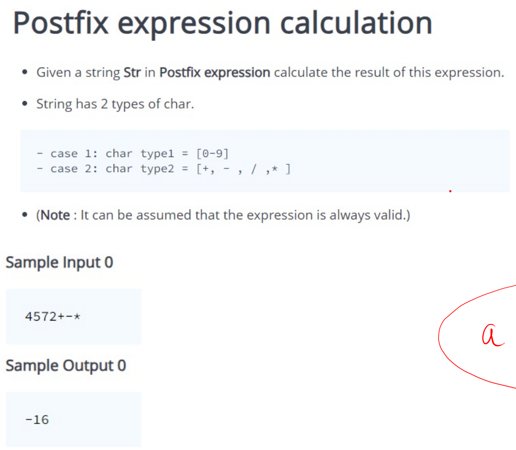


Postfix expression calculation

- Given a string **Str** in **Postfix expression** calculate the result of this expression.
- String has 2 types of char.

```
- case 1: char type1 = [0-9]
- case 2: char type2 = [+, - , / ,* ]
```

- (**Note** : It can be assumed that the expression is always valid.)

Sample Input 0

```
4572+-*
```

Sample Output 0

```
-16
```

$a + b = 10$ → equation.

$a + b$

+ → operator

→ operation

a, b → operands.

* infix notation

↳ operator in b/w operand

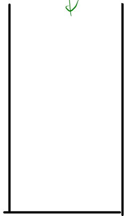
* postfix notation

↳ operator after operand

eg. 4 5 7 2 + - *

↳ int

Integer.



[illegible]

v1 v2

$7 + 2 = 9$

$5 - 9 = -4$

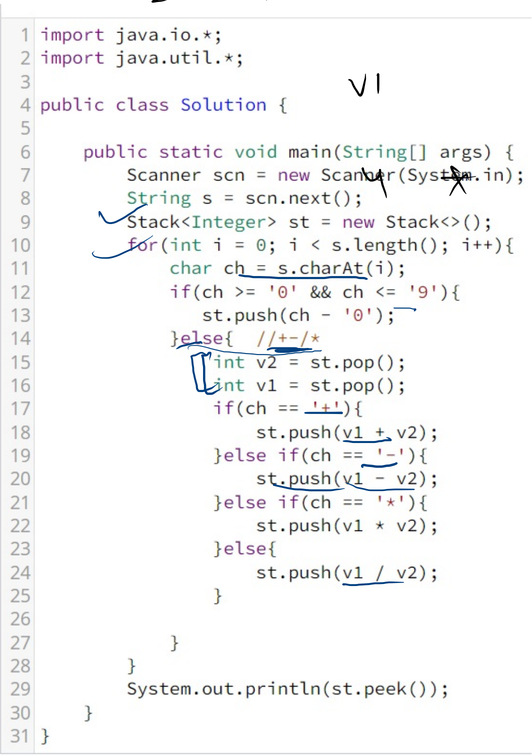
$4 * -4 = -16$

first remove v2

↓

v1

4 5 7 2 + - *

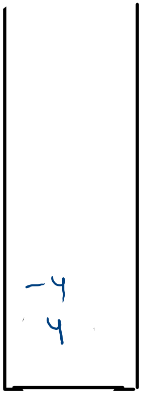


```
import java.io.*;
import java.util.*;

public class Solution {

    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);
        String s = scn.next();
        Stack<Integer> st = new Stack<>();
        for(int i = 0; i < s.length(); i++){
            char ch = s.charAt(i);
            if(ch >= '0' && ch <= '9'){
                st.push(ch - '0');
            }else{  //+-/*
                int v2 = st.pop();
                int v1 = st.pop();
                if(ch == '+'){
                    st.push(v1 + v2);
                }else if(ch == '-'){
                    st.push(v1 - v2);
                }else if(ch == '*'){
                    st.push(v1 * v2);
                }else{
                    st.push(v1 / v2);
                }

            }
        }
        System.out.println(st.peek());
    }
}
```

