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 class working with String (instead of modifying the previous lab int type). Given MyStackL.java with Node.java

Task 1: Complete MyQueueL.java. The structure of this class employs Node tail attribute so that accessing the last node can be achieved easily.

```
package code;

public class MyQueueL {
  private Node head, tail;

  public MyQueueL() {
        head = tail = null;
  }

  public void enqueue(String d) {
    Node n = new Node(d);
    if (head == null)
       head = tail = n;
    /* your code */
  }
  public String dequeue() {
    if (isEmpty()) return "";
    String value = head.value;
    head = head.next;
    /* your code */
    return value;
  }
  /* isFull() isEmpty() */
  public String top() {
    return head.value;
  }
  public String top() {
    return head.value;
  }
  public String getLast() {
    return tail.value;
  }
  public String dumpToString() {
    StringBuffer sb = new
                   StringBuffer();
    Node n = head;
    while (n != null) {
      sb.append(n.value + " ");
      n = n.next;
    }
    return sb.toString();
  }

  @Override
  public String toString() {
    StringBuilder sb = new
                  StringBuilder();
    sb.append("First->");
    Node temp = head;
    while (temp != null) {
      sb.append(
         temp.value).append("->");
      temp = temp.next;
    }
    sb.append("Last");
    return sb.toString();
  }
}
```

Notice the two helper method created –String getLast() and String dumpToString(). getLast() clearly shows the benefits of using tail in the queue structure while dumpToString() provides a cleaner output format than toString().

Use the following main method to test your MyQueueL.java completion.

```
import SQ.MyQueueL;
import SQ.MyRPN;
import SQ.MyShuntingYard;

public class L6_Infix_Main {
    public static void main(String[] args) {
        demo1();
        // String infixString = "( 4 + 2 ) / 3 * ( 8 - 5 )";
        // //expect 4 2 + 3 / 8 5 - *
        // if (args.length > 0)
        //     infixString = args[0];
        // computeInfix(infixString);
    }
…
    private static void demo1() {
        System.out.println("-----MyQueueL Tester-----");
        MyQueueL queue = new MyQueueL();
        queue.enqueue("1");
        queue.enqueue("3");
        queue.enqueue("5");
        queue.enqueue("7");
        System.out.println(queue.dumpToString());
        System.out.println("Top " + queue.top());
        System.out.println("Last " + queue.getLast());
        System.out.println("Dequeue " + queue.dequeue());
        System.out.println("Dequeue " + queue.dequeue());
        System.out.println("Dequeue " + queue.dequeue());
        System.out.println("Dequeue " + queue.dequeue());
        queue.enqueue("9");
        System.out.println(queue);
        System.out.println("----MyQueueL Test End----");
    }
}
```

Task 2: use the Shunting Yard pseudo code to complete MyShuntingYard.java

```
package code;
…
public class MyShuntingYard {
  private static int order(String c) {
        return switch (c) {
            case "+", "-" -> 1;
            case "*", "/" -> 2;
            default -> 0;
        };
  }
  public static String infixToPostfix(String infixString) {
    MyQueueL queue = new MyQueueL();
    MyStackL stack = new MyStackL();
    String resultPostfixString = "";
    StringTokenizer st = new StringTokenizer(infixString);
    while (st.hasMoreTokens()) {
        String t = st.nextToken();
        if (MyRPN.isNumeric(t))
            queue.enqueue(t);
        else if (t.equals("(")) {
            stack.push(t);
        } else if (t.equals(")")) {
            while (!stack.peek().equals("(")){
                queue.enqueue(stack.pop());
            }
            stack.pop(); //discard "("
        } else {
            if(!stack.isEmpty()) { // double lovely bug
                /* your code */
            }
            /* your code */
        }
        // println("current q = " + queue.dumpToString());
        }
        /* your code */
        resultPostfixString = queue.dumpToString();
        return resultPostfixString; //"happy coding";
    }
}
```

Notice that our infix calculator handles only + - * and /, all of which have left-associativity property.

```
import SQ.MyQueueL;
import SQ.MyRPN;
import SQ.MyShuntingYard;

public class L6_Infix_Main {
    public static void main(String[] args) {
        // demo1();
        String infixString = "( 4 + 2 ) / 3 * ( 8 - 5 )";
        //expect 4 2 + 3 / 8 5 - *
        if (args.length > 0)
            infixString = args[0];
        computeInfix(infixString);
    }
    public static void computeInfix(String infixString) {
        String postfixString =
               MyShuntingYard.infixToPostfix(infixString);
        double ans = MyRPN.computeRPN(postfixString);
        System.out.println(ans);
    }
    private static void demo1() {
…
    }
}
```

Submission: MyQueueL_XXYYYY.java and MyShuntingYard_XXYYYY.java

Due date: TBA