PW SKILLS

Lesson Plan

Stack-4

PW SKILLS

Infix,Prefix,Postfix

Infix expressions are mathematical expressions where operators are placed between operands, and infix evaluation involves calculating the result of such expressions while considering operator precedence and parentheses.

Let's break down infix evaluation with and without brackets:

Infix Evaluation (Without Brackets):
For infix evaluation without brackets, you need to consider operator precedence to correctly evaluate the expression. You can use the algorithm known as the shunting-yard algorithm to convert infix expressions to postfix (or Reverse Polish Notation, RPN) and then evaluate the postfix expression.

Here's an example of infix evaluation without brackets using the shunting-yard algorithm:

Expression: `3 + 4 * 2 / (1 - 5)^2`

1. Convert infix to postfix: `3 4 2 * 1 5 - 2 ^ / +`
2. Evaluate the postfix expression using a stack-based algorithm or by iterating through the postfix expression.

Infix Evaluation (With Brackets):
When dealing with infix expressions that include brackets, you need to respect the precedence of operations within brackets and evaluate them first.

Expression: `(3 + 4) * 2 / (1 - 5)^2`

1. Start from left to right and identify the innermost brackets.
2. Evaluate expressions within the innermost brackets first.
3. Replace the evaluated expressions with their results in the original expression.
4. Repeat the process until there are no more brackets.

Steps:
- `(3 + 4) * 2 / (1 - 5)^2`
- `7 * 2 / (1 - 5)^2`
- `14 / (1 - 5)^2`
- `14 / 16`
- Result: `0.875`

Evaluating Expressions with Brackets Recursively:
The process involves recursively evaluating subexpressions within brackets until there are no more brackets in the expression.

This evaluation process maintains proper operator precedence by prioritizing operations within brackets, gradually simplifying the expression until the final result is obtained.

Prefix Notation (Polish Notation):
In prefix notation, the operator precedes its operands. For example, the infix expression (5 + 3) * 2 in prefix notation becomes * + 5 3 2.

Postfix Notation (Reverse Polish Notation):
In postfix notation, the operator follows its operands. Using the same infix expression (5 + 3) * 2, the postfix notation is 5 3 + 2 *.

Q. Given an infix expression, the task is to convert it to a prefix expression.
Input: A * B + C / D
Output: + * A B/ C D

Input: (A – B/C) * (A/K-L)
Output: *-A/BC-/AKL

Explanation: To convert an infix expression to a prefix expression, we can use the stack data structure. The idea is as follows:

Step 1: Reverse the infix expression. Note while reversing each '(' will become ')' and each ')' becomes '('.
Step 2: Convert the reversed infix expression to "nearly" postfix expression.
While converting to postfix expression, instead of using pop operation to pop operators with greater than or equal precedence, here we will only pop the operators from stack that have greater precedence.
Step 3: Reverse the postfix expression.
The stack is used to convert infix expression to postfix form.

