CS162 ASSIGNMENT 10

NAME:

ARCHIT AGRAWAL

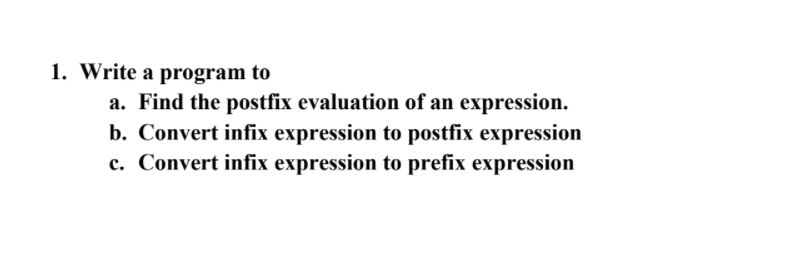
ROLL NO. :

202052307

SECTION:

A

**Question**



***CODE***

import java.util.\*;

class Node<T>{

    T data;

    Node next;

    public Node(){}

    public Node(T element){

        this.data = element;

    }

    public Node(T element, Node addr){

        this.data = element;

        this.next = addr;

    }

}

class LinkedList<T>{

    protected Node<T> head;

    protected int size;

    public LinkedList(){

        head = null;

        size = 0;

    }

    //The method append adds an element at the end (index = size) of the linked list.

    public void append(T data){

        Node<T> newNode = new Node<T>(data);

        if (head == null) {

            head = newNode;

        } else {

            Node <T> lastNode = head;

            while (lastNode.next != null) {

                lastNode = lastNode.next;

            }

            lastNode.next = newNode;

        }

        size++;

    }

    //The method add adds a given element at the given index.

    public void add(int index, T data){

        if(index < 0 || index > size){

            throw new IndexOutOfBoundsException("Index = "+index +" Size = "+size);

        }

        if(index == 0){

            Node<T> newNode = new Node<T>(data);

            newNode.next = head;

            head = newNode;

        } else {

            Node<T> newNode = new Node<T>(data);

            Node<T> prevNode = head;

            int i = 0;

            while(i < index - 1){

                prevNode = prevNode.next;

                i++;

            }

            newNode.next = prevNode.next;

            prevNode.next = newNode;

        }

        size++;

    }

    //The method removeKey removes the first occurence of the element passed to it from the linked list.

    public void removeKey(T key){

        if(size == 0){

            return;

        }

        if(head.data == key){

            head = head.next;

        } else {

            Node<T> currNode = head.next;

            Node<T> prevNode = head;

            while(currNode != null){

                //to prevent NullPointerException

                if(currNode.data.equals(key)){

                    prevNode.next = currNode.next;

                    break;

                } else {

                    prevNode = currNode;

                    currNode = currNode.next;

                }

            }

        }

        size--;

    }

    //The method removeIndex removes the element from the index passed to it and returns the removed element.

    public T removeIndex(int index){

        checkIndex(index);

        T removed;

        if(index == 0){

            removed = head.data;

            head = head.next;

        } else {

            Node<T> currNode = head.next;

            Node<T> prevNode = head;

            int i = 0;

            while(i < index - 1){

                prevNode = prevNode.next;

                currNode = currNode.next;

                i++;

            }

            removed = currNode.data;

            prevNode.next = currNode.next;

        }

        size--;

        return removed;

    }

    //This method prints the linked list in formatted way

    public void printLinkedList() {

        Node<T> currentNode = head;

        while(currentNode != null){

            System.out.print(currentNode.data + " -> ");

            currentNode = currentNode.next;

        }

        System.out.println();

    }

    public boolean isEmpty(){

        return size == 0;

    }

    public int size(){

        return size;

    }

    public void checkIndex(int index){

        if(index < 0 || index >= size){

            throw new IndexOutOfBoundsException("Index = "+index +" size = "+ size);

        }

    }

    public T get(int index){

        checkIndex(index);

        Node<T> currNode = head;

        int i = 0;

        while(i != index){

            currNode = currNode.next;

            i++;

        }

        return currNode.data;

    }

    public int indexOf(T data){

        Node<T> temp = head;

        int index = 0;

        while(temp != null){

            if(temp.data.equals(data)){

                return index;

            }

            index++;

            temp = temp.next;

        }

        return -1;

    }

}

class StackLL<T> extends LinkedList<T>{

    public StackLL(){

        super();

    }

    public void push(T data){

        super.append(data);

    }

    public T pop(){

        if(super.isEmpty()){

            throw new IllegalArgumentException("Nothing to pop, Stack is empty.");

        }

        return super.removeIndex(size - 1);

    }

    public T peek(){

        if(super.isEmpty()){

            throw new IllegalArgumentException("Stack is empty.");

        }

        return super.get(size - 1);

    }

    public int search(T data){

        int i = super.indexOf(data);

        return i == -1 ? -1 : (size - 1);

