IARE

IARE
INSTITUTE OF
AERONAUTICAL ENGINEERING
(An Autonomous Institute affiliated to JNTUH, Hyderabad)
Dundigal, Hyderabad - 500 043

LABORATORY WORK BOOK

Name of the Student : N. Ravi Chandrika

Class : CSD-B Semester : IIrd Semester

Course Code : ACSD11 Course Name : OS Laboratory

Name of the Course Faculty : Mr. A. Srikanth Faculty ID : IARE10652

Exercise Number : 5 Week Number : 5 Date : 05/10/2024

Roll Number									
2	3	9	5	1	A	6	7	B	3

S. No.	Exercise Number	EXERCISE NAME	MARKS AWARDED						
			Aim/ Preparation	Algorithm / Procedure / Performance in the Lab		Source Code / Calculations and Graphs	Program Execution / Results and Error Analysis	Viva - Voce	Total
			4	4		4	4	4	20
1	5-1	Implementation of Stack							
2	5-2	Balanced parentheses checking							
3	5-3	Evaluation of postfix expression	4	2	2	4	4	4	20
4	5-4	Infix to postfix expression conversion							
5	5-5	Reverse a Stack.							
6									
7									
8									
9									
10									
11									
12									

Signature of the Student

Signature of the Faculty

START WRITING FROM HERE

5.1 Aim: A stack is a linear data structure that stores item in a last-in, first-out (LIFO) or first-in, last-out (FILO) manner. In a stack a new element is added at one end of an element is removed from another end.

Code:

```
import java.util.Stack;
public class Implementation_of_Stack {
    public static void main(String args[])
        Stack<Objects> Stack = new Stacks();
        Stack.push(10);
        Stack.push("Chandrika");
        Stack.push(30);
        System.out.println("Popped Element: " + Stack.pop());
        System.out.println("Top Element: " + Stack.peek());
        System.out.println("Is Stack Empty?" + Stack.isEmpty());
        System.out.println("Stack is Size:" + Stack.size());
    }
}
```

Output:

Popped element : 30

Top Element : Chandrika

Is stack Empty? False

Stack size : 2.

ROLL NUMBER :

5.2 Aim: Given an expression string write a java program to find whether a given string has balanced paranthesis or not.

Code:

```
import java.util.*;
class BalancedParanthesesChecker {
    public static void main(String args[]) {
        String expression = "{(a+b)*[c-d]}";
        Stack<Character> stack = new stack<>();
        boolean isBalanced = True;
        for(char ch : expression.toCharArray()) {
            if (ch == '{' || ch == '[' || ch == '(') {
                stack.push(ch);
            }
            else if (ch == '}' || ch == ')' || ch == ']') {
                if(stack.isEmpty()) {
                    isBalanced = False;
                    break;
                }
                Char tem = Stack.pop()
                if(!(tem == '[' && ch == ')') && !(tem == '[' && ch == ']')
                    isBalanced = False;
                    break;
                }
            }
        }
    }
```

```
        if(!stack.isEmpty()){
            isBalanced = false;
        }
        System.out.println(isBalanced);
    }
}
```

Input: {(a+b)*[c-d)}

Output: False.

5-3 Aim: Given a postfix expression, the task is to evaluate the postfix expression. Postfix expression: The expression of the form *ab operator * (ab+)i.e when a pair of operands is followed by an operator.

Code:

```
import java.util.*;
class PostfixEvaluator {
    public static void main(String args[]) {
        String expression = "2 3 1 * + 9 - ";
        Stack<Integer> stack = new Stack<>();
        String[] tokens = expression.split(regex: " ");
        for (String token: tokens){
            if (token.equals(anObject:"+") || token.equals(anObject:"-")
                int operand2 = stack.pop();
                int operand1 = stack.pop();
```

```
switch (token) {
    case "+":
        Stack.push (operand1 + operand2);
        break;
    case "-":
        Stack.push (operand1 - operand2);
        break;
    case "*";
        Stack.push(operand1 - operand2);
        break;
    case "/";
        Stak.push (operand1 /operand2);
        break;
    default:
        System.out.println (x:" Invalid Operator");
    }
}
else {
    Stack.push (Integer.parseInt (token));
}
}
System.out.println(stack.pop());
}
}
```

Input : 2 3 1 * + 9 −

Output : -4.

ROLL NUMBER :

5.4 Aim: For a given infix expression, convert it to postfix expression.
Infix expression: The expression of the form "a operator b" (a+b) i.e when an operator in b/w every pair of operands.

Code:

```
import java.util.*;
class InfixtoPostfixConverter {
    private static int precedence(char operator) {
        switch(operator) {
            case '+':
            case '-':
                return 1;
            case '*':
            case '/':
                return 2;
            case '^':
                return 3;
            default:
                return -1;
        }
    }
    public static void main(String args[])
        String infix = "((D-A)^G1+(H/I))-K";
        StringBuilder postfix = new StringBuilder();
        Stack<Character> stack = new Stack<>();
    for(char c : infix.toCharArray()) {
        if(Character.isLetterOrDigit(c)) {
            postfix.append(c).append(c:' ');
```

```
        }
        else if (cc = '(') {
            Stack.push(cc);
        }
        else if (cc == ')') {
        while(!Stack.isEmpty() && stack.peek() != '(') {
            postfix.append(stack.pop()).append(cc:' ');
        }
        Stack.pop();
    }
    else if (cc=='+' || c=='-' || c=='*' || c=='/' || c=='^') {
        while (!stack.isEmpty() && precedence(Stack.peek())) {
            postfix.append(Stack.pop()).append(cc:' ');
        }
        Stack.push(cc);
    }
}
while (!stack.isEmpty()) {
    postfix.append(Stack.pop()).append(cc:' ');
}
System.out.println(postfix.toString().trim());
}
}
```

Input: ((D-B)*G+(A/H))-R

ROLL NUMBER :

Output : D B - G * A H / I R - :

5.5 Ans: The Stack is a linear data structure which works on the LIFO concept. LIFO stands for last-in, first-out. In the stack, the insertion & deletion are possible, at the one end is called top of the stack.

Code:

```
import java.util.*;
public class StackReversal {
    public static void main(String args[]) {
        Scanner scanner = new Scanner(System.in);
        String[] input = scanner.nextLine().split("regex:" ");
        Stack<Integer> stack = new Stack<>();
        Stack<Integer> result = new Stack<>();
        for (String s: input) {
            stack.push(Integer.parseInt(s));
        }
        int n = stack.size();
        for (int i=0; i<n; i++) {
            result.push(stack.pop());
        }
        for (int i=0; i<n; i++) {
            System.out.print(result.get(i)+" ");
        }
    }
```

```
        scanner.close();
    }
}
```

Input: 1 2 3 4 5

Output: 5 4 3 2 1.