Code:

```
// Java program to convert infix to prefix
import java.util.Stack;

class InfixToPrefixConverter {

    // Function to check if the character is an operator
    static boolean isOperator(char c) {
        return (!Character.isLetter(c) && !Character.isDigit(c));
    }

    // Function to get the priority of operators
    static int getPriority(char c) {
        if (c == '-' || c == '+') return 1;
        else if (c == '*' || c == '/') return 2;
        else if (c == '^') return 3;
        return 0;
    }

    // Function to convert the infix expression to postfix
    static String infixToPostfix(String infix) {
        infix = '(' + infix + ')';
        int l = infix.length();
        Stack<Character> charStack = new Stack<>();
        StringBuilder output = new StringBuilder();

        for (int i = 0; i < l; i++) {

            // If the scanned character is an operand, add it to output.
            if (Character.isLetter(infix.charAt(i)) || Character.isDigit(infix.charAt(i)))
                output.append(infix.charAt(i));
```

```
            // If the scanned character is an '(', push it to the stack.
            else if (infix.charAt(i) == '(')
                charStack.push('(');

            // If the scanned character is an ')', pop and output from the stack
            // until an '(' is encountered.
            else if (infix.charAt(i) == ')') {
                while (charStack.peek() != '(') {
                    output.append(charStack.pop());
                }

                // Remove '(' from the stack
                charStack.pop();
            }

            // Operator found
            else {
                if (isOperator(charStack.peek())) {
                    if (infix.charAt(i) == '^') {
                        while (getPriority(infix.charAt(i)) <= getPriority(charStack.peek()))
{
                            output.append(charStack.pop());
                        }
                    } else {
                        while (getPriority(infix.charAt(i)) < getPriority(charStack.peek()))
{
                            output.append(charStack.pop());
                        }
                    }

                    // Push current Operator on stack
                    charStack.push(infix.charAt(i));
                }
            }
        }
        while (!charStack.empty()) {
            output.append(charStack.pop());
        }
        return output.toString();
    }

    // Function to convert infix to prefix notation
    static String infixToPrefix(String infix) {
        // Reverse String and replace ( with ) and vice versa
        // Get Postfix
        // Reverse Postfix
        int l = infix.length();
```

```
        // Reverse infix
        StringBuilder reversedInfix = new StringBuilder(infix);
        reversedInfix.reverse();
        infix = reversedInfix.toString();

        // Replace ( with ) and vice versa
        for (int i = 0; i < l; i++) {
            if (infix.charAt(i) == '(') {
                reversedInfix.setCharAt(i, ')');
            } else if (infix.charAt(i) == ')') {
                reversedInfix.setCharAt(i, '(');
            }
        }

        String prefix = infixToPostfix(reversedInfix.toString());

        // Reverse postfix
        StringBuilder reversedPrefix = new StringBuilder(prefix);
        reversedPrefix.reverse();

        return reversedPrefix.toString();
    }

    // Driver code
    public static void main(String[] args) {
        String s = "x+y*z/w+u";

        // Function call
        System.out.println(infixToPrefix(s));
    }
}
```

Q. Write a program to convert an Infix expression to Postfix form.

Input: A + B * C + D
Output: ABC*+D+

Input: ((A + B) – C * (D / E)) + F
Output: AB+CDE/*-F+

Idea: To convert infix expression to postfix expression, use the stack data structure. Scan the infix expression from left to right. Whenever we get an operand, add it to the postfix expression and if we get an operator or parenthesis add it to the stack by maintaining their precedence.

Code:

```java
// Java code to convert infix expression to postfix

import java.util.Stack;

public class InfixToPostfixConverter {

    // Function to return precedence of operators
    static int prec(char c) {
        if (c == '^')
            return 3;
        else if (c == '/' || c == '*')
            return 2;
        else if (c == '+' || c == '-')
            return 1;
        else
            return -1;
    }

    // The main function to convert infix expression to postfix expression
    static void infixToPostfix(String s) {

        Stack<Character> st = new Stack<>();
        StringBuilder result = new StringBuilder();

        for (int i = 0; i < s.length(); i++) {
            char c = s.charAt(i);

            // If the scanned character is an operand, add it to the output string.
            if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z') || (c >= '0' && c <= '9'))
                result.append(c);

            // If the scanned character is an '(', push it to the stack.
            else if (c == '(')
                st.push('(');

            // If the scanned character is an ')',
            // pop and add to the output string from the stack
            // until an '(' is encountered.
            else if (c == ')') {
                while (st.peek() != '(') {
                    result.append(st.pop());
                }
                st.pop();
            }

            // If an operator is scanned
            else {
                while (!st.empty() && prec(c) <= prec(st.peek())) {
                    result.append(st.pop());
```

```
                }
                st.push(c);
            }
        }

        // Pop all the remaining elements from the stack
        while (!st.empty()) {
            result.append(st.pop());
        }

        System.out.println(result.toString());
    }

    // Driver code
    public static void main(String[] args) {
        String exp = "a+b*(c^d-e)^(f+g*h)-i";

        // Function call
        infixToPostfix(exp);
    }
}
```

Q. Given a postfix expression, the task is to evaluate the postfix expression.

Input: str = "2 3 1 * + 9 -"
Output: -4

Explanation: If the expression is converted into an infix expression, it will be 2 + (3 * 1) – 9 = 5 – 9 = -4.

