Valid Parentheses 4

Given a string s containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.

An input string is valid if:

- Open brackets must be closed by the same type of brackets.
- Open brackets must be closed in the correct order.

Sample Input 0

```
()[]{}
```

Sample Output 0

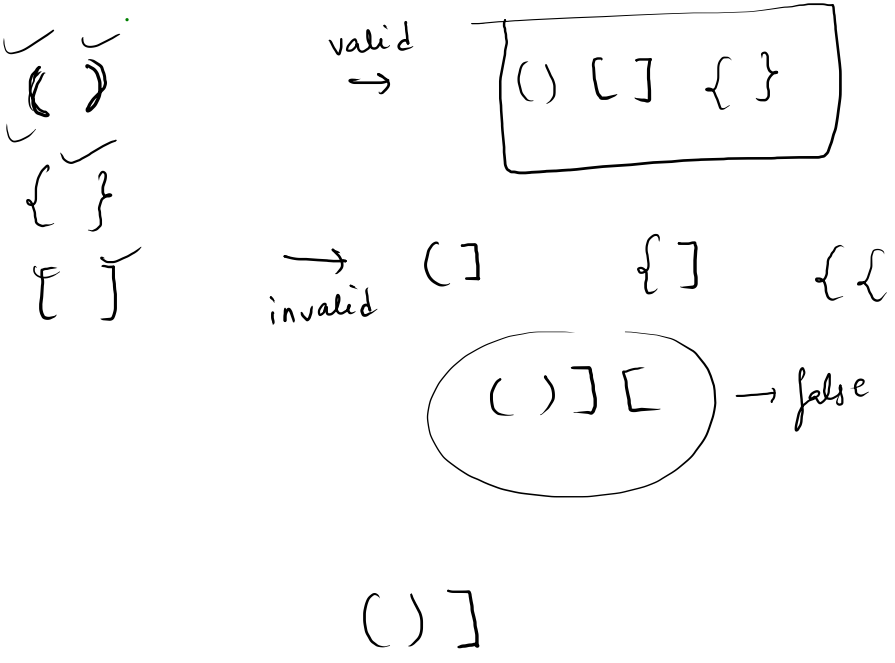
```
true
```

Sample Input 1

```
(]
```

Sample Output 1

```
false
```



[ ] { ( ) } } ( ) ⇝ false.

open → push

ch == ']' st.peek() != '[' → false

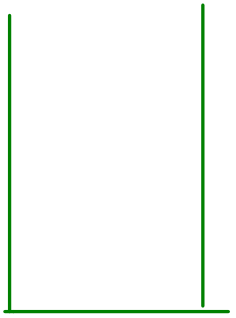
ch == ')' st.peek() != '(' → false

ch == '}' st.peek != '{' → false

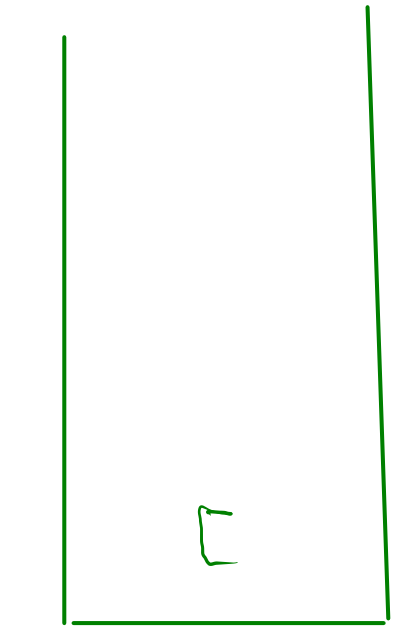
[ ] { ( ) } (( ))

0 1 2 3 4 5 6 7 8 9

i

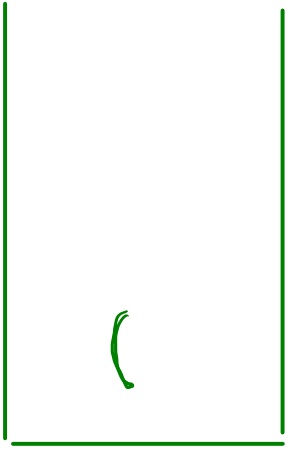


if (st.size == 0) → true / false.