    }

    public void display(){

        super.printLinkedList();

    }

    public void append(T data){}

    public void add(int index, T data){}

    public void removeKey(T key){}

    public T removeIndex(int index){

        return null;

    }

    public void printLinkedList(){}

    public void checkIndex(int index){}

    public T get(int index){

        return null;

    }

}

public class Main {

    public static double evaluatePostfix(String postfix){

        StackLL<Double> stack1 = new StackLL<Double>();

        int x;

        for( x = 0; x < postfix.length(); x++){

            char ch = postfix.charAt(x);

            if(ch == ' ') continue;

            else if(Character.isDigit(ch)){

                double num = 0;

                while(Character.isDigit(ch)){

                    num = num\*10 + (ch-48);

                    x++;

                    ch = postfix.charAt(x);

                }

                x--;

                stack1.push(num);

            }

            else if((ch == '-' || ch == '+') && x<(postfix.length()-1) && postfix.charAt(x+1) != ' '){

                x++;

                char c = postfix.charAt(x);

                double n = 0;

                while(Character.isDigit(c)){

                    n = n\*10 + (c-48);

                    x++;

                    c = postfix.charAt(x);

                }

                x--;

                n = (ch == '-') ? (n\*-1) : n;

                stack1.push(n);

            }

            else{

                double a = stack1.pop();

                double b = stack1.pop();

                switch(ch){

                    case '+':

                        stack1.push(b+a);

                        break;

                    case '-':

                        stack1.push(b-a);

                        break;

                    case '\*':

                        stack1.push(b\*a);

                        break;

                    case '/':

                        stack1.push(b/a);

                        break;

                    case '^':

                        stack1.push(Math.pow(b,a));

                        break;

                }

            }

        }

        return stack1.pop();

    }

    public static int precedence(char op){

        if(op == '^') return 3;

        else if(op == '\*' || op == '/' || op == '%') return 2;

        else if(op == '+' || op == '-') return 1;

        else return -1;

    }

    public static String infixToPostfix(String infix){

        StackLL<Character> stack2 = new StackLL<Character>();

        String postfix = "";

        for(int x = 0; x < infix.length(); x++){

            char ch = infix.charAt(x);

            if(ch == ' ') continue;

            if(Character.isLetterOrDigit(ch)) postfix += ch;

            else if(ch == '('){

                stack2.push(ch);

            }

            else if(ch == ')'){

                while(stack2.peek() != '('){

                    postfix += stack2.pop();

                }

                stack2.pop();

            }

            else{

                 while(!stack2.isEmpty() && precedence(ch) <= precedence(stack2.peek())){

                    postfix += stack2.pop();

                }

                stack2.push(ch);

            }

        }

        while(!stack2.isEmpty()){

            postfix += stack2.pop();

        }

        return postfix;

    }

    public static String reverse(String infix){

        String rev = "";

        for(int x = 0;x < infix.length(); x++){

            char ch = infix.charAt(x);

            if(ch == '(') ch = ')';

            else if(ch == ')') ch = '(';

            rev = ch + rev;

        }

        return rev;

    }

    public static String infixToPrefix(String infix){

        StackLL<Character> stack3 = new StackLL<Character>();

        String infix\_rev = reverse(infix);

        int x;

        String prefix\_rev = "";

        for(x = 0;x < infix\_rev.length(); x++){

            char ch = infix\_rev.charAt(x);

            if(ch == ' ') continue;

            if(Character.isLetterOrDigit(ch)) prefix\_rev += ch;

            else if(ch == '('){

                stack3.push(ch);

            }

            else if(ch == ')'){

                while(stack3.peek() != '('){

                    prefix\_rev += stack3.pop();

                }

                stack3.pop();

            }

            else{

                while (!stack3.isEmpty() && (precedence(ch) < precedence(stack3.peek()) || (ch == '^' && precedence(ch) <= precedence(stack3.peek())))) {

                    prefix\_rev += stack3.pop();

                }

                stack3.push(ch);

            }

        }

        while(!stack3.isEmpty()){

            prefix\_rev += stack3.pop();

        }

        return reverse(prefix\_rev);

    }

    public static void main(String args[]){

        Scanner Sc = new Scanner(System.in);

        String str1,str2;

        System.out.println(" ");

        System.out.println("SAMPLE INPUTS AND OUTPUTS:");

        System.out.println("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

        System.out.println("Postfix Expression :  "+"20 40 100 + 2 / \* 50 / 100 \* -100 +");

        double val = evaluatePostfix("20 40 100 + 2 / \* 50 / 100 \* -100 +");

        System.out.println("Value of postfix expression : "+val);

        System.out.println(" ");

        System.out.println("Infix Expression : "+"(a+b)\*c-(d-e)\*(f+g)");

        System.out.println("Postfix Expression for above : "+infixToPostfix("(a+b)\*c-(d-e)\*(f+g)"));

