CS162 ASSIGNMENT 10

NAME:

ARCHIT AGRAWAL

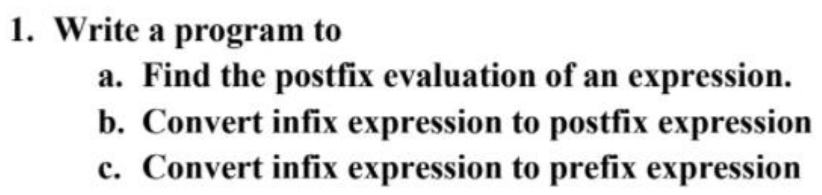
ROLL NO. :

202052307

SECTION:

A

Question


1. Write a program to
    a. Find the postfix evaluation of an expression.
    b. Convert infix expression to postfix expression
    c. Convert infix expression to prefix expression

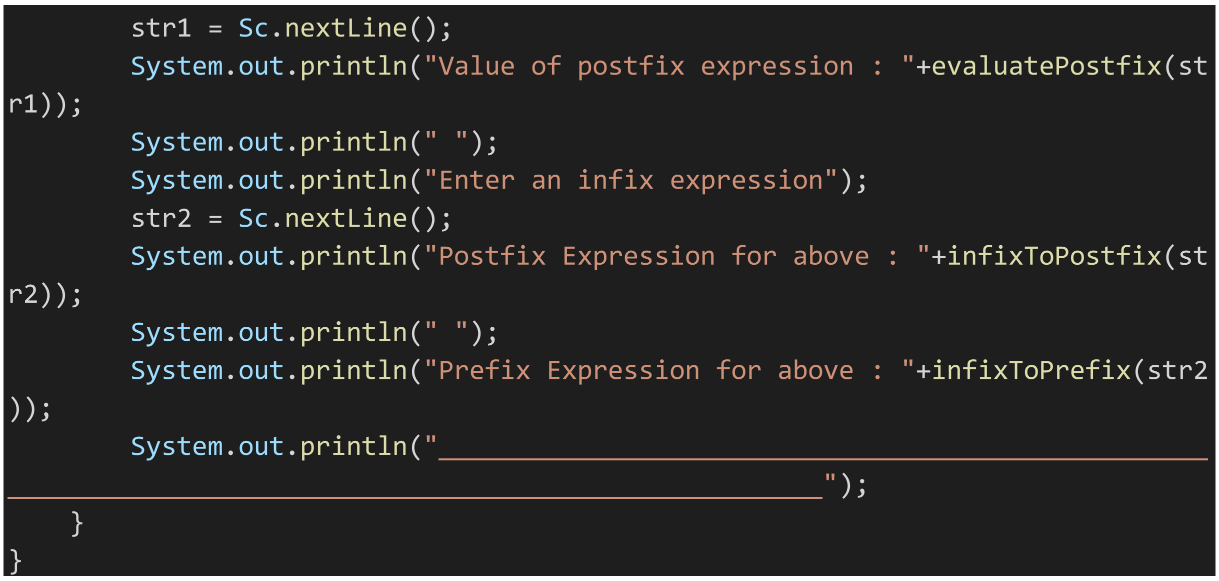
CODE

```
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 linke
d 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 = "+s
ize);
        }

        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 t
o 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 a
nd 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;
```

```java
        }
        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);
```

