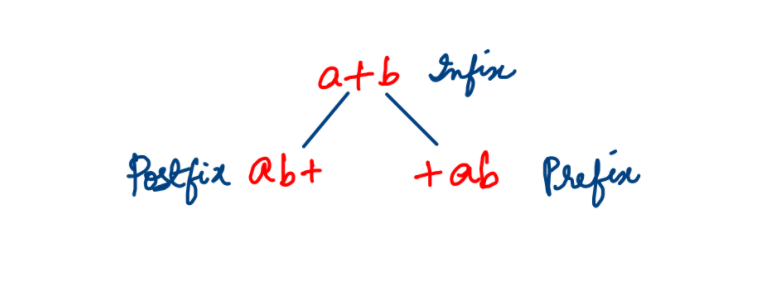
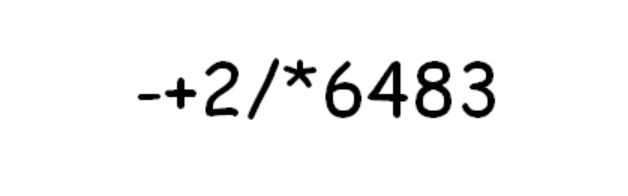
**1. Problem Discussion**

The problem states that: 1. You will be given a prefix expression and you will be required to evaluate and print its value. 2. You are required to convert it to infix and print it. 3. You are required to convert it to postfix and print it. You also need to understand what prefix, infix and postfix are for a given expression. Have a look at figure below to understand this.

****

Let's understand this question through the following example:

****

**2. Approach**

We form an algorithm for this problem. Look at the discussion given below. For prefix expressions, evaluation is done from the last element to the first element. We create 3 stacks : "P" for Postfix Stack, "I" for Infix Stack and "V" for Value Stack. Let's start the process for all the elements one by one. We have 2 cases: either we will get an operand or an operator. Let's look at both.

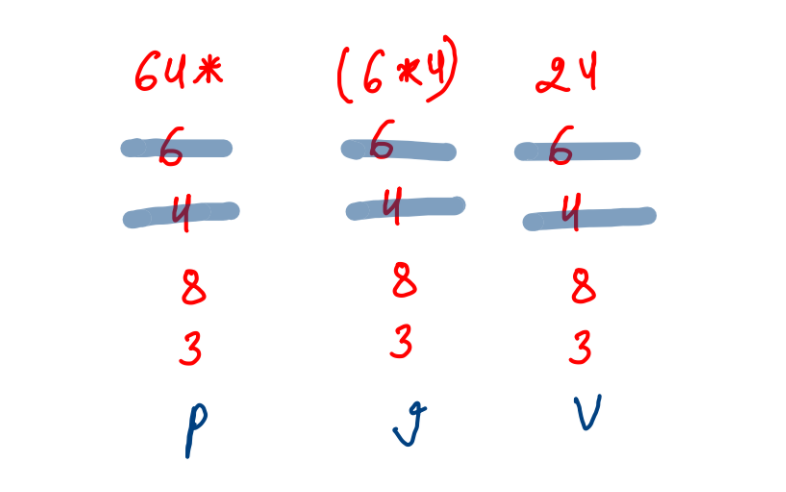
OPERAND-

If we get an operand we simply push it into all 3 stacks. Starting from the last element, we get an operand 3 so we push it in all the three stacks. Similarly, 8, 4 and 6 are operands so they are also pushed in all the stacks.

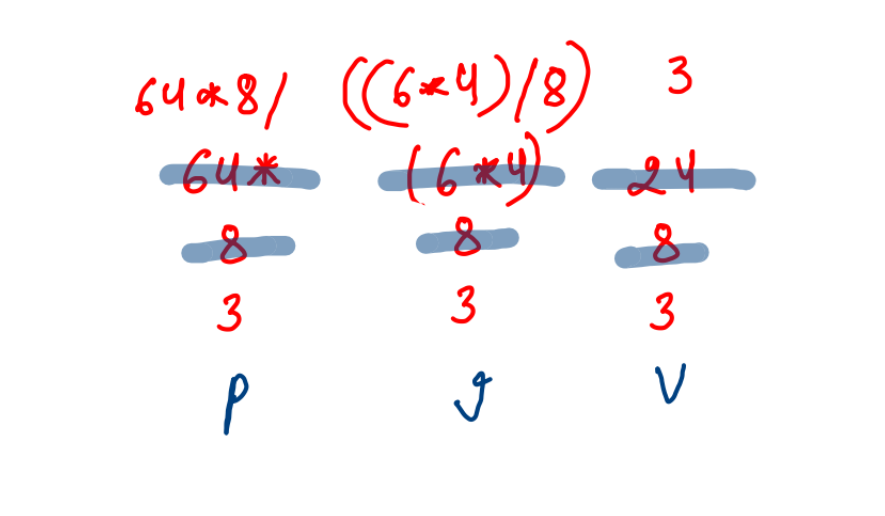
OPERATOR-

In this question, we always consider the topmost value as the first operand and the second top value as the second operand. Now we get an operator "\*".

