

```

1 String ListtoString(LinkedList<String> list){
2     StringBuilder sb = new StringBuilder();
3     for (String L : list) {
4         sb.append(L).append(" ");
5     }
6     return sb.toString();
7 }
8
9 int precedence(String operator){
10     if(operator.equals("*") || operator.equals("/")) return 1;
11     else return 0;
12 }
13
14 boolean isNumeric(String s) {
15     try {
16         Double.parseDouble(s);
17     }catch (NumberFormatException e) {
18         return false;
19     }
20     return true;
21 }
22
23 String getINFIX(){
24     return infix;
25 }
26 String getPREFIX(){
27     return prefix;
28 }
29 String getPOSTFIX(){
30     return postfix;
31 }
32 //Tree
33 void getTREE(){
34     printTree(constructExpressionTree(postfixlist),0);
35 }
36
37 Node constructExpressionTree(LinkedList<String> postfix){ //5 2 2 / + 6
38     LinkedList<Node> list = new LinkedList<>();
39     for (String s : postfix){
40         Node node = new Node(s);
41         if(isNumeric(s)){
42             list.add(node);
43         }else{
44             Node right = list.removeLast();
45             Node left = list.removeLast();
46             node.right = right;
47             node.left = left;
48             list.add(node);
49         }
50     }
51     return list.removeLast();
52 }
53
54 void printTree(Node tree, int space){
55     if(tree == null)
56         return;
57     space += 5;
58     printTree(tree.right, space);
59
60     System.out.println();
61     for(int i = 5; i < space; i++)
62         System.out.print(" ");
63     System.out.println(tree.val);
64
65     printTree(tree.left, space);
66 }
67
68 }

```

```

1 import java.util.LinkedList;
2
3 public class Main{
4     public static void main(String[] args) {
5         String input = "5 + 2 / 2 - 6";
6         equation q1 = new equation(input);
7         System.out.println("Infix: " + q1.getInFIX()); //Infix: 5 + 2 / 2 - 6
8         System.out.println("Prefix: " + q1.getPREFIX()); //Prefix: - + 5 / 2 2 6
9         System.out.println("Postfix: " + q1.getPOSTFIX()); //Postfix: 5 2 2 / + 6 -
10        q1.getTREE();
11    }
12 }
13
14 class equation{
15     private String infix;
16     private String postfix;
17     private String prefix;
18     private LinkedList<String> infixlist;
19     private LinkedList<String> postfixlist;
20     private LinkedList<String> prefixlist;
21
22     equation(String ip){
23         infix = ip;
24         addinfixinlist(infix);
25         postfixlist = INTOPOST(infixlist);
26         postfix = ListtoString(postfixlist);
27         prefixlist = INTOPRE(infixlist);
28         prefix = ListtoString(prefixlist);
29     }
30
31     void addinfixinlist(String ip){
32         infixlist = new LinkedList<>();
33         String[] arrip = ip.split(" ");
34         for(String arr : arrip)infixlist.add(arr);
35     }
36
37     LinkedList<String> INTOPOST(LinkedList<String> infix){
38         LinkedList<String> postfix = new LinkedList<>();
39         LinkedList<String> list = new LinkedList<>();
40         for (int i=0; i<infix.size(); i++){ //5 + 2 / 2 - 6
41             String s = infix.get(i);
42             if(isNumeric(s)){
43                 postfix.add(s);
44             }else{
45                 while (!list.isEmpty() && precedence(s)<precedence(list.peek())) { // peek()->0
46                     postfix.add(list.pop()); //pop()->0
47                 }
48                 list.push(s);
49             }
50         }
51         while (!list.isEmpty()) {
52             postfix.add(list.pop());
53         }
54         return postfix;
55     }
56
57     LinkedList<String> INTOPRE(LinkedList<String> infix) {
58         LinkedList<String> prefix = new LinkedList<>();
59         LinkedList<String> list = new LinkedList<>();
60         for (int i=infix.size()-1; i>=0; i--) { //5 + 2 / 2 - 6
61             String s = infix.get(i);
62             if (isNumeric(s)){
63                 prefix.addFirst(s);
64             }else{
65                 while (!list.isEmpty() && precedence(s)<precedence(list.peek())) {
66                     prefix.addFirst(list.pop());
67                 }
68                 list.push(s);
69             }
70         }
71         while (!list.isEmpty()) {
72             prefix.addFirst(list.pop());
73         }
74         return prefix;
75     }
76 }

```



```
1  class Node{
2      String val;
3      Node left;
4      Node right;
5
6      Node(String val) {
7          this.val = val;
8          left = null;
9          right = null;
10     }
11 }
12
```

Infix: 5 + 2 / 2 - 6  
Prefix: - + 5 / 2 2 6  
Postfix: 5 2 2 / + 6 -

6  
-  
2  
/  
2  
+  
5