Infix Conversions

Easy

Prev Next

1. You are given an infix expression.
2. You are required to convert it to postfix and print it.
3. You are required to convert it to prefix and print it.

Constraints

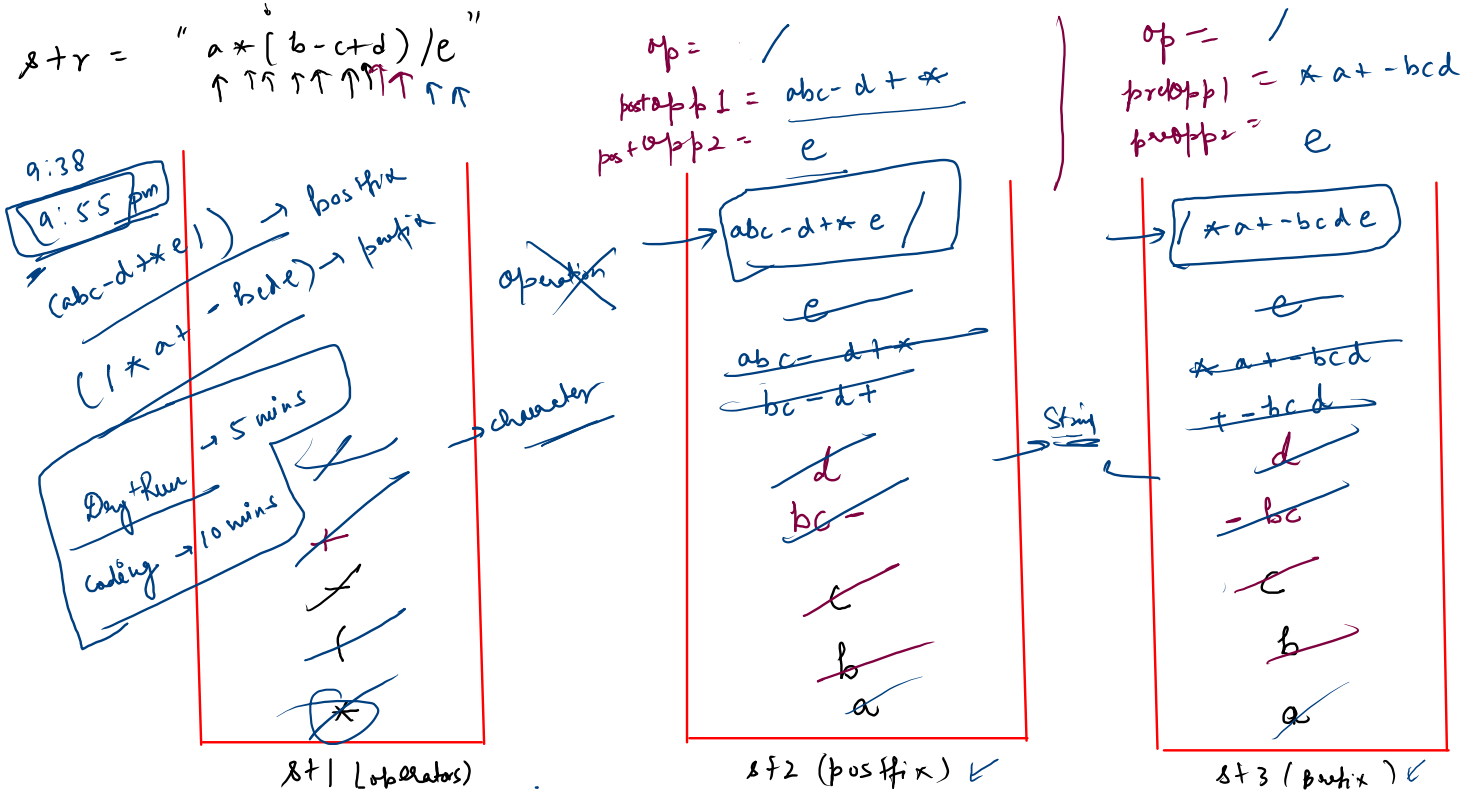
1. Expression is balanced
2. The only operators used are +, -, *, /
3. Opening and closing brackets - () - are used to impact precedence of operations
4. + and - have equal precedence which is less than * and /. * and / also have equal precedence.
5. In two operators of equal precedence give preference to the one on left.
6. All operands are single digit numbers.

Sample Input

a*(b-c+d)/e

postfix → ab+

prefix → +ab



1

```
public static int precedence(char ch){
    if (ch == '+' || ch == '-') return 1;
    else if (ch == '*' || ch == '/') return 2;
    else return 0;
}
```

5 min.