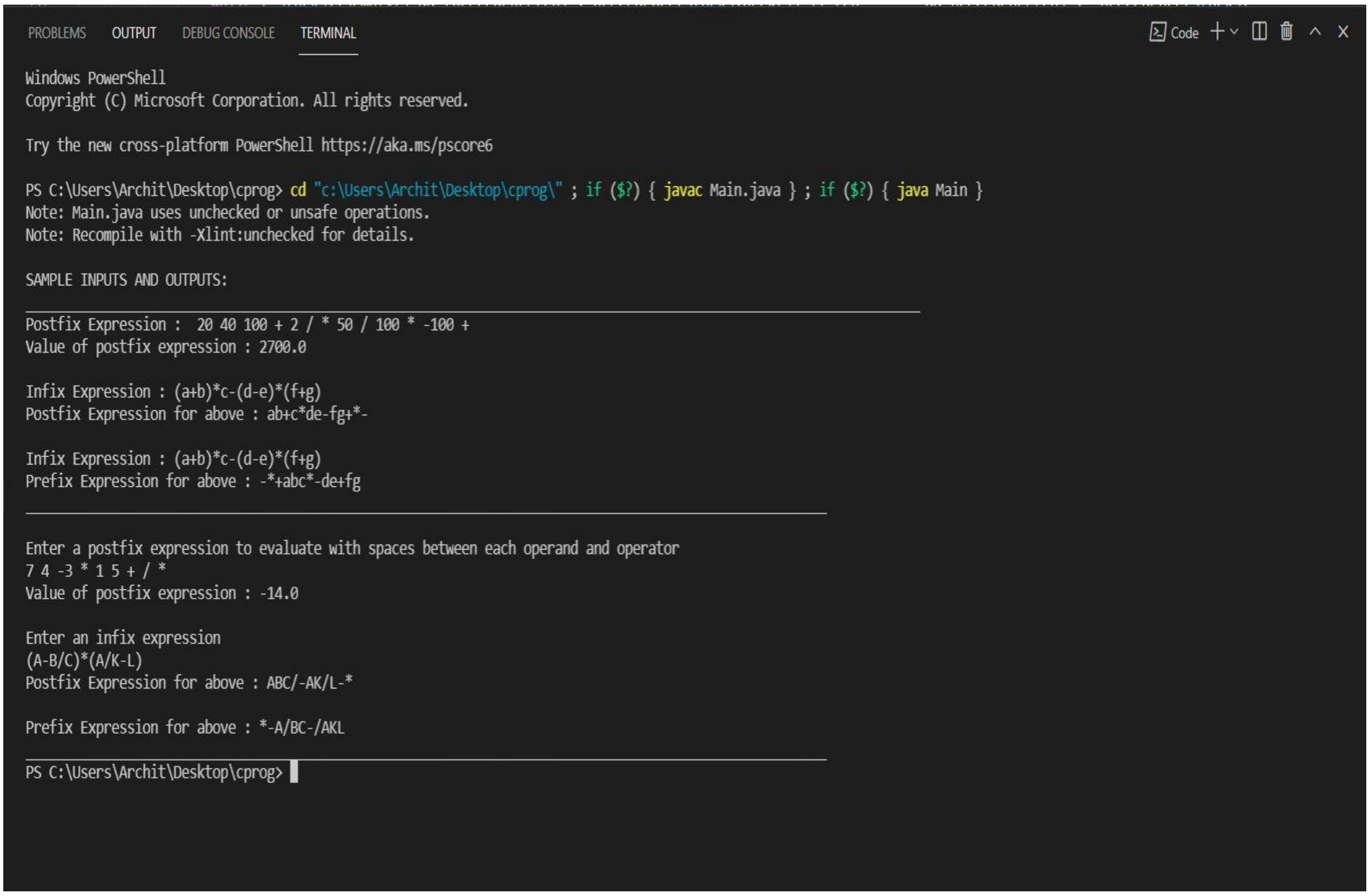
```java
            }
            else{
                double a = stack1.pop();
                double b = stack1.pop();
                switch(ch){
                    case '+':
                        stack1.push(b+a);
                        break;
                    case '-':
                        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(stack
2.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(stack
3.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 / 10
0 * -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");
```

ARCHIT AGRAWAL 202052307



```
        str1 = Sc.nextLine();
        System.out.println("Value of postfix expression : "+evaluatePostfix(st
r1));
        System.out.println(" ");
        System.out.println("Enter an infix expression");
        str2 = Sc.nextLine();
        System.out.println("Postfix Expression for above : "+infixToPostfix(st
r2));
        System.out.println(" ");
        System.out.println("Prefix Expression for above : "+infixToPrefix(str2
));
        System.out.println("__________________________________________________
_____________________________________________________");
    }
}
```

OUTPUT



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\Archit\Desktop\cprog> cd "c:\Users\Archit\Desktop\cprog\" ; if ($?) { javac Main.java } ; if ($?) { java Main }
Note: Main.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.

SAMPLE INPUTS AND OUTPUTS:
________________________________________________________________________
Postfix Expression :  20 40 100 + 2 / * 50 / 100 * -100 +
Value of postfix expression : 2700.0

Infix Expression : (a+b)*c-(d-e)*(f+g)
Postfix Expression for above : ab+c*de-fg+*-

Infix Expression : (a+b)*c-(d-e)*(f+g)
Prefix Expression for above : -*+abc*-de+fg
________________________________________________________________

Enter a postfix expression to evaluate with spaces between each operand and operator
7 4 -3 * 1 5 + / *
Value of postfix expression : -14.0

Enter an infix expression
(A-B/C)*(A/K-L)
Postfix Expression for above : ABC/-AK/L-*

Prefix Expression for above : *-A/BC-/AKL
________________________________________________________________
PS C:\Users\Archit\Desktop\cprog> ▌
```