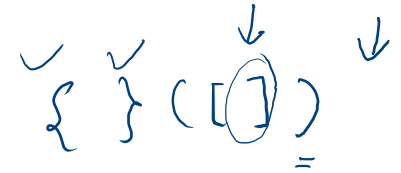


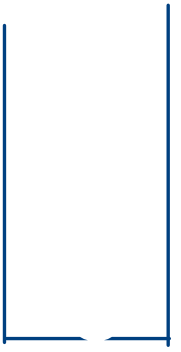
eg.] ( ) [ ]

eg 2. ( ) (

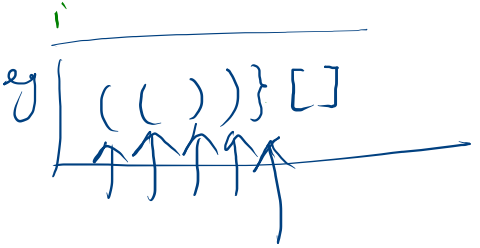


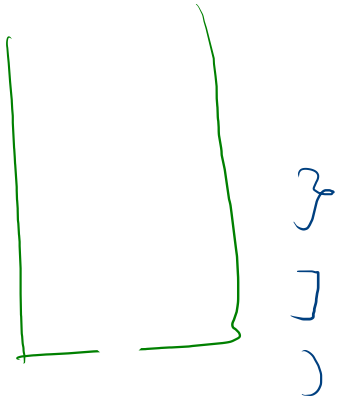
```java
public class Solution {
    public static boolean isValid(String s){
        Stack<Character> st = new Stack<>();
        for(int i = 0; i < s.length(); i++){
            char ch = s.charAt(i);
            if(ch == '(' || ch == '{' || ch == '['){
                st.push(ch);
            }else{        //]})
                if(st.size()==0){
                    return false;
                }
                else if(ch == ')' && st.peek() != '('){
                    return false;
                }
                else if(ch == ']' && st.peek() != '['){
                    return false;
                }
                else if(ch == '}' && st.peek() != '{'){
                    return false;
                }else{
                    st.pop();
                }
            }
        }
        return st.size()==0;
    }

    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);
        String s = scn.next();
        System.out.println(isValid(s));
    }
```





```java
public class Solution {
    public static boolean isValid(String s){
        Stack<Character> st = new Stack<>();
        for(int i = 0; i < s.length(); i++){
            char ch = s.charAt(i);
            if(ch == '(' || ch == '{' || ch == '['){
                st.push(ch);
            }else{        //]})
                if(st.size()==0){
                    return false;
                }
                else if(ch == ')' && st.peek() != '('){
                    return false;
                }
                else if(ch == ']' && st.peek() != '['){
                    return false;
                }
                else if(ch == '}' && st.peek() != '{'){
                    return false;
                }else{
                    st.pop();
                }
            }
        }
        return st.size()==0;
    }

    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);
        String s = scn.next();
        System.out.println(isValid(s));
    }
}
```





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

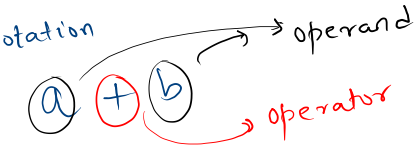
```
4572+-*
```

Sample Output 0

```
-16
```

+ ab → prefix

infix notation



2 + 4

Digits. ch >= '0' && ch <= '9'

