**Due : Sept. 22 (9/22), 13:00**

## Note: The submission format has changed from HW0. Please carefully read the section about the submission procedure.

## Overview

This algorithm consists of three parts: implementing a linearly linked list, implementing a circularly linked list, and implementing a basic polygon class.

## General Notes

- Do not use Eclipse. We recommend Sublime Text (Linux/Mac/Windows), Atom (Linux/Mac/Windows), Notepad++ (Windows), or TextWrangler (Mac).
  - Code that contains the line "package ..." at the beginning of the file breaks our autograder and Eclipse automatically adds that line.
- Do not change any method or class signatures. You should only edit inside of the functions. If your code changes any class or method names or signatures, you will receive an automatic 0. You should not implement any other functions or instance variables besides the ones that are provided, unless explicitly allowed.
- Make sure your code compiles. Non-compiling code will automatically receive a 0. If you have a problem that is causing you to not be able to compile, it may be better to just comment out the incorrect code and return a dummy value (something like null or -1) so the rest can compile.
- Make sure that your code does not print out anything (there should be no System.out.println in your code). You will receive an automatic 0 if your code outputs something to STDOUT during the tests.

- To ensure that your code will be accepted by the autograder, you should submit your code on YSCEC, download it again, unzip it, recompile it and check the provided test suite. This way, you know that the file you are submitting is the correct one.

## Linked Lists

As you learned in the lecture, the linked list data structure is one of the most fundamental data structures in computer science. In comparison to the standard, contiguous memory array, linked lists sacrifice look up speed for faster insertion and deletion. In the first part of this assignment, you will implement a singularly (linear) linked list. Your linked list will contain both a *head* and *tail* pointer and will have support for several common operations such as insertion, deletion, searching, and combining lists.

In some implementations, a designated *head* node always exists and can not be deleted or changed (even if the list is cleared or a node is added to the front). However, we will not use this convention. Instead, an empty list should have a null *head* pointer, and the *head* can change if a node is added/deleted from the front. The same applies to the Circularly Linked List.

## Circularly Linked Lists

There are many variations on the basic linked list that are used frequently in practice. Two common ones are doubly linked lists (where each node has a pointer to a *next* and *previous* node) and circularly linked lists (where the *tail* of the linked list points to the *head*).

You will implement a doubly, circularly linked list. This implementation will only have a *head* pointer. Again, you will implement several common operations related to circularly linked lists.

## Polygons

One common use of circularly linked lists is representing the boundary of a 2-dimensional polygon. Since the boundary of a polygon can be represented by a circular, ordered list of vertices, we can simply use our previously implemented circular linked list with a custom Point class (which we have implemented for you) to handle the creation and manipulation of the polygon.

You will implement a basic polygon built on top of a circularly linked list. You do not have to implement many of the methods required by the linked list/circularly linked list sections (since you already did it in the circularly linked list implementation), but you will have to implement a special algorithm, *point-in-polygon*.