Input: str = "100 200 + 2 / 5 * 7 +"
Output: 757

Idea: Iterate the expression from left to right and keep on storing the operands into a stack. Once an operator is received, pop the two topmost elements and evaluate them and push the result in the stack again.

Code:

```
import java.util.Stack;

public class PostfixEvaluator {

    // Function to evaluate a postfix expression
    static int evaluatePostfix(String exp) {
        // Create a stack to store operands
        Stack<Integer> st = new Stack<>();

        // Scan all characters one by one
        for (int i = 0; i < exp.length(); ++i) {
```

```
            // If the scanned character is an operand (number here),
            // push it to the stack.
            if (Character.isDigit(exp.charAt(i)))
                st.push(exp.charAt(i) - '0');

            // If the scanned character is an operator,
            // pop two elements from the stack and apply the operator.
            else {
                int val1 = st.pop();
                int val2 = st.pop();
                switch (exp.charAt(i)) {
                    case '+':
                        st.push(val2 + val1);
                        break;
                    case '-':
                        st.push(val2 - val1);
                        break;
                    case '*':
                        st.push(val2 * val1);
                        break;
                    case '/':
                        st.push(val2 / val1);
                        break;
                }
            }
        }
        // The final result will be on the top of the stack.
        return st.pop();
    }

    // Driver code
    public static void main(String[] args) {
        String exp = "231*+9-";
        System.out.println("Result: " + evaluatePostfix(exp));
    }
}
```

Q. Prefix Evaluation

Input: -+8/632
Output: 8

Input: -+7*45+20
Output: 25

Algorithm: EVALUATE_PREFIX(STRING)

Step 1: Put a pointer P at the end of the end
Step 2: If character at P is an operand push it to Stack

Step 3: If the character at P is an operator pop two elements from the Stack. Operate on these elements according to the operator, and push the result back to the Stack

Step 4: Decrement P by 1 and go to Step 2 as long as there are characters left to be scanned in the expression.

Step 5: The Result is stored at the top of the Stack, return it

Step 6: End

Code:

```
import java.util.Stack;

public class PrefixEvaluator {

    // Function to evaluate a prefix expression
    static double evaluatePrefix(String exprsn) {
        Stack<Double> stack = new Stack<>();

        for (int j = exprsn.length() - 1; j >= 0; j--) {
            // If jth character is the delimiter (which is space in this case), then skip it
            if (exprsn.charAt(j) == ' ')
                continue;

            // Push operand to stack
            // To convert exprsn[j] to digit, subtract '0' from exprsn[j]
            if (Character.isDigit(exprsn.charAt(j))) {
                // There may be more than one digit in a number
                double num = 0;
                int i = j;
                while (j >= 0 && Character.isDigit(exprsn.charAt(j)))
                    j--;

                j++;
                // From [j, i] exprsn contains a number
                for (int k = j; k <= i; k++)
                    num = num * 10 + Double.parseDouble(String.valueOf(exprsn.charAt(k)));

                stack.push(num);
            } else {
                // Operator encountered
                // Pop two elements from stack
                double o1 = stack.pop();
                double o2 = stack.pop();

                // Use switch case to operate on o1 and o2 and perform o1 O o2
                switch (exprsn.charAt(j)) {
                    case '+':
```

```
                    stack.push(o1 + o2);
                    break;
                case '-':
                    stack.push(o1 - o2);
                    break;
                case '*':
                    stack.push(o1 * o2);
                    break;
                case '/':
                    stack.push(o1 / o2);
                    break;
            }
        }
    }
    return stack.pop();
}

// Driver code
public static void main(String[] args) {
    String exprsn = "+ 5 * 4 7";
    System.out.println("Result: " + evaluatePrefix(exprsn));
}
}
```

Q. Given a Prefix expression, convert it into a Infix expression.

Input: Prefix : *+AB-CD
Output: Infix : ((A+B)*(C-D))