operator + - * /

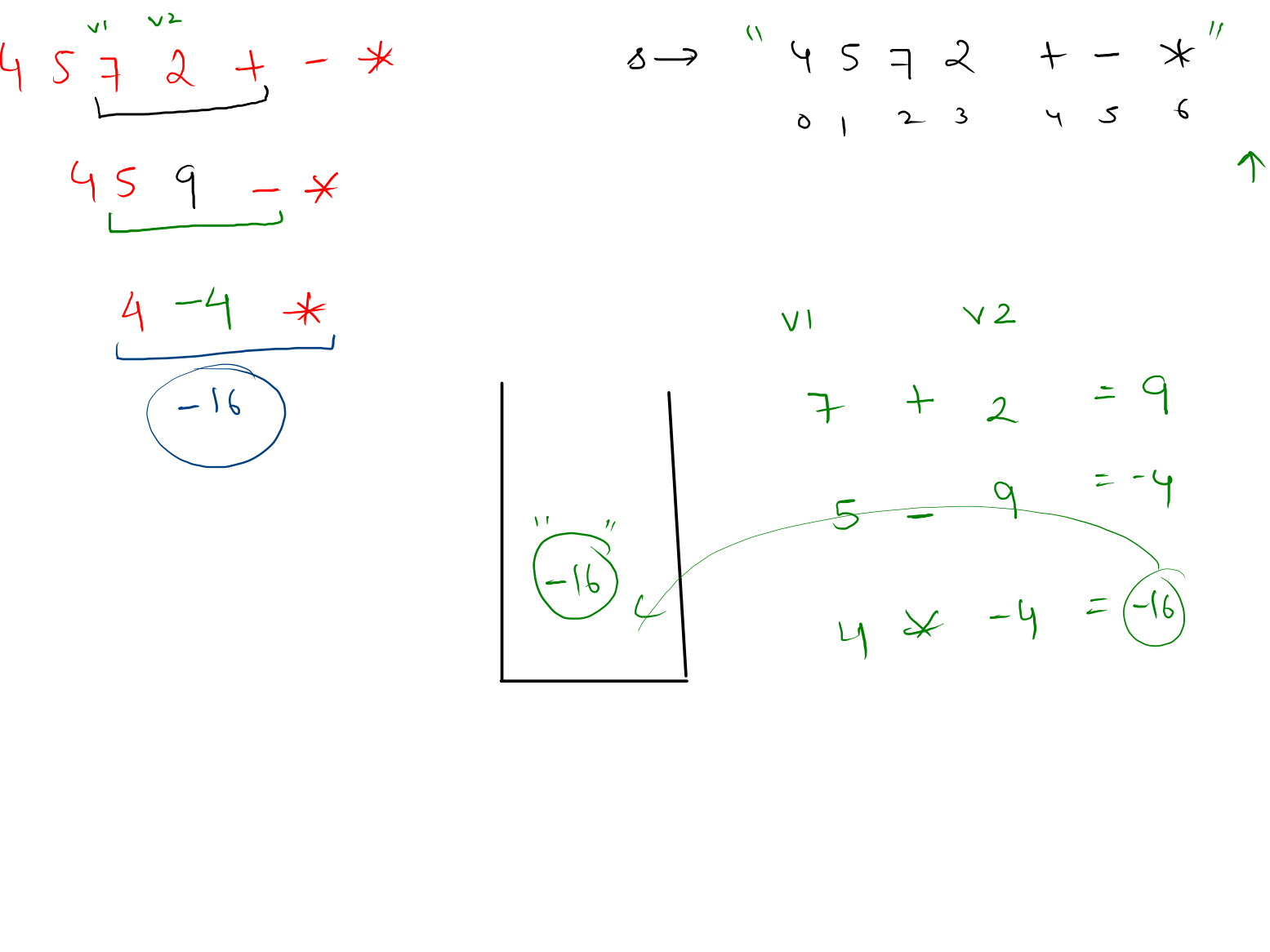
postfix → postfix notation.

a b +

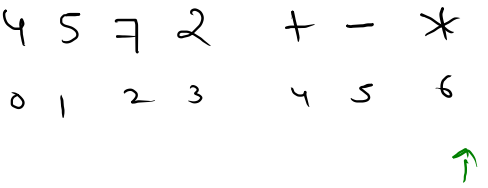
remember:

9 - 2 ≠ 2 - 9

9+2 == 2+9


v1 v2
4 5 7 2 + - *
4 5 9 - *
4 -4 *
-16
s → " 4 5 7 2 + - * "
0 1 2 3 4 5 6
v1 v2
7 + 2 = 9
5 - 9 = -4
4 * -4 = -16
" "
-16

```java
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());
    }
}
```



```java
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<String> st = new Stack<>();
        for(int i = 0; i < s.length(); i++){
            char ch = s.charAt(i);

            if(ch >= '0' && ch <= '9'){
                st.push("" + ch);
            }else{
                int v2 = Integer.parseInt(st.pop());
                int v1 = Integer.parseInt(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());
    }
}
```

Asteroid Collision

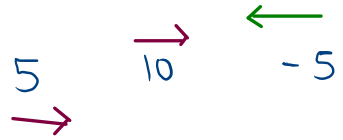
We are given an array asteroids of integers representing asteroids in a row.

For each asteroid, the absolute value represents its size, and the sign represents its direction (positive meaning right, negative meaning left). Each asteroid moves at the same speed.

Find out the state of the asteroids after all collisions. If two asteroids meet, the smaller one will explode. If both are the same size, both will explode. Two asteroids moving in the same direction will never meet.