        System.out.println(" ");

        System.out.println("Infix Expression : "+"(a+b)\*c-(d-e)\*(f+g)");

        System.out.println("Prefix Expression for above : "+infixToPrefix("(a+b)\*c-(d-e)\*(f+g)"));

        System.out.println("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

        System.out.println(" ");

        System.out.println("Enter a postfix expression to evaluate with spaces between each operand and operator");

        str1 = Sc.nextLine();

        System.out.println("Value of postfix expression : "+evaluatePostfix(str1));

        System.out.println(" ");

        System.out.println("Enter an infix expression");

        str2 = Sc.nextLine();

        System.out.println("Postfix Expression for above : "+infixToPostfix(str2));

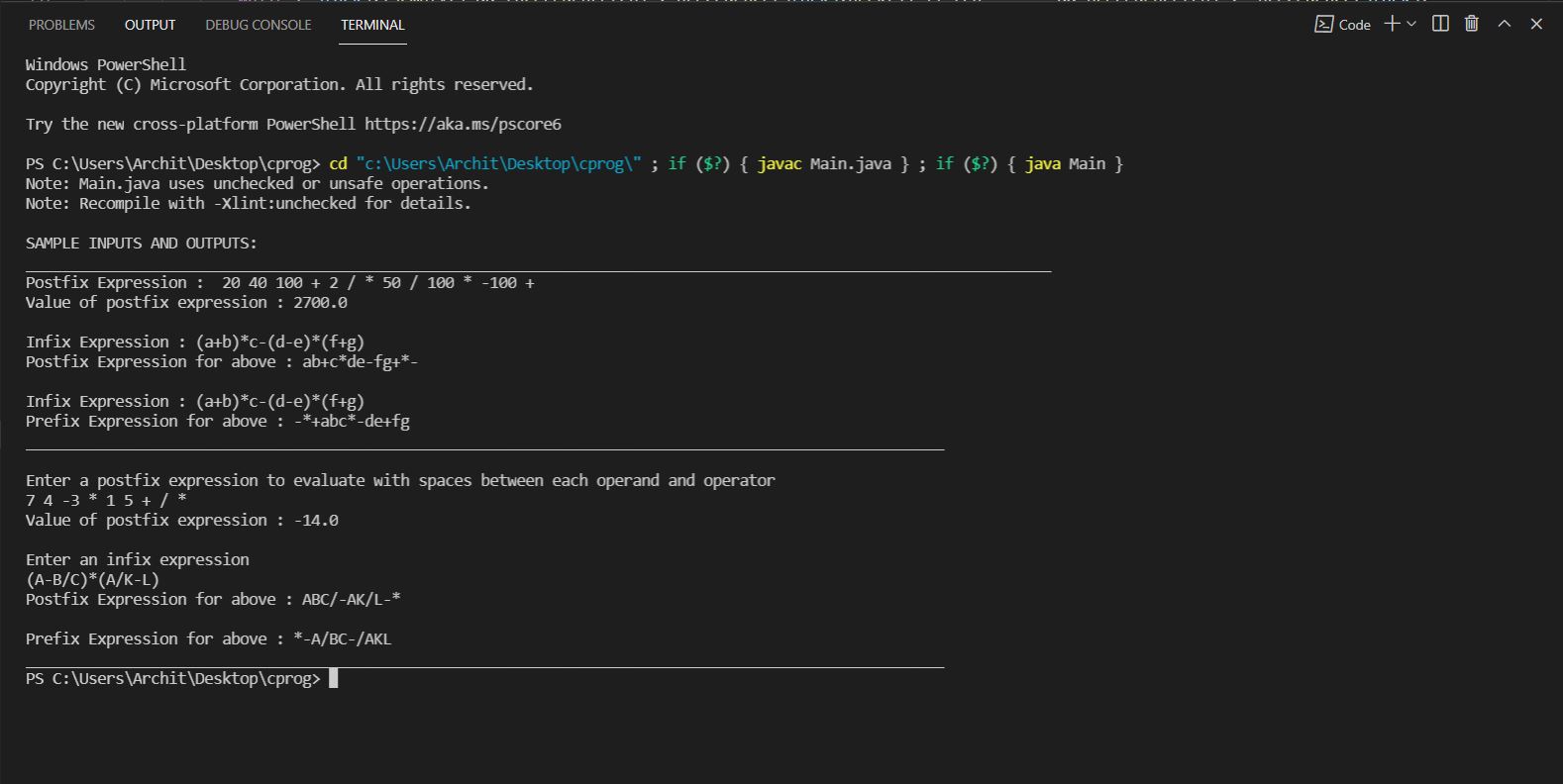
        System.out.println(" ");

        System.out.println("Prefix Expression for above : "+infixToPrefix(str2));

        System.out.println("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

    }

}

***OUTPUT***