COMP 2012 Object-Oriented Programming and Data Structures

Lab 2 Constructors and Destructors

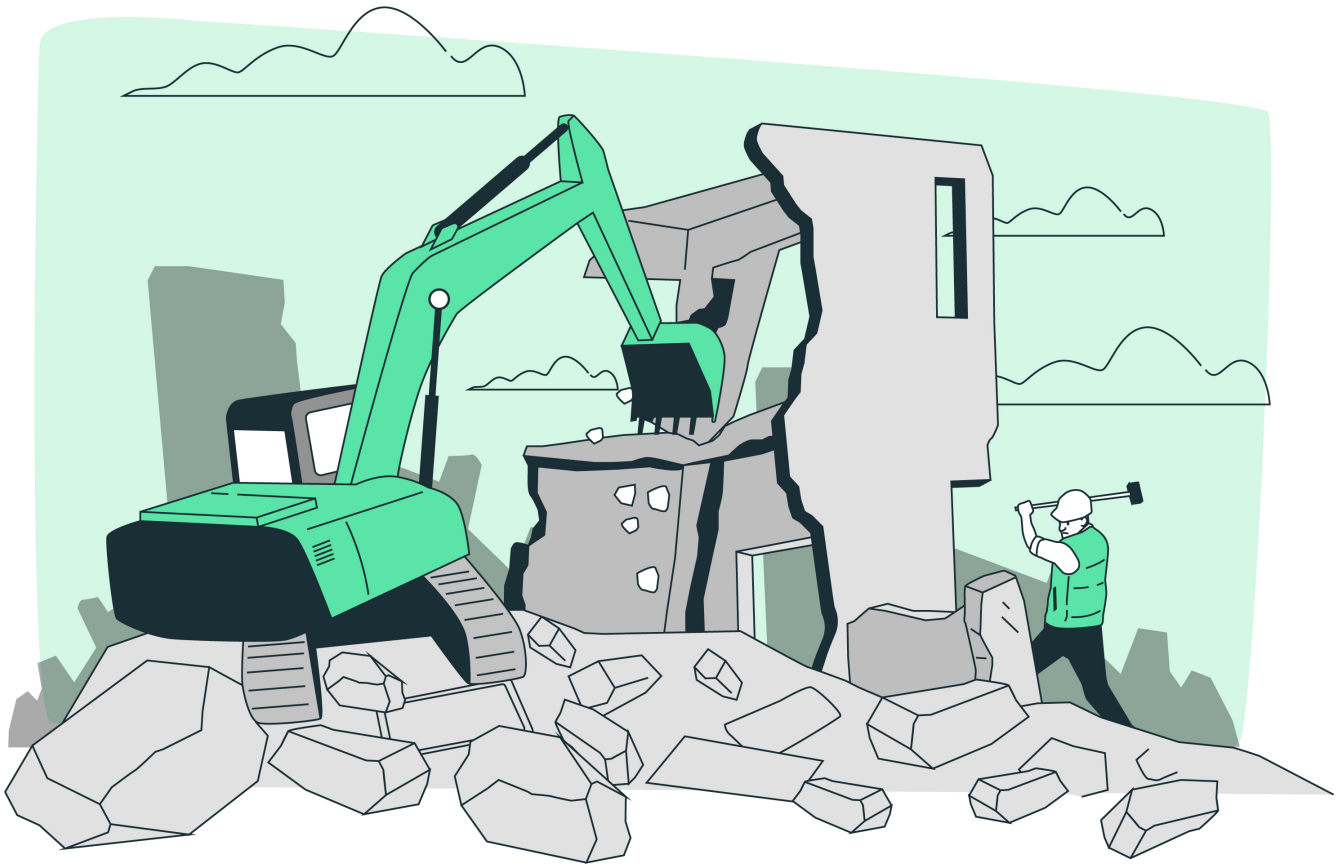

Image by storyset on Freepik

Overview

In this lab, you are required to implement a class named PointSet. A PointSet represents a set of points, and it maintains a dynamic array of point objects. Each point object has two double data members: x and y, which represents the x-coordinate and y-coordinate of a point.

You can download the source files in a zip package HERE.

This is a sample output of the program (`lab2.exe`):

```
PointSet1:
Initialized by PointSet's conversion constructor
The PointSet is empty!
Remove last point
No points
The PointSet is empty!
Does PointSet1 collinear? true

PointSet2:
Initialized by PointSet's other constructor
Number of points: 4
points:
(0,0)
(1.5,2)
(3,4)
(7.5,10)
Does PointSet2 collinear? true

PointSet3:
Initialized by PointSet's copy constructor
Remove last point
Add point (3, 1.7)
Number of points: 4
points:
(0,0)
(1.5,2)
(3,4)
(3,1.7)

Add point (5, 5)
Insufficient array space
Number of points: 4
points:
(0,0)
(1.5,2)
(3,4)
(3,1.7)
Does PointSet3 collinear? false
```

You are given the class definition and implementation of the Point class (`Point.h` and `Point.cpp`):

```
class Point
{
  public:
    Point();                    // Default constructor. It creates a point (0,0)
    Point(double x);            // Conversion constructor that can convert a double to a Point object
                                // Assign x to data member x, and set y to 0
    Point(double x, double y); // Construct a point with the given coordinates
    bool equal(const Point& p) const; // Check whether two points are the same
    void print() const;         // Print the point
    double getX() const;        // Get x coordinate
    double getY() const;        // Get y coordinate

  private:
    double x, y;                // Coordinates
};
```

