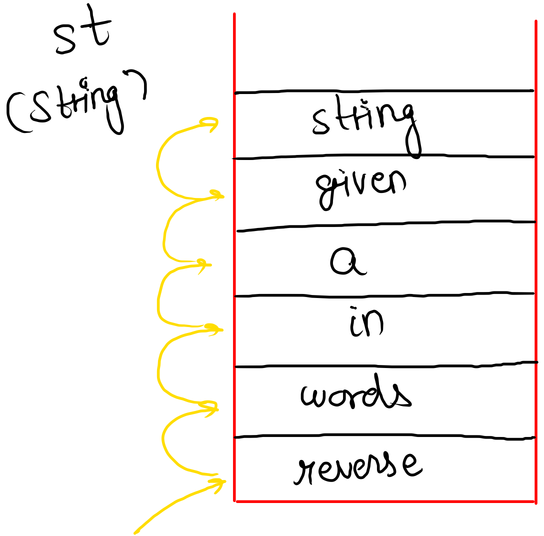
Reverse Words in a Given String

str = "reverse_words_in_a_given_string"

arr = str.split(" ")

0	1	2	3	4	5
"reverse"	"words"	"in"	"a"	"given"	"string'

use stack to reverse the arr



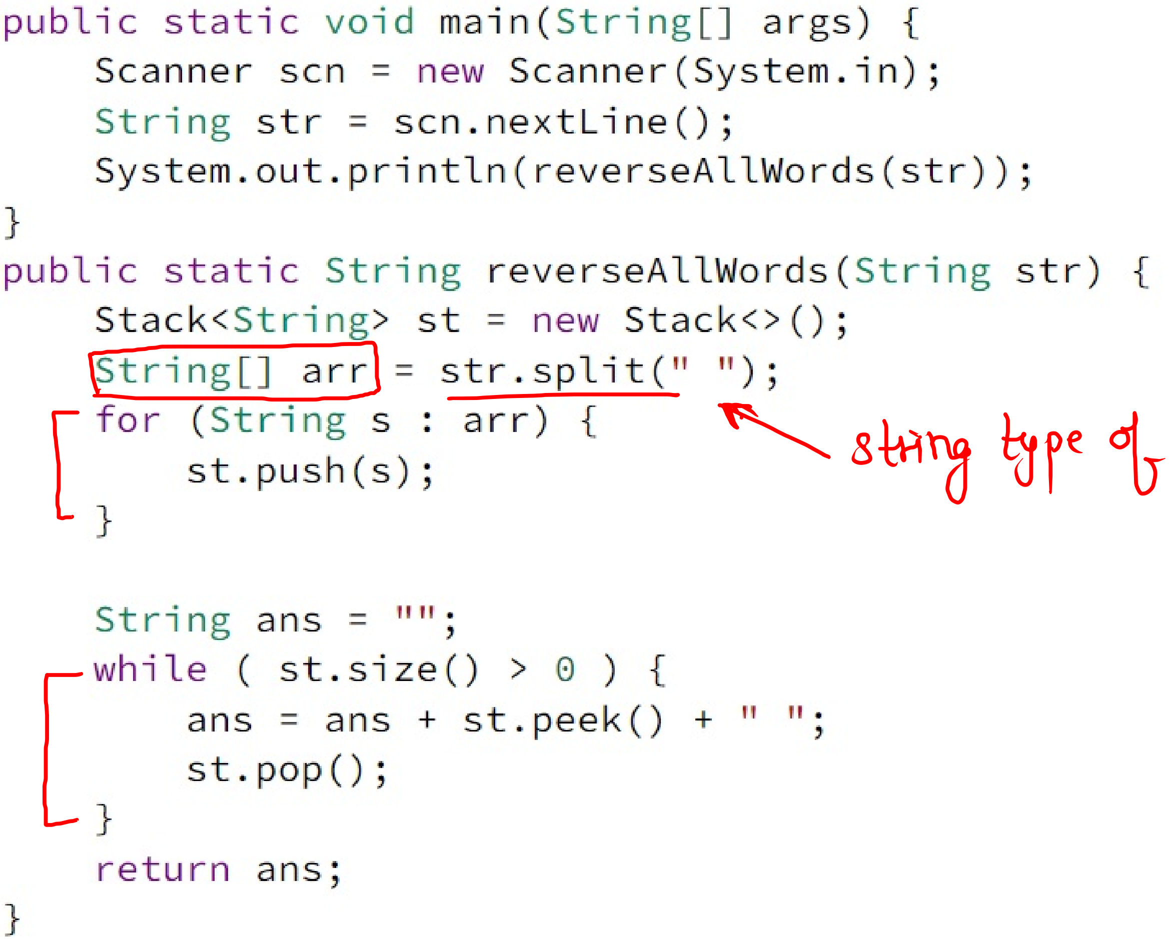
loop until st.size() > 0

ans = ans + st.peek() + " ";

st.pop();

code

T.C = O(N), N= size of stack



```
public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);
    String str = scn.nextLine();
    System.out.println(reverseAllWords(str));
}
public static String reverseAllWords(String str) {
    Stack<String> st = new Stack<>();
    String[] arr = str.split(" ");
    for (String s : arr) {
        st.push(s);
    }

    String ans = "";
    while ( st.size() > 0 ) {
        ans = ans + st.peek() + " ";
        st.pop();
    }
    return ans;
}
```

