COMP 2230 - Data Structure, Algorithm Analysis, and Program Design

Laboratory No. 3

Total: 60

Due date and time for submission: Midnight of Sunday, October 6, 2019

Method of submission: Moodle.tru.ca

The Linked List is very important library, which we will use in many other ADT's in future. Therefore, it is important to understand and practice using the LinkedListADT.

In this lab, we will have some exercise on Stack, LinkedList, ArrayList and Queue concepts. These exercises will help understand these concepts better.

Q1. [5 points] Implement Stack with the linked list ADT that you have created in the class.

Q2. [5 points] Implement Queue with the linked list ADT that you have created in the class.

Q3. [5 points] Write a program using stack to check a given expression has balanced symbols. Only valid balanced symbols are (){}[]. For example, the following expressions has balanced symbols:

{(a+b) – c},

{[2] + (2 * 6)}

However, ([4+5], ([4+4)], ((A+B) + [C-D]} are not balanced expression.

Algorithm:

```
a) Create a stack
b) While (end of input is not reached)
    a. If the character read is not a symbol to be balanced,
       ignore it.
    b. If the character is an opening symbol like (, [, {, push it
       onto the stack.
    c. If it is a closing symbol like ),],}, then if the stack is
       empty report an error. Otherwise pop the stack.
    d. If the symbol popped is not the corresponding opening
       symbol, report an error.
c) At the end of input, if the stack is not empty report an error.
```

Q4. [10 points] Write a program to convert an infix expression to postfix expression using stack.

Algorithm:

```
a) Create a stack
b) For each character t in the input stream
        If(t is an operand)
            append t to the output.
        Else if (t is a right parenthesis)
                Pop and append to the output until a left
                parenthesis is popped (but do not append this
                parenthesis to the output).
        Else if(t is an operator or left parenthesis)
                Pop and append to the output until one of the
                lower priority than t is encountered or a left
                parenthesis is encountered or the stack is empty.

                Push t
c) Pop and append to the output until the stack is empty.
```

Q5. [10 points] Write a program that will evaluate a valid postfix expression and print the result.

Q6. [10 points] Implement a queue using two stacks.

Algorithm:

```
a) Create two stacks, let's call it S1 and S2

For Enqueue: push new item in S1



For Dequeue:

a) if S2 is empty, fill it by popping each elements from S1 and
   pushing it onto S2.
b) Pop and return top element from the S2
```

Q7. [5 points] Is it possible to implement one stack using two queues? Discuss your solution. Also, analyze the running time of the stack operation.

Q8. [10 points] Given a stack of integers, how do you check whether each successive pair of numbers in the stack is consecutive or not. The pairs can be increasing or decreasing, and if the stack has an odd number of elements, the element at the top is left out of a pair. For example, if the stack of elements are [4, 5, -2, -3, 11, 10, 5, 6, 20], then the output should be true because each of the pairs (4, 5), (-2, -3), (11, 10), and (5, 6) consists of consecutive numbers. Write a static method called `public static boolean checkStackPairwiseOrder(StackADT<Integer> s)`, this method should return true if the pairs are either increasing or decreasing, false otherwise. Test your method with some test data and demo program.

Hint: *You might want to use Stack and Queue both for solving this problem.*

Note:

Make sure, you have exception handling code in place for each exceptional conditions. Also, make sure you have tested all the codes with various boundary conditions.

Please submit all the .java files and scree captures of test run of the code.

Grading Rubric

1. Code compiled without any error	*20%*
2. Good Comments	*20%*
3. All requirements are met	*40%*
4. Screenshots of several test runs are included	*10%*
5. Apply OOP principals • Modularity • Good use of classes • Encapsulation	*10%*

Note: All problems will be graded based on the above-mentioned rubric.