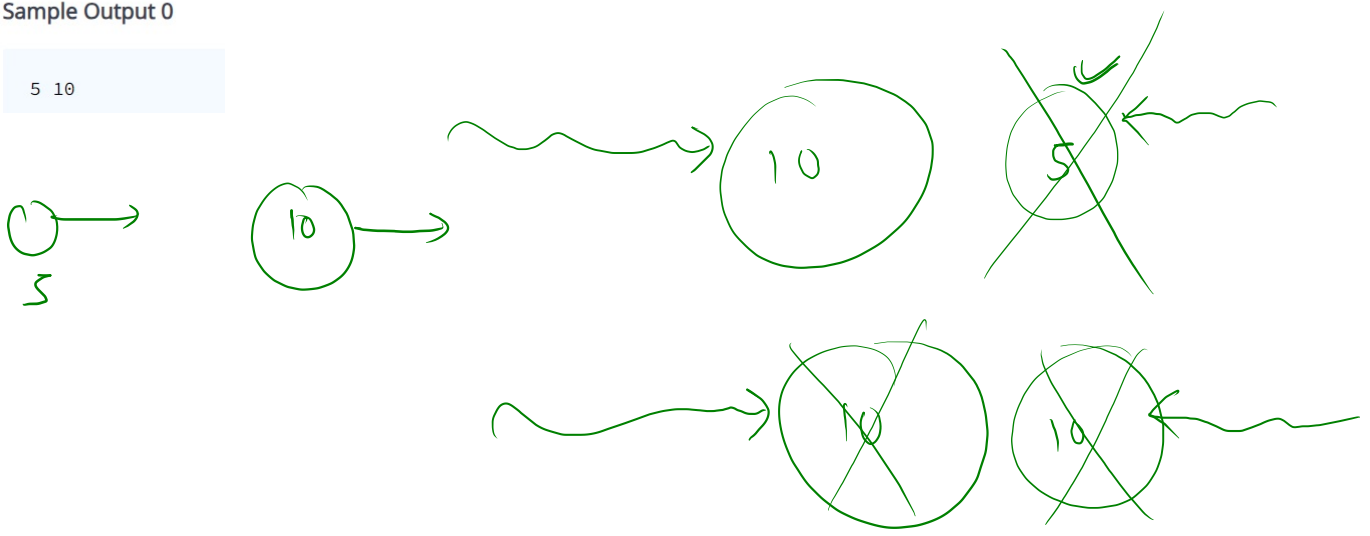
Sample Input 0

```
3
5
10
-5
```



Sample Output 0

```
5 10
```



∞ 10 4 2 −5 −6

c_1

10 4 −5 −6

c_2

10 −5 −6

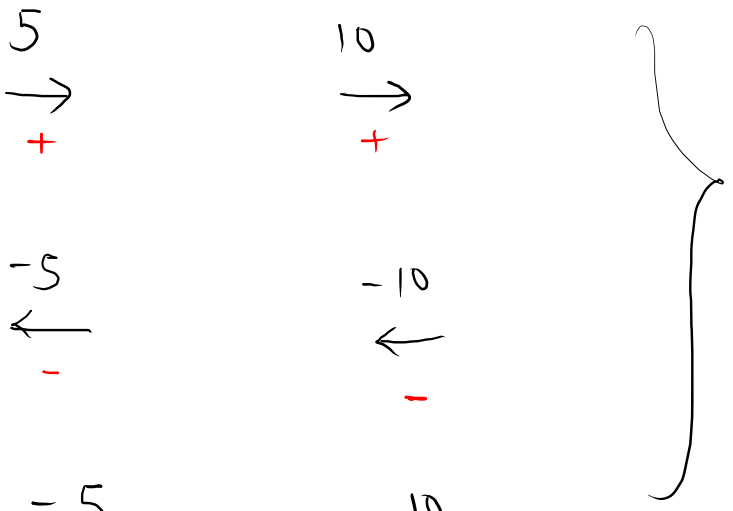
c_3

10 −6

c_4

$= 10$

Case 1.

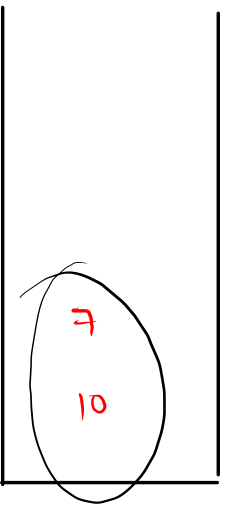

5
10
+
+
No
collision
Case 2
-5
-10
-
-
case 3.
-5
10
+
***
case 4.
10
-5
collision
+
-
case 5.
-2
+
-

eg 10 7 4 2 X -2 0 -5 0 -6

curr

-ve 0

⇒ peek +ve



sum = 1 - 2 = -ve → curr > peek
↳ pop

1 -2

sum = 2 - 2 = = 0 → curr = peek
↳ pop
curr = 0

sum = 4 - 5 = -ve → pop

sum 7 - 5 = 2 = +ve → curr = 0

7 - 6 = 1 = +ve

```
        //collision (+ -)
        for(int i = 0; i < n; i++){
            int curr = A[i];
            while(st.size() != 0 && curr < 0 && st.peek() > 0){    //collision
                int sum  = st.peek() + curr;
                if(sum > 0){        //st.peek() > curr
                   curr = 0;
                }else if(sum == 0){
                    curr = 0;
                    st.pop();
                }else{              //st.peek() < curr
                    st.pop();
                }
            }

            if(curr != 0){
                st.push(curr);
            }
        }

        for(int ele : st){
            System.out.print(ele + " ");
        }

    }
}
```