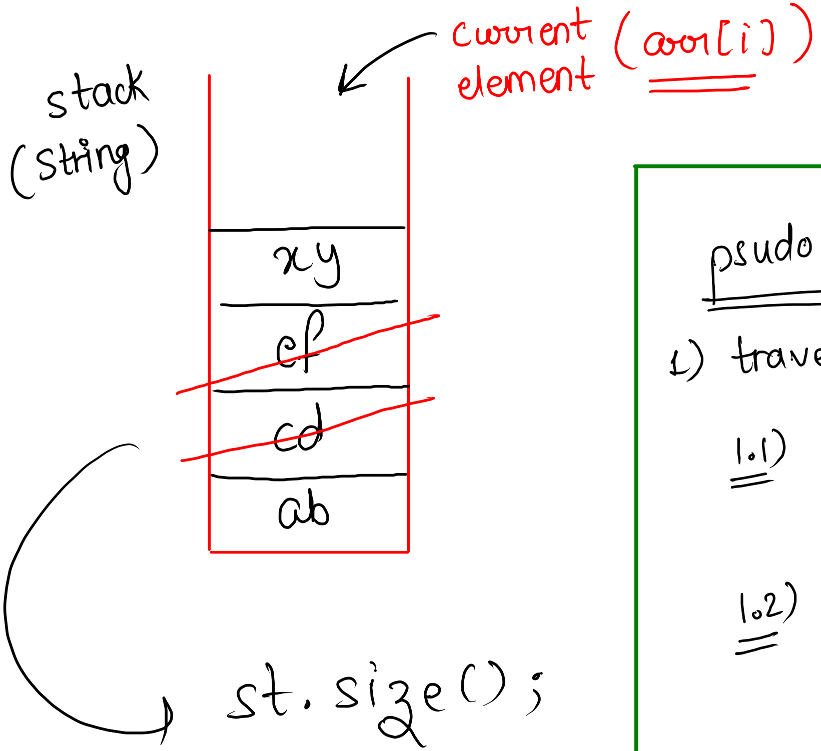
string type of arr

Delete consecutive

arr =

0	1	2	3	4	5
ab	cd	ef	ef	cd	xy

len = 2

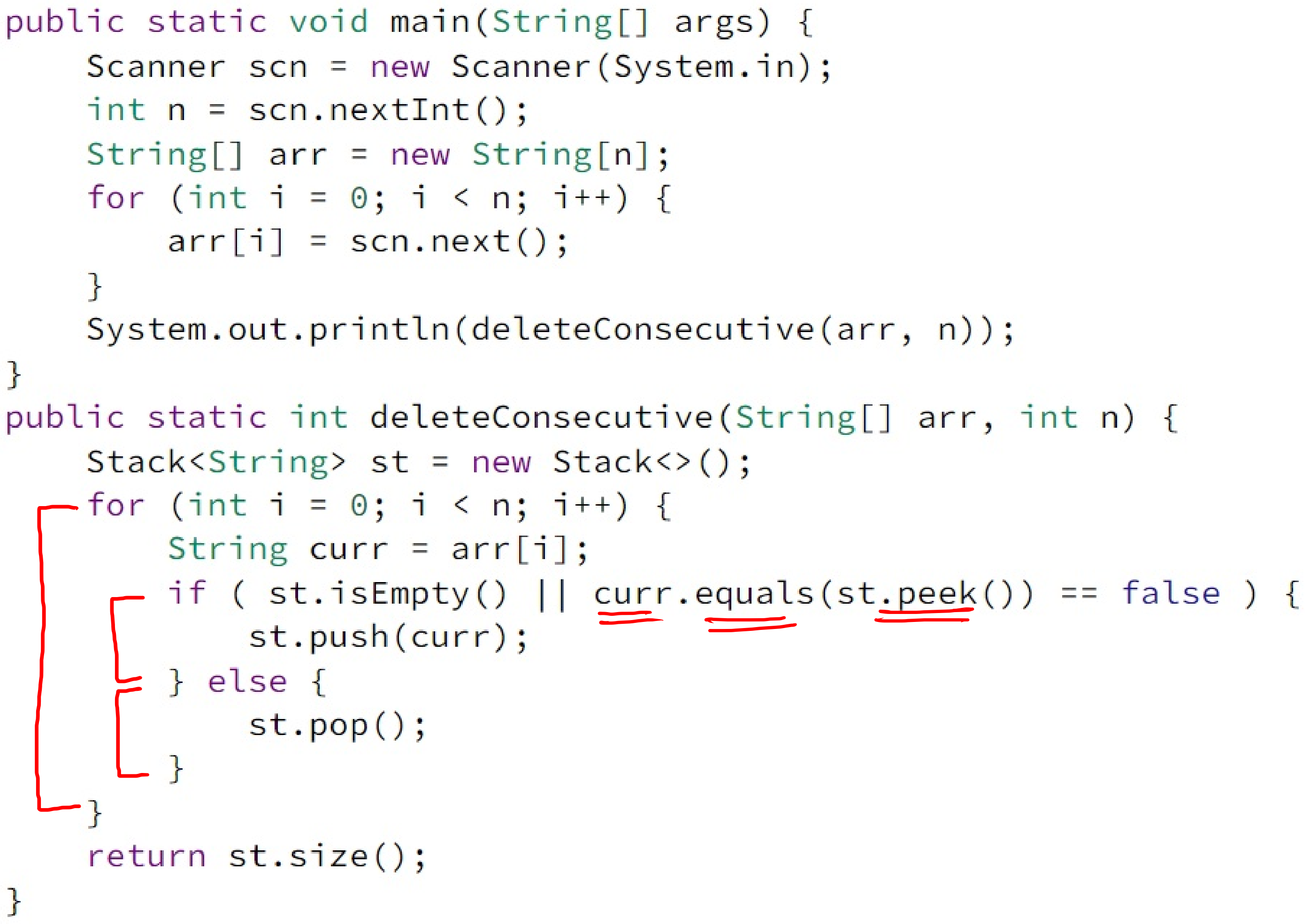


st.size();

psudo code

```
1) traverse from 0 to n-1 in arr
    1.1) if peek != curr
            push curr
    1.2) else
            pop
```

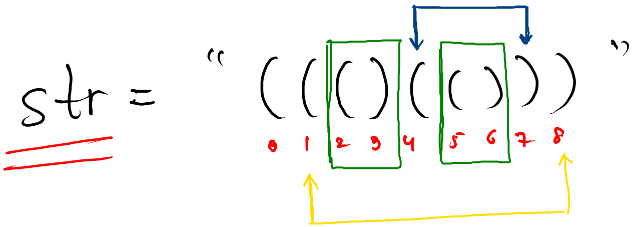
Code



```
public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);
    int n = scn.nextInt();
    String[] arr = new String[n];
    for (int i = 0; i < n; i++) {
        arr[i] = scn.next();
    }
    System.out.println(deleteConsecutive(arr, n));
}
public static int deleteConsecutive(String[] arr, int n) {
    Stack<String> st = new Stack<>();
    for (int i = 0; i < n; i++) {
        String curr = arr[i];
        if ( st.isEmpty() || curr.equals(st.peek()) == false ) {
            st.push(curr);
        } else {
            st.pop();
        }
    }
    return st.size();
}
```

Valid Parentheses 4

