function will take another `Point` as a parameter and return the distance between the two points.

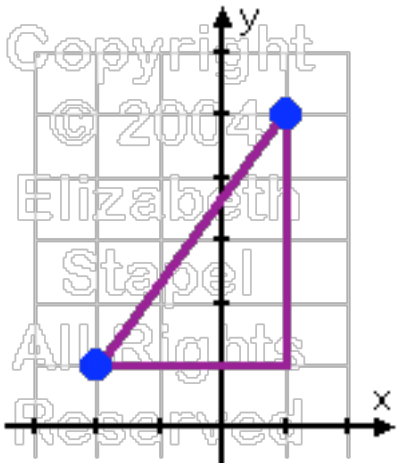
Problem 5: 25 points.

In this problem, you will write definitions for the functions in the class `Point`. Here is a little bit more information about the functions. You may assume that `cmath` has been included. *Note that none of your code for this problem should include* `cin` *or* `cout` *statements!*

- The default constructor will initialize the x and y coordinates to 0.
  The copy constructor will copy the values of the other point.
- The `Set` function will set x equal to its first input and y equal to its second input.
- The `GetX` function will return the point's x-coordinate, and the `GetY` function will return the point's y-coordinate.
- The Quadrant function will return 1, 2, 3, or 4, depending on the quadrant the point is in:



- The `Distance` function should return the distance between the given point and the point that was passed as a parameter. To get the distance between (x1, y1) and (x, y), you can imagine a right triangle whose hypotenuse is the distance.



In other words:
$(x-x1)^2 + (y-y1)^2 = distance^2$

Problem 6: 20 points.

Assume that the class definitions you wrote in Problems 4 and 5 are located in a file called `Point.h` in the same directory as the program you're about to write.

Write **a complete program** that uses the `Point` class from `Point.h` to do the following:

- Create an array of 50 `Point` objects. Ask the user to supply x and y values for all 50 points (by typing 100 consecutive numbers), and use these values to set up your objects.
- Print an error message and exit the program if the user entered an invalid number.
- Using your member functions, print the number of points in each quadrant (1 through 4).
- Using your member functions, print the X and Y coordinates of the point that is farthest from the origin (0,0).