Objective(s):

a. To be able to create customed LinkedList data structure
b. Students are able to demonstrate their understanding on implementing Shunting Yard Algorithm.

(For the sake of simplifying the lab's technical difficulty, let's use a new Node.java class working with String (instead of modifying the previous lab int type attribute of nested Node class).

Task 1: Complete **Task1.java**. Similar to MyStack.java which we encapsulate a java collection. In this case, we'll use LinkedList

```
public class Task1<T> {
    private List<T> items = new LinkedList<>(); // remove(0) is O(1)
    public void enqueue(T d) {
        /* your code */
    }
    public T dequeue() {
        /* your code */
    }
    public T peek() {
        /* your code */
    }
    public boolean isEmpty() {
        return items.isEmpty();
    }
    public String toString() {
        StringBuilder sb = new StringBuilder();
        sb.append("top->");
        for (T item : items)
            sb.append(item).append("-> ");
        sb.append("bottom");
        return sb.toString();
    }
}
```

Task 2: To make Task1 iterable like java's collection class, we need to implements Iterable (hence, implement public Iterator<T> iterator() method. Save as Task1.java to MyQueueL_XXYYYY.java

```
public class MyQueueL_XXYYYY<T> implements Iterable<T> {

    private List<T> items = new LinkedList<>(); // remove(0) is O(1)

    …

    @Override
    public Iterator<T> iterator() {
        /* your code */
    }
}
```

```
static void task_2() {
    MyQueueL_XXYYYY<String> queue = new MyQueueL_XXYYYY<>();
    queue.enqueue("Apple");
    queue.enqueue("Banana");
    queue.enqueue("Cantaloupe");
    System.out.print("standard for each: ");
    for (var item : queue) {
        System.out.print(item + " ");
    }
    System.out.println();
    System.out.println("demo iterator");
    Iterator<String> iter = queue.iterator();
    char ch = 'n';
    while (iter.hasNext()) {
        String item = iter.next();
        if (item.indexOf(ch) != -1) {
            System.out.print(item + " ");
        }
    }
    System.out.println();
}
```

Task 3: Implement MyShauntingYard.java which contains

public static String infixToPostfix(String infixString) (with any auxiliary method required) so that it ..

```
private static int order(String c) {
    return switch (c) {
        case "+", "-" -> 1;
        case "*", "/" -> 2;
        default -> 0;
    };
}
public static String infixToPostfix(String infixString) {

}
```

```
static void task_3() { // compute InFix
    String inFix = "( 4 + 2 ) / 3 * ( 8 - 5 )";
    String postFix = MyShuntingYard.infixToPostfix(inFix);
    System.out.println("postFix= " + postFix);
    double ans = MyRPN.computeRPN(postFix);
    System.out.println(ans);
}
```

Task 4: Largest Island in a 2D Grid (BFS Approach)

```
public class Solution_XXYYYY {
  public int maxLandArea(int[][] grid) {
    int max = 0;
    int rows = grid.length;
    int cols = grid[0].length;
    boolean[][] visited = new boolean[rows][cols];

    for (int r = 0; r < rows; r++) {
        for (int c = 0; c < cols; c++) {
            if (grid[r][c] == 1 && !visited[r][c]) {
                int area = bfs(grid, visited, r, c);
                max = Math.max(max, area);
            }
        }

    return max;
  }
  private int bfs(int[][] grid, boolean[][] visited, int startR, int startC) {
    int[] dr = {-1, 1, 0, 0}; // up, down
    int[] dc = {0, 0, -1, 1}; // left, right
    int count = 0;

    /* your code */
  }
  private boolean isValid(int[][] grid, boolean[][] visited, int r, int c) {
    return r >= 0 && r < grid.length &&
           c >= 0 && c < grid[0].length &&
           grid[r][c] == 1 && !visited[r][c];
  }

  public static void main(String[] args) {
    int[][] grid = {
            {1, 1, 0, 0},
            {1, 0, 0, 1},
            {0, 0, 1, 1},
            {1, 1, 0, 0}
        };
    Solution s = new Solution();
    int result = s.maxLandArea(grid);
    System.out.println("Largest land area: " + result);
  }
}
```

You are given a 2D grid of integers where: **1** represents land, **0** represents sea

A land cell is connected to its adjacent land cells horizontally or vertically (not diagonally).

Your task is to write a **Solution_XXYYYY.java** that finds and returns the size of the largest connected land region in the grid.

Submission: MyQueueL_XXYYYY.java, MyShuntingYard.java and Solution_XXYYYY.java

Due date: TBA