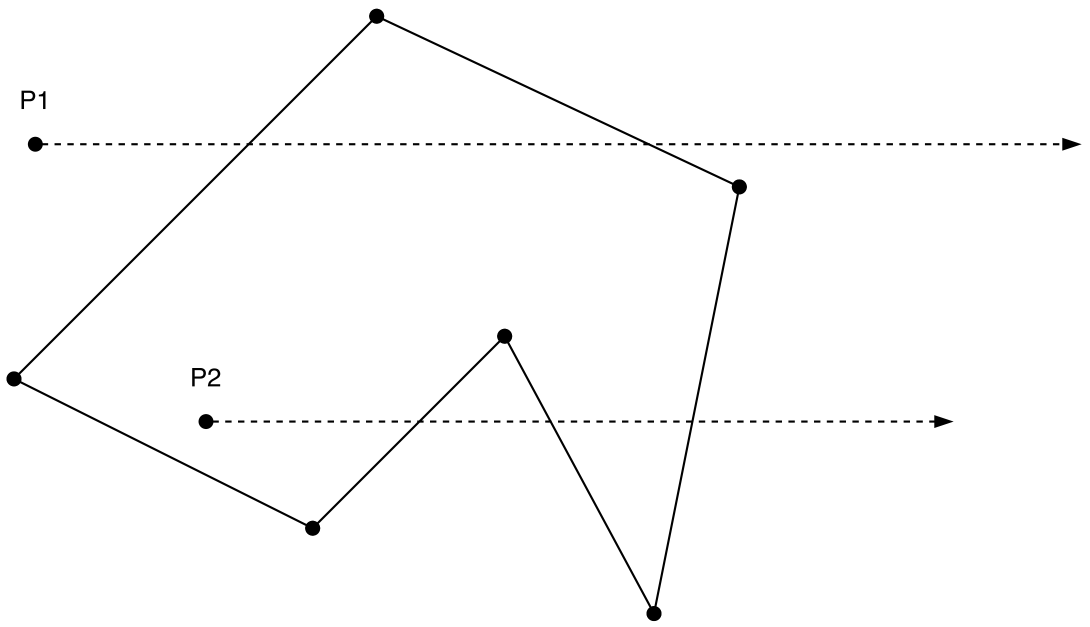
## Point in Polygon

Given a polygon $P$, and a point $p$, one important question is to determine whether or not $p$ lies inside $P$. A popular algorithm for this problem is known as the $Ray - Casting$ algorithm. The algorithm is as follows:

```
Data: Polygon $P$; Point $p$
Result: Whether or not $p$ is inside $P$
$r$ = a ray starting at $p$ extending to $\infty$
$count = 0$
for edge $e \in P$ do
    if $r$ intersects $e$ then
        $count = count + 1$
    end
end
if count is odd then
    return True
else
    return False
end
```

**Algorithm 1:** Ray-Casting Algorithm

Here, an edge is defined as two sequential points in the polygon boundary. Below you can see a demonstration of the algorithm. The ray extending from point $P1$, which is outside the polygon, crosses two edges while the ray extending from $P2$, which is inside, crosses 3.



While in practice there are some edge cases that you should consider for this algorithm (if the point to be tested is a point on the boundary of the polygon, if the polygon has 3 points but they are all colinear, etc), we will not be testing them in this assignment. All tested polygons will have non-zero area (if they have at least 3 points) and will not self-intersect.

For this algorithm any ray will work in general, but in this assignment you should use a ray of slope 0 (meaning it extends to infinity towards the right of the point). If you use this slope, will guarantee that none of the tests have a case where the ray intersects the polygon at a point where two edges meet. If you use another slope, you will have to consider this edge case, which can be difficult.

One final note is that if the polygon contains fewer than 3 points, you should always return *False* for this algorithm.

## General Directions

- Write your name and student ID number in the comment at the top of the java files.
- Implement all of the required functions in the java files.
- You should not import anything that is not already included in the file.
- Pay careful attention to the required return types and edge cases.

## Submission Procedure

### This is different than the submission procedure from HW0.

To submit your homework, simply drag and drop each file individually into the attachment box on the YSCEC assignment page. You should **NOT** zip them or place them in any folder. For this assignment, you should submit only the following files:

- LinkedList.java
- CircularLinkedList.java
- Polygon.java
- ⟨student_id⟩.txt

Inside of the ⟨student_id⟩.txt file, you should include the following text, and write your name at the bottom.

*In completing this assignment, I pledge that I have not given nor received any unauthorized assistance.*

If this file is missing, you will get a 0 on the assignment. It should be named *exactly* your student id, with no other text. For example, *2017123456.txt* is correct while something like *2017123456_hw1.txt* will receive a 0.

## Testing

We have provided a small test suite for you to check your code. You can test your code by compiling and running the tester program. You will always get a notification if you fail any of the tests, but you may not get a notification if you passed.

Note that the test suite we use to grade your code will be much more rigorous than the one provided here (and not necessarily a superset of the provided tests). You should consider making your own test cases to check your code more thoroughly.