2

```
Scanner scn = new Scanner(System.in);
String str = scn.nextLine();

Stack<Character> st1 = new Stack<>(); // operators and opening bracket
Stack<String> st2 = new Stack<>(); // postfix exp
Stack<String> st3 = new Stack<>(); // prefix exp
```

4

```
while(st1.size() > 0 ){
     char op = st1.pop();
    String postfixopp2 = st2.pop();
    String postfixopp1 = st2.pop();

    st2.push(postfixopp1 + postfixopp2 + op);

    String prefixopp2 = st3.pop();
    String prefixopp1 = st3.pop();

    st3.push(op + prefixopp1 + prefixopp2);
}

System.out.println(st2.pop());
System.out.println(st3.pop());
```

3

```
for(int i=0;i<str.length();i++){
    char ch = str.charAt(i);

    if (ch == '('){
        st1.push(ch);
    } else if (ch == ')') {
        while(st1.peek() != '('){
            char op = st1.pop();
            String postfixopp2 = st2.pop();
            String postfixopp1 = st2.pop();

            st2.push(postfixopp1 + postfixopp2 + op);

            String prefixopp2 = st3.pop();
            String prefixopp1 = st3.pop();

            st3.push(op + prefixopp1 + prefixopp2);
        }
        st1.pop(); // poping opening bracket

    } else if (ch == '+' || ch == '-' || ch == '*' || ch == '/'){
        while(st1.size() >0 && precedence(ch) <= precedence(st1.peek())){
             char op = st1.pop();
            String postfixopp2 = st2.pop();
            String postfixopp1 = st2.pop();

            st2.push(postfixopp1 + postfixopp2 + op);

            String prefixopp2 = st3.pop();
            String prefixopp1 = st3.pop();

            st3.push(op + prefixopp1 + prefixopp2);
        }
        st1.push(ch);
    } else if (ch >='a' && ch <= 'z'){
        st2.push(ch + "");
        st3.push(ch + "");
    }
}
```

1. You are given a postfix expression.
2. You are required to evaluate it and print it's value.
3. You are required to convert it to infix and print it.
4. You are required to convert it to prefix and print it.

Note -> Use brackets in infix expression for indicating precedence. Check sample input output for more details.

Str = "264*8/+3-"

Constraints

1. Expression is a valid postfix expression
2. The only operators used are +, -, *, /
3. All operands are single digit numbers.

Sample Input

264*8/+3-

Sample Output

2
((2+((6*4)/8))-3)
-+2/*6483

str = " 2 6 4 * 8 / + 3 - "

5 - 3 = 2

⇒ validate

op = -

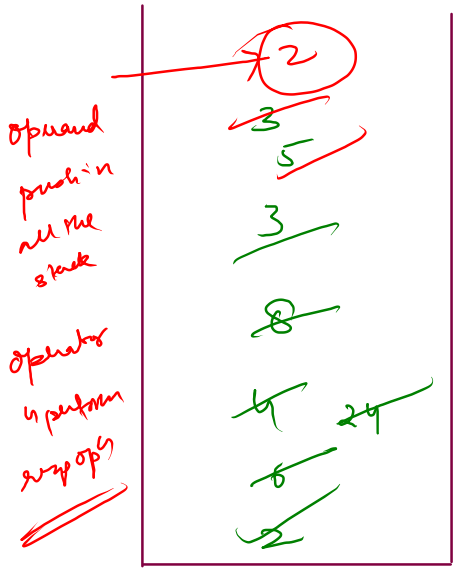
op2 = 3

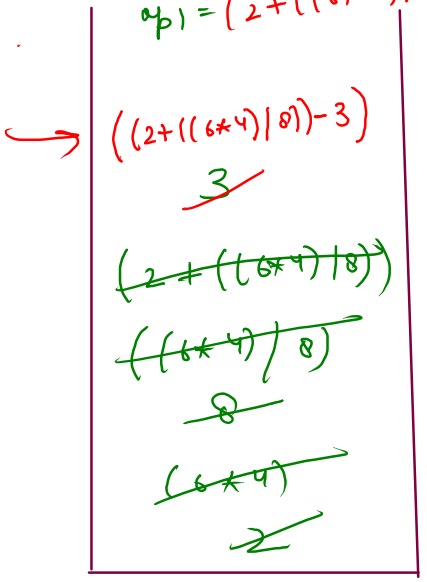
op1 = (2+((6*4)/8))