Input: Prefix : *-A/BC-/AKL
Output: Infix : ((A-(B/C))*((A/K)-L))

Algorithm for Prefix to Infix: Read the Prefix expression in reverse order (from right to left)
If the symbol is an operand, then push it onto the Stack
If the symbol is an operator, then pop two operands from the Stack
Create a string by concatenating the two operands and the operator between them.
string = (operand1 + operator + operand2)
And push the resultant string back to Stack
Repeat the above steps until the end of Prefix expression.
At the end stack will have only 1 string i.e resultant string

Code:

```
// Java Program to convert prefix to Infix
import java.util.Stack;
```

```
public class PrefixToInfixConverter {

    // Function to check if character is an operator or not
    static boolean isOperator(char x) {
        switch (x) {
            case '+':
            case '-':
            case '/':
            case '*':
            case '^':
            case '%':
                return true;
        }
        return false;
    }

    // Convert prefix to infix expression
    static String preToInfix(String preExp) {
        Stack<String> stack = new Stack<>();

        // Length of expression
        int length = preExp.length();

        // Reading from right to left
        for (int i = length - 1; i >= 0; i--) {

            // Check if the symbol is an operator
            if (isOperator(preExp.charAt(i))) {

                // Pop two operands from the stack
                String op1 = stack.pop();
                String op2 = stack.pop();

                // Concatenate the operands and operator
                String temp = "(" + op1 + preExp.charAt(i) + op2 + ")";

                // Push string temp back to the stack
                stack.push(temp);
            }

            // If the symbol is an operand
            else {

                // Push the operand to the stack
                stack.push(String.valueOf(preExp.charAt(i)));
            }
```

```
        }

        // Stack now contains the infix expression
        return stack.pop();
    }

    // Driver Code
    public static void main(String[] args) {
        String preExp = "*-A/BC-/AKL";
        System.out.println("Infix: " + preToInfix(preExp));
    }
}
```

Q. Given a Prefix expression, convert it into a Postfix expression.

Input: Prefix : *+AB-CD
Output: Postfix : AB+CD-*

Explanation: Prefix to Infix : (A+B) * (C-D)
Infix to Postfix : AB+CD-*

Input: Prefix : *-A/BC-/AKL
Output: Postfix : ABC/-AK/L-*

Explanation: Prefix to Infix : (A-(B/C))*((A/K)-L)
Infix to Postfix : ABC/-AK/L-*

Algorithm: Read the Prefix expression in reverse order (from right to left)
If the symbol is an operand, then push it onto the Stack
If the symbol is an operator, then pop two operands from the Stack
Create a string by concatenating the two operands and the operator after them.
string = operand1 + operand2 + operator
And push the resultant string back to Stack
Repeat the above steps until end of Prefix expression.

Code:

```
// Java Program to convert prefix to postfix
import java.util.Stack;

public class PrefixToPostfixConverter {

    // Function to check if character is an operator or not
    static boolean isOperator(char x) {
        switch (x) {
            case '+':
            case '-':
            case '/':
```

```
            case '*':
                return true;
        }
        return false;
    }

    // Convert prefix to postfix expression
    static String preToPost(String preExp) {
        Stack<String> stack = new Stack<>();

        // Length of expression
        int length = preExp.length();

        // Reading from right to left
        for (int i = length - 1; i >= 0; i--) {

            // Check if the symbol is an operator
            if (isOperator(preExp.charAt(i))) {

                // Pop two operands from the stack
                String op1 = stack.pop();
                String op2 = stack.pop();

                // Concatenate the operands and operator
                String temp = op1 + op2 + preExp.charAt(i);

                // Push string temp back to the stack
                stack.push(temp);
            }

            // If the symbol is an operand
            else {

                // Push the operand to the stack
                stack.push(String.valueOf(preExp.charAt(i)));
            }
        }

        // Stack now contains only the postfix expression
        return stack.pop();
    }

    // Driver Code
    public static void main(String[] args) {
        String preExp = "*-A/BC-/AKL";
        System.out.println("Postfix: " + preToPost(preExp));
    }
}
```