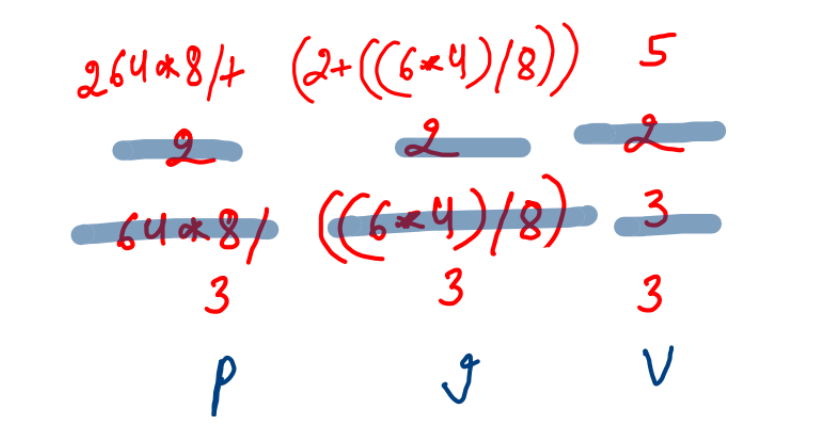
Value Stack- For this, we pop the top 2 elements from "V" and push the value obtained by computing those values with the given operator. In this case, we pop 6 and 4 and then push 6\*4=24 into the same stack "V". Infix Stack- In this stack too we pop the top 2 elements and return an infix expression for those 2 values and the operator i.e. (6\*4) in this case. Postfix Stack - For this stack we pop the top 2 values and push the postfix expression for those values and the operator i.e. 64\* in this case.

****

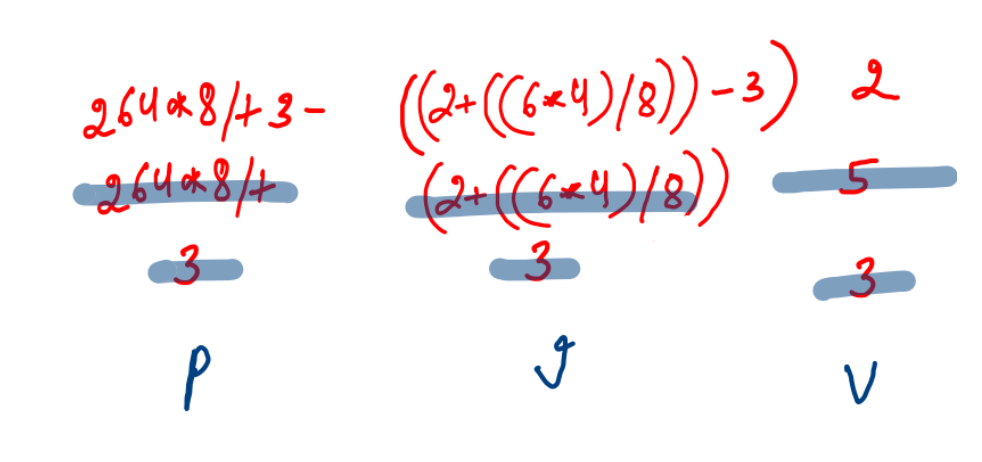
Similarly, as we move along the expression, we get another operator '/'. Again the same process is followed.

****

Next we get an operand '2' which is pushed in all the stacks. After that we obtain an operator '+' which gives the following result:

****

Our last element is '-'. Could you do the stacking process for it yourself?

****

Now one element is left in each stack which is our answer. We are now finished with our algorithm. Let's code it ! Since you already know how to write the code of the previous questions of Infix and Postfix, we are confident that you will be able to crack the code for this problem. If you are not able to just refer to the code given below and check out its explanation.

**3. Code**

ConsoleJava

import java.io.\*;

import java.util.\*;

public class Main {

public static void main(String[] args) throws Exception {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

String exp = br.readLine();

Stack< Integer> vs = new Stack< >(); //1

Stack< String> is = new Stack< >();

Stack< String> ps = new Stack< >();

for (int i = exp.length() - 1; i >= 0; i--) {

char ch = exp.charAt(i); //2

if (ch == '+' || ch == '-' || ch == '\*' || ch == '/') { //3

int v1 = vs.pop();

int v2 = vs.pop();

int val = operation(v1, v2, ch);

vs.push(val);

String iv1 = is.pop(); //4

String iv2 = is.pop();

String ival = "(" + iv1 + ch + iv2 + ")";

is.push(ival);

String pv1 = ps.pop(); //5

String pv2 = ps.pop();

String pval = pv1 + pv2 + ch;

ps.push(pval);

} else {

vs.push(ch - '0'); //6

is.push(ch + "");

ps.push(ch + "");

}

}

System.out.println(vs.pop()); //7

System.out.println(is.pop());

System.out.println(ps.pop());

}

public static int operation (int v1, int v2, char op) { //8

if (op == '+') {

return v1 + v2;

} else if (op == '-') {

return v1 - v2;

} else if (op == '\*') {

return v1 \* v2;

} else {

return v1 / v2;

}

}

}

**4. Code Discussion**

1• First we make 3 stacks: vs, is and ps of type Integer, String and String respectively. 2• For every character in the expression starting from the last element, we check whether it is an operand or an operator. 3• If it is an operator, then we compute the 3 stacks . 4• First for "vs", the topmost value is v1 and second topmost value is v2. They are arithmetically computed by calling the "operation" function and the calculated value is pushed into vs. 5• Next for "is", the topmost string and second top most string are written in the form of an infix expression and pushed into the stack. 6• Lastly for "ps", the top 2 strings are pushed into the stack as a postfix expression. 7• Else if the character is an operand then the character is changed into integer and pushed into "vs" and as a string pushed into both "is" and "ps". 8• As seen in the algorithm, when all the elements are used up then there is one value left in each stack which is our final answer. Hence they all are popped and printed. 9• Function "operation" is used in //3 and helps us in calculating the arithmetic answer for the given operator and operands.

**5. Analysis**

Time Complexity : O(n)

This is because a single for loop is used in the program and the rest of the functions of stack use O(1) time.

Space Complexity : O(n)

We use a stack of space n to run the program hence space complexity is of order n.