V1

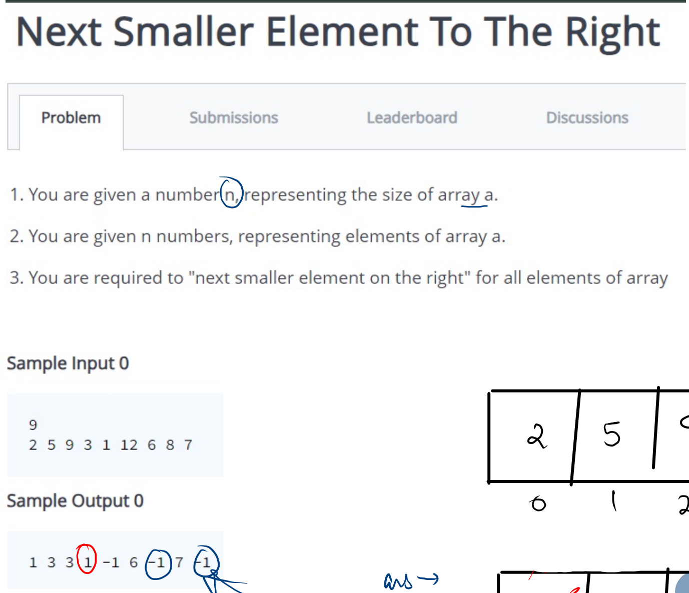
v1 v2

5 - 9 = -4



8 2 /

8 / 2 = 4



Next Smaller Element To The Right

1. You are given a number n, representing the size of array a.

2. You are given n numbers, representing elements of array a.

3. You are required to "next smaller element on the right" for all elements of array

Sample Input 0

```
9
2 5 9 3 1 12 6 8 7
```

Sample Output 0

```
1 3 3 1 -1 6 -1 7 -1
```

ans →

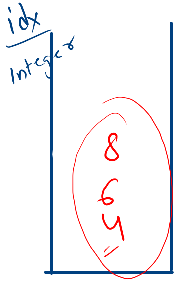
nsor

nsol
ngor
ngol

n = 9

2	5	9	3	1	12	6	8	7
0	1	2	3	4	5	6	7	8

~~-1~~ 1	~~-1~~ 3	3 ~~-1~~	~~-1~~ 1	-1	~~-1~~ 6	-1	7 ~~-1~~	-1
0	1	2	3	4	5	6	7	8



while (A[i] < A[st.peek])

idx = st.pop()
idx = 7

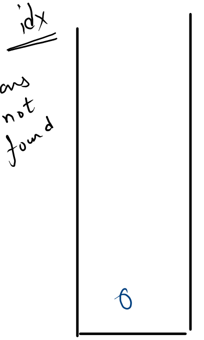
7 < A[7]

7 < 8

2	5	9	3	1	12	6	8	7
0	1	2	3	4	5	6	7	8

ans

~~-1~~ 1	~~-1~~ 3	~~-1~~ 3	~~-1~~ 1	-1	6 ~~-1~~	-1	7 ~~-1~~	-1

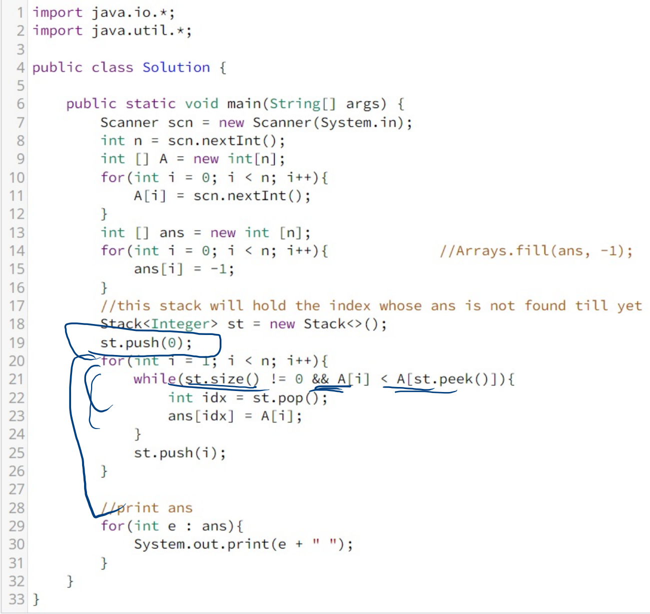


7<A[6]

7< 8

7<6

idx = 7



```
import java.io.*;
import java.util.*;

public class Solution {

    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);
        int n = scn.nextInt();
        int [] A = new int[n];
        for(int i = 0; i < n; i++){
            A[i] = scn.nextInt();
        }
        int [] ans = new int [n];
        for(int i = 0; i < n; i++){                 //Arrays.fill(ans, -1);
            ans[i] = -1;
        }
        //this stack will hold the index whose ans is not found till yet
        Stack<Integer> st = new Stack<>();
        st.push(0);
        for(int i = 1; i < n; i++){
            while(st.size() != 0 && A[i] < A[st.peek()]){
                int idx = st.pop();
                ans[idx] = A[i];
            }
            st.push(i);
        }

        //print ans
        for(int e : ans){
            System.out.print(e + " ");
        }
    }
}
```