You are also given the class definition and partially-completed implementation of the PointSet class (`PointSet.h` and `PointSet.cpp`):

```
class PointSet
{
  public:
    // Conversion constructor
    // Dynamically allocate an array of Points with capacity "capacity" and make it pointed by the data
member "points".
    // It also initializes the data member "capacity" and set numPoints to 0.
    PointSet(int capacity);

    // Other constructor
    // Dynamically allocate an array of Points with capacity equals to the parameter "numPoints" and
make it pointed by the data member "points".
    // Copy all the points in the parameter "points" over.
    // Remember to set all the corresponding data members properly.
    PointSet(const Point points[], int numPoints);

    // Copy constructor - Perform deep copy
    PointSet(const PointSet& s);

    // Destructor
    // Deallocate all the dynamically allocated memory to prevent memory leak
    ~PointSet();

    // Add a point p to the end of the point set, output (cout) the message "Insufficient array space"
and
    // do nothing else if there are already "capacity" number of points in the point set.
    void addPoint(const Point& p);

    // Remove the last point (the one with the largest index) in the set,
    // output (cout) the message "No points" and do nothing if the set has no points at all.
    void removeLastPoint();
```

Lab Tasks

Your task is to implement all constructors and destructor as well as four member functions for the `PointSet` class. Specifically, you are required to add appropriate lines of code in 8 different places in `PointSet.cpp` (**TODO #1 ~ TODO #8**).

TODO #1: PointSet(int capacity)
The conversion constructor dynamically allocates an array of `Points` with capacity `capacity` and makes it pointed by the data member `points`. It also initializes the data member `capacity` and sets `numPoints` to 0.

TODO #2: PointSet(const Point points[], int numPoints)
The other constructor dynamically allocates an array of `Points` with capacity equals to the parameter `numPoints` and makes it pointed by the data member `points`. It copies all the points in the parameter `points` over.

TODO #3: PointSet(const PointSet& s)
The copy constructor performs a deep copy of the parameter `s`.

TODO #4: ~PointSet()
The destructor deallocates all the dynamically allocated memory associated with the `PointSet` class to prevent memory leaks.

TODO #5: void addPoint(const Point& p)
The member function adds a point `p` to the end of the point set, outputs (cout) the message "Insufficient array space" and does nothing else if there is already `capacity` number of points in the point set.

TODO #6: void removeLastPoint()
The member function removes the last point (the one with the largest index) in the set, outputs (cout) the message "No points" and does nothing if the set has no points at all.

TODO #7: bool isCollinear() const
The member function returns true if all the points in the point set are collinear, lying on the same line; it returns false otherwise. Accordingly, it will always return true if the number of points in the point set is less than or equal to 2. You might refer to the algorithm for checking point collinearity, as suggested below.

TODO #8: bool isCollinear_3points(const Point p1, const Point p2, const Point p3) const
The member function returns true if points `p1`, `p2`, and `p3` are collinear, lying on the same line; it returns false otherwise. You might refer to the algorithm for checking point collinearity, as suggested below.

Algorithm for Checking Point Collinearity

To determine whether 3 given points p_1, p_2, p_3 are collinear, we could compute the area of the triangle formed by these 3 points using the following formula:

$$AREA = \frac{1}{2} \mid x_1(y_2 - y_3) + x_2(y_3 - y_1) + x_3(y_1 - y_2) \mid$$

where, (x_1,y_1), (x_2,y_2), and (x_3,y_3) are the coordinates of p_1, p_2 and p_3, respectively.

p_1, p_2, p_3 are collinear if and only if AREA = 0

Now, if we want to determine whether a set of N points { p_1, p_2, ..., p_N } are collinear, we could use the following algorithm (in pseudo-code):

- If N ≤ 2, returns `true`
- Otherwise, returns (isCollinear_3points(p_1, p_2, p_3) `&&` isCollinear_3points(p_2, p_3, p_4) `&&` ... `&&` isCollinear_3points(p_{N-2}, p_{N-1}, p_N))

where isCollinear_3points(p_a, p_b, p_c) returns `true` if p_a, p_b, p_c are collinear (or, equivalently, the area of the triangle formed by p_a, p_b, p_c equals to 0); returns `false` otherwise.

Compile and Test

After you have finished all tasks (**TODO #1 ~ TODO #8**), you can compile the files by typing `make all` in the terminal to produce the executable `lab2.exe`. Your program must be able to re-generate **exactly the same output as the sample output** that has been shown in the overview section (also included in the zip package as `output.txt`). You should read the `main.cpp` file carefully to understand how the sample output is generated.

Since this lab involves dynamic memory allocation, it is important for you to learn how to check for possible memory leaks in your programs and have them properly fixed before you submit your lab assignment. You risk losing marks if you submit a program that would cause memory leaks.

Tutorials on how to use software tools, such as Dr. Memory and valgrind, for the detection of possible memory leaks have been made available at the "Self-learning Center" on the Course Homepage.

Submission Deadline and Guidelines:

The lab assignment is due 10 minutes after the end of your lab session.
We will use an online grading system ZINC to grade your lab work. Therefore, you are required to put the following file in a zip file and have it uploaded to ZINC for assessment:

- `PointSet.cpp`

You can make multiple submissions to ZINC before the deadline. Only the last submission will be graded.

Extra Notes

You might have noticed that, in `Point.cpp`, we simply implement the function `equal()` in the following way:

```
bool equal(const Point& p) const
{
  return (x == p.x && y == p.y);
}
```

Does this equal function really work? If not, how should it be modified?

This is not a mandatory task. And you are encouraged to solve the problem and learn to handle the comparison of floating values properly.

Menu

- Overview
- Source Files
- Lab Tasks
- Submission Guidelines

Page maintained by

Namkiu Chan
Last Modified: 03/02/2023 04:43:22

Homepage

Course Homepage