DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
Faculty of Engineering
University of Moratuwa

CS 2022: Data Structures & Algorithms

Lab 4 : Stack Implementation Using Linked List

Objectives: The aim of this lab session is to provide you with the understanding of how basic abstract data types can be implemented using linked data elements and their applications.

Learning Outcomes: After completing this lab the students should be able to;

1. Implement simple data structures using linked structures.
2. Use the implemented data structures as libraries for simple applications.

In this lab session you will;

- Use the linked list implementation to implement a stack (and a queue).
- Use the stack implementation to evaluate postfix notation.

You are expected to use java programming language for completion of this lab exercise.

- You should upload your completed code as a **single .java file** in the Moodle page.
- The .java file should contain 4 classes – `Node`, `CS2022LinkedList`, `CS2022LinkedStack`, `CS2022PostfixCalculator` and `IndexN0_lab3` classes.
- File names should be **`lab4_IndexN0.java`**.

Instructions for testing the code go to the directory where your .java file and input files are saved.

a. To compile your code: javac **`IndexN0_lab4.java`**
b. To run your code for input 1: java **`IndexN0_lab4 input1.txt`**
c. To run your code for input 2: java **`IndexN0_lab4 input2.txt`**

Task 1: ***Implementing a Stack***

In the second task of the lab, you are expected to implement a stack using the lined list you implemented as the first task of the lab. Your stack implementation should support push, pop and peek operations. Using your lined list library as a library (without any modification to the liked list) implement a stack which supports following operations.

`void init_stack()`

This function will initialize the stack to an empty stack.

`booelan is_empty()`

This function will check and see if the stack is empty. It will return true if the list is empty and false otherwise.

`booelan push(E element)`

This function will place item on the top of the stack. It will return true if the insertion is successful and false otherwise.

`E pop()`

This function will remove the top element in the stack and return it as element provided that the stack is not empty.

You can use the following definitions as the starting point for your implementation.

```
package CS2022Labs;

public class CS2022LinkedStack<E> {

	private CS2022LinkedStack<E> elements;
	//Add any additional things you need here.

	public void init_stack(){
	}

	public boolean is_empty(){
	}

	public boolean push(E element){
	}

	public E pop(){
	}

}
```

Task 2: ***Evaluate Postfix Notation Using Stack Implementation***

In the second task of the lab, you are expected to implement a calculator to evaluate postfix notation using the stack you implemented as the first task of the lab.

(Details of the Postfix notation can be found here: http://c2.com/cgi/wiki?PostfixNotation)

Assume the inputs are real numbers and only arithmetic operations addition ('+'), deduction ('-') , multiplication ('*'), and division ('/') are included in the calculation.

Example:

```
Input   : 1 2 + 3 * 6 + 2 3 + /
Output  : 3
```

Create a class CS2022PostfixCalculator which contain following implementation:

```
float calculate(String S)
```

This function will take a string containing postfix arithmetic operation and return the results.

```
package CS2022Labs;

public class CS2022PostfixCalculator{

      private String input;
      //Add any additional things you need here.

      public float calculate(String s){
      }

}
```

Task 3: Implementing the main method

Your main method should be able to read the command from the file. Input file name should be taken as a command line argument when you run the program (e.g. : "java IndexNo_lab3 input.txt" should produce results.out for input1.txt file). Following are the commands.

- push
- pop
- calculate

Command push will be followed by a list of elements to be pushed into stack. Command pop should pop the top element in the stack, print and append the result to 'result.out'. Command 'calculate' is followed by an string of postfix notation to be evaluated. You should evaluate the string print and append the result to 'result.out'.

Hint : You may test your implementation using command line inputs and outputs before reading input from file and writing the results to the file. You can use this resource (http://www.tutorialspoint.com/javaexamples/file_append.htm) to learn how to append new line to end of an existing file.

```
public class lab4_IndexNo {

      public static void main(String[] args){
                  //Take the input file name from 'args'.
                  //Run the commands in the input file.
                  //Print the relevant outputs and append the result to result.out
            }

}
```

Sample input.txt

```
push 3,40,20
pop
pop
push t
pop
calculate 1 2 + 3 * 6 +
calculate 2 3 + 5 1 * /
```

Sample result.out

```
20
40
t
15
1
```

Extra Exercise: ***Implementing a Linked Queue***

This exercise will not be evaluated. Please do not submit.

In the third task of the lab, you are expected to implement a queue using the lined list you implemented as the first task of the lab. Your queue implementation should support enqueue, dequeue and search operations. Using your lined list library as a library (without any modification to the liked list) implement a queue which supports following operations.

`void init_queue()`

This function will initialize the stack to an empty queue.

`boolean is_empty()`

This function will check and see if the queue is empty. It will return true if the list is empty and false otherwise.

`boolean enqueue(E element)`

This function will add the item at the end of the queue. It will return true if the insertion is successful and false otherwise.

`E dequeue()`

This function will remove the first element in the queue and return it as element provided that the queue is not empty.

You are expected to decide the appropriate design for the class "CS2022LinkedQueue", which will hold the details of the queue and support efficient enqueue and dequeue operations.