Q. Postfix to Infix

Input: abJava
Output: (a + (b + c))

Input: ab*c+
Output: ((a*b)+c)

Algorithm:

1. While there are input symbol left
1.1 Read the next symbol from the input.
2. If the symbol is an operand
2.1 Push it onto the stack.
3. Otherwise,
3.1 the symbol is an operator.
3.2 Pop the top 2 values from the stack.
3.3 Put the operator, with the values as arguments and form a string.
3.4 Push the resulted string back to stack.
4. If there is only one value in the stack
4.1 That value in the stack is the desired infix string.

Below is the implementation of above approach:

Code:

```
// Java program to find infix for
// a given postfix.
import java.util.Stack;

public class PostfixToInfixConverter {

    static boolean isOperand(char x) {
        return (x ≥ 'a' && x ≤ 'z') || (x ≥ 'A' && x ≤ 'Z');
    }

    // Get Infix for a given postfix expression
    static String getInfix(String exp) {
        Stack<String> stack = new Stack<>();

        for (int i = 0; i < exp.length(); i++) {
            // Push operands
            if (isOperand(exp.charAt(i))) {
                stack.push(String.valueOf(exp.charAt(i)));
            } else {
                // We assume that input is a valid postfix and expect an operator.
                String op1 = stack.pop();
                String op2 = stack.pop();
                stack.push("(" + op2 + exp.charAt(i) + op1 + ")");
```

```
            }
        }

        // There must be a single element in stack now which is the required infix.
        return stack.pop();
    }

    // Driver code
    public static void main(String[] args) {
        String exp = "ab*c+";
        System.out.println(getInfix(exp));
    }
}
```

Q. Postfix to Prefix Conversion

Input: Postfix : AB+CD-*
Output: Prefix : *+AB-CD
Explanation: Postfix to Infix : (A+B) * (C-D)
Infix to Prefix : *+AB-CD

Input: Postfix : ABC/-AK/L-*
Output: Prefix : *-A/BC-/AKL
Explanation: Postfix to Infix : ((A-(B/C))*((A/K)-L))
Infix to Prefix : *-A/BC-/AKL

Algorithm for Postfix to Prefix:

Read the Postfix expression from left to right
If the symbol is an operand, then push it onto the Stack
If the symbol is an operator, then pop two operands from the Stack
Create a string by concatenating the two operands and the operator before them.
string = operator + operand2 + operand1
And push the resultant string back to Stack
Repeat the above steps until end of Postfix expression.

Code:

```
// Java Program to convert postfix to prefix
import java.util.Stack;

public class PostfixToPrefixConverter {

    // Function to check if character is an operator or not
    static boolean isOperator(char x) {
        switch (x) {
            case '+':
            case '-':
```

```
            case '/':
            case '*':
                return true;
        }
        return false;
    }

    // Convert postfix to prefix expression
    static String postToPre(String postExp) {
        Stack<String> stack = new Stack<>();

        // Length of expression
        int length = postExp.length();

        // Reading from left to right
        for (int i = 0; i < length; i++) {
            // Check if symbol is an operator
            if (isOperator(postExp.charAt(i))) {
                // Pop two operands from stack
                String op1 = stack.pop();
                String op2 = stack.pop();

                // Concatenate the operands and operator
                String temp = postExp.charAt(i) + op2 + op1;

                // Push string temp back to stack
                stack.push(temp);
            } else {
                // If symbol is an operand, push it to the stack
                stack.push(String.valueOf(postExp.charAt(i)));
            }
        }

        // The stack now contains the prefix expression
        StringBuilder ans = new StringBuilder();
        while (!stack.empty()) {
            ans.insert(0, stack.pop());
        }
        return ans.toString();
    }

    // Driver Code
    public static void main(String[] args) {
        String postExp = "ABC/-AK/L-*";

        // Function call
        System.out.println("Prefix: " + postToPre(postExp));
    }
}
```

PW SKILLS

THANK YOU !

PW SKILLS