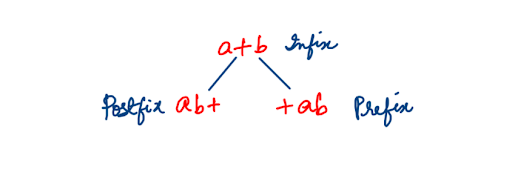
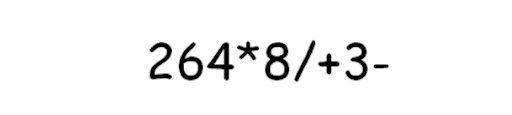
**1. Problem Discussion :**

You are given a postfix expression and you are required to evaluate and print its value. You are required to convert it to infix and print it. You are required to convert it to prefix and print it. You also need to understand what prefix, infix and postfix are for a given expression. Have a look at figure 1 to understand this.

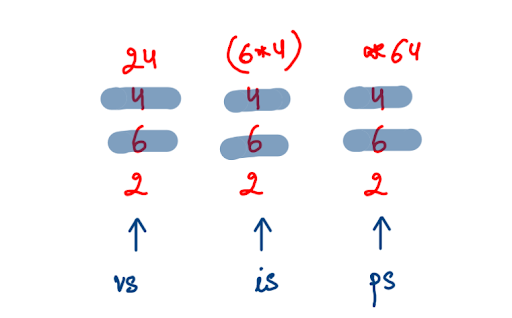
****

Let's understand it through the example given in the question.

****

**2. Approach :**

We form an algorithm for this problem. Look at the discussion for Figure 3 below. First, we solve it till 264\* .

****

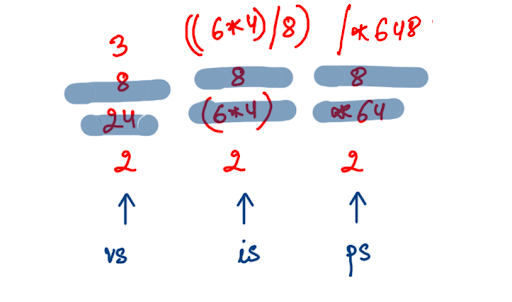
We create 3 stacks : "vs" for Value Stack, "is" for Infix Stack and "ps" for Prefix Stack. Let's start the process of pushing for all the elements of the input one by one. We have 2 cases: either we will get an operand or an operator. Let's look at both.

OPERAND-

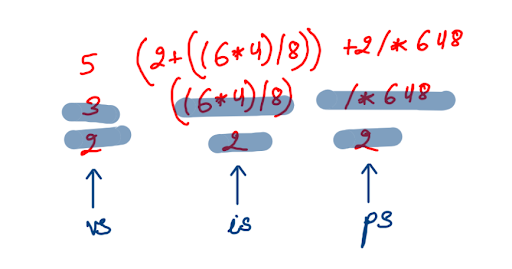
First we get an operand 2 so we push it in all the three stacks. Again, since 6 and 4 are operands they are pushed in all the stacks.

OPERATOR-

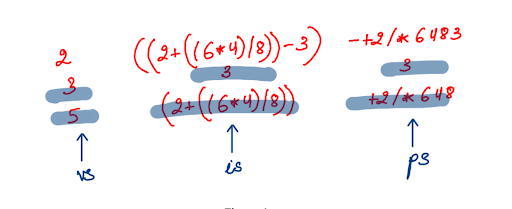
Now we get an operator "\*". Value Stack- For this, we pop the top 2 elements from "vs" and push the value obtained by computing those values with the given operator in the stack. In this case, we pop 4 and 6 and then push 6\*4=24 into the same stack "vs". Infix Stack- In this stack too we pop the top 2 elements and return an infix expression for those 2 values and the operator i.e. (6\*4) in this case. Prefix Stack - For this stack we pop the top 2 values and push the prefix expression for those values and the operator i.e. \*64 in this case. In the cases of the Infix and Prefix Conversions we always consider the topmost value as the second operand and the second top value as the first operand. You might not see a difference if the positions of operands are changed in case of addition or multiplication. However their position matters for cases of subtraction and division. Let's see the stacking processes for the rest of the elements now. We move on to the next element 8 in the postfix expression. As it's an operand it is pushed into all three stacks. The next element is an operator "/". For this we repeat the process above (see figure 4).

****

Now we again get an operator '+' as the next element. So the stacks would like this:

****

Last elements are 3 and '-'. 3 will be pushed in all the three stacks and appropriate computation is done for the operator, '- '.

****

We are now finished with our algorithm. Now that you are aware of how the algorithm works, try to code it yourself. We'll be with you every step of the way. This code might appear a bit lengthy but trust us, it's very easy and just based on the algorithm.

**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 = 0; i < exp.length(); i++) {

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

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

int v2 = vs.pop();

int v1 = vs.pop();

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

vs.push(val);

String iv2 = is.pop(); //4

String iv1 = is.pop();

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

is.push(ival);

String pv2 = ps.pop(); //5

String pv1 = ps.pop();

String pval = ch + pv1 + pv2;

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, we check whether it is an operand or an operator. 3• If it is an operator, then we compute the 3 stacks . First for "vs", the topmost value is v2 and second topmost value is v1.They are arithmetically computed by calling the "operation" function and the calculated value is pushed into vs. 4•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. 5• Lastly for "ps", the top 2 strings are pushed into the stack as a prefix expression. 6• 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". 7• 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. 8•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. We suggest you check out the solution video for a detailed explanation.