2 Part : Container of Points and Lines 60%

2.1 Description

In this part of the assignment you will develop a class for a container for storing and managing geometric points and lines. To this end, you will also develop a class for representing two dimensional points with coordinates of type `double`, and another class for representing lines defined by two points, that is, lines given in two-point form[1].

The class `Point` is a simple class for representing two-dimensional points as a pair of their coordinates of type `double`. Two `Point` objects are equal if and only if both of their coordinates are equal. Note that the order of coordinates does matter, that is, the points $(\mathbf{a}, \mathbf{b})$ and $(\mathbf{b}, \mathbf{a})$ are not equal unless $\mathbf{a}$ and $\mathbf{b}$ are equal. For the class `Point` you will implement methods for getting information about a `Point` object, but once a `Point` object is created, its coordinates will not be changed.

The class `Line` will be used for representing lines defined by two points. More precisely, a `Line` object will contain two pointers to objects of class `Point`. If the two `Point` objects are equal, then the `Line` object represents a degenerate line, which is a single point. If the points are different and have coordinates $(\mathbf{x_1}, \mathbf{y_1})$ and $(\mathbf{x_2}, \mathbf{y_2})$, then the `Line` object represents the line consisting of all points (x, y) that satisfy the following equation:

$$(y - \mathbf{y_1})(\mathbf{x_2} - \mathbf{x_1}) = (x - \mathbf{x_1})(\mathbf{y_2} - \mathbf{y_1}).$$

This is illustrated in Figure 1. A point $(\mathbf{a}, \mathbf{b})$ lies on the line defined by $(\mathbf{x_1}, \mathbf{y_1})$ and $(\mathbf{x_2}, \mathbf{y_2})$ if and only if the coordinates $\mathbf{a}$ and $\mathbf{b}$ satisfy the above equation when we replace x by $\mathbf{a}$ and y by $\mathbf{b}$. The point **(5,2.5)** is only one of the many points on the line defined by **(1,0.5)** and **(2.4,1.2)**. We say that the point **(5,2.5)** is incident with that line.

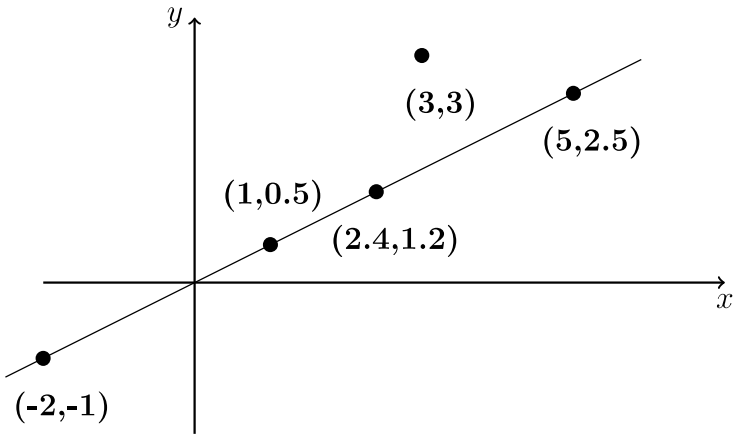

Figure 1: A line defined by the two points **(1,0.5)** and **(2.4,1.2)**.

Based on this, we consider two `Line` objects equal if they represent the same line. Thus, two pairs of `Line` objects are equal if exactly one of the two conditions holds:

[1] http://mathworld.wolfram.com/Two-PointForm.html

(1) Both lines are single points (degenerate lines) that are equal.

(2) The lines are not degenerate, and all four points lie on the same line.

For example, the pair of points **(-2,-1)** and **(5,2.5)** defines the same line as the pair of points **(1,0.5)** and **(2.4,1.2)**, as shown in Figure 1. The pair of points **(3,3)** and **(5,2.5)**, on the other hand defines a different line (not shown in the figure).

Note that, in particular, this means that the order in which the two points defining a line are given does *not* affect the line defined by the two points.

An important technical issue of the class `Line` is that it does not own the `Point` objects to which it stores pointers. This means that the only way to provide the points defining a line is by someone else allocating these points and giving them to the `Line` ready made. As the constructors and other methods of `Line` do not allocate the memory for the `Point` objects, the methods of the class are not in charge of deallocating this memory. It would be an error to deallocate this memory even in the destructor of `Line`.

Finally, the container class will store two arrays, one of pointers to objects of class `Point`, and one of pointers to objects of class `Line`. It manages (allocates and frees) the memory for all the `Point` and `Line` objects and the arrays storing the pointers. When adding points and lines, reserving additional space for the corresponding array may be necessary. For this, the private member functions `resizePointsArray_` and `resizeLinesArray_` should be implemented. When an object of class PLContainer is destroyed, all the memory allocated for it should be deallocated. The points and the lines in the container are interlinked, in the sense that it can only contain lines that are defined by points that are already present in the container.

Your job is to complete the implementations by adding private variables and complete all the function members of the three classes. A detailed description of each function is given in the corresponding header file (`Point.h`, `Line.h` or `PLContainer.h`). Below, we illustrate the desired behaviour of the functions of `PLContainer` on a few examples.

2.2 Example Usage

Please see the file `PLContainer.h` for the declaration of the methods of class `PLContainer`. Consider a container object `plc` containing the following points and lines.

Points	**Lines**
(1,2)	((1,2),(3,4))
(3,4)	((1,2),(4,2))
(5,6)	((5,6),(4,2))
(4,2)	((4,2),(3,4))

Below we illustrate the desired behaviour of some of the methods on several examples. In the following, `pp` is of type pointer to `Point`, and `lp` is of type pointer to `Line`.

> `plc.addPoint(3,4,pp)` should return false since the point (3,4) is already present. Before that, it updates `pp` to a pointer to the existing `Point` object.

`plc.addPoint(0.3,1.7,pp)` should return true, and the point (0.3, 1.7) should be added to the container, and `pp` should be a pointer to the new `Point` object.

`plc.addLine(1,2,5,6,lp)` should return false since the line defined by (1,2) and (5,6) is the same as the line defined by (1,2) and (3,4) which is already present. Sets `lp` to pointer to the `Line` object representing the line defined by (1,2) and (3,4).

`plc.addLine(1,2,5.5,6,lp)` returns false since the point (5.5, 6) is not in `plc`. In this case, the reference pointer `lp` is not changed.

`plc.getLine(1,2,5,6,lp)` returns true since the line defined by (1,2) and (5,6) is the same as the line defined by (1,2) and (3,4) which is in `plc`. Furthermore, sets `lp` to pointer to the `Line` object representing the line defined by (1,2) and (3,4).

`plc.incidentLines(5,6,number,capacity)` returns an array with pointers to the lines ((1,2),(3,4)) and ((5,6),(4,2)), since these are all the lines in the container `plc` on which the point (5,6) is lies. Furthermore `number` is updated to 2, and `capacity` is set to the capacity of the returned array.

`plc.deletePoint(4,2)` should return true, and the point (4,2), together with the lines ((1,2),(4,2)) and ((5,6),(4,2)) and ((4,2),(3,4)), which are the lines defined using the point (4,2), should be deleted from the container `plc`.

`plc.deleteLine(1,2,5,6)` deletes the line ((1,2),(3,4)) from `plc` and returns true.