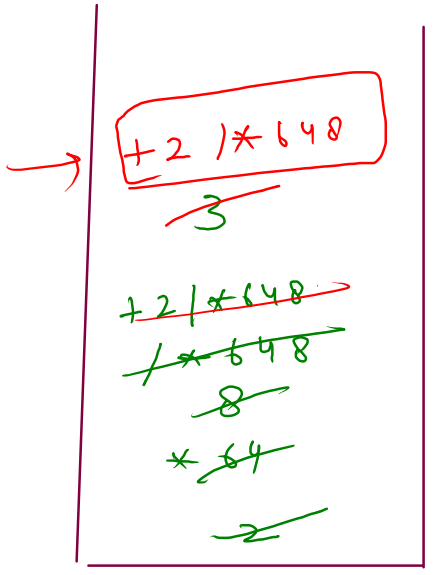
op = +

op1 = 2

op2 = /*648





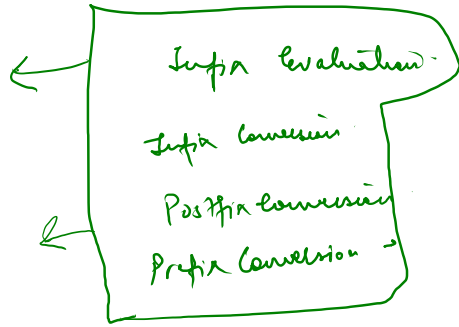


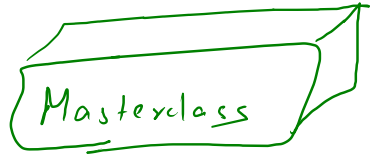
Integer → st1 (ans)

String st2 (Infix)

String → st3 (prefix)

```java
public class Solution {
    public static int operation(int a, int b, char ch){
        if (ch == '+') return a+b;
        else if (ch == '-') return a-b;
        else if (ch == '*') return a*b;
        else return a/b;
    }

    public static void main(String[] args) {
        /* Enter your code here. Read input from STDIN. Print output to STDOUT. Your class should be r
        Scanner scn = new Scanner(System.in);
        String str = scn.nextLine();

        Stack<Integer> st1 = new Stack<>();
        Stack<String> st2 = new Stack<>(); // For Infix
        Stack<String> st3 = new Stack<>(); // For Prefix

        for(int i=0;i<str.length();i++){
            char ch = str.charAt(i);

            if (ch >= '0' && ch <='9'){
                st1.push(ch - '0');
                st2.push(ch + "");
                st3.push(ch + "");
            } else {
                int b = st1.pop();
                int a = st1.pop();
                st1.push(operation(a,b,ch));

                String op2 = st2.pop();
                String op1 = st2.pop();

                st2.push('(' + op1 + ch + op2 +')');

                op2 = st3.pop();
                op1 = st3.pop();

                st3.push(ch + op1 + op2);
            }
        }

        System.out.println(st1.pop());
        System.out.println(st2.pop());
        System.out.println(st3.pop());
    }
}
```

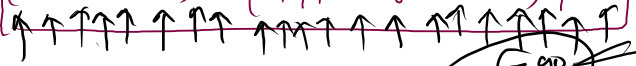


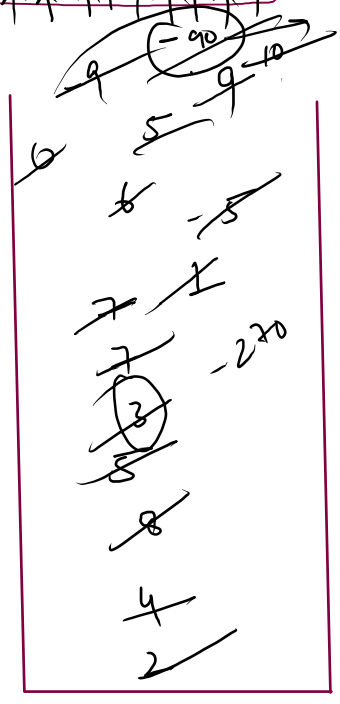


→ Function never affect the time complexity.

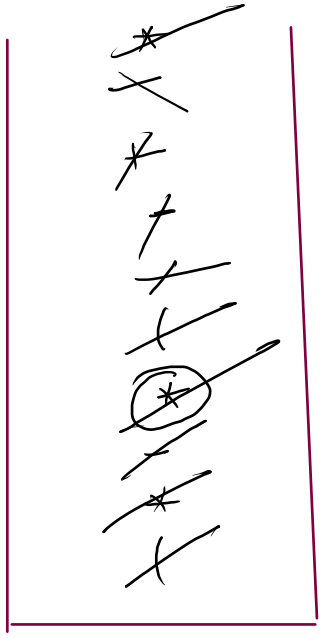
→ Modularise code.

(2*4-5)*(7|7-6+5-9)*10





st1 (operands)



st2 (operators)

Ans = -90

-9 * 10 = -90

3* -90 = -270

= -270

2*4=8

8-5=3

7|7=1

1-6=-5

5-5=0

0-9=-9

2-3 digram +

10 minus

12 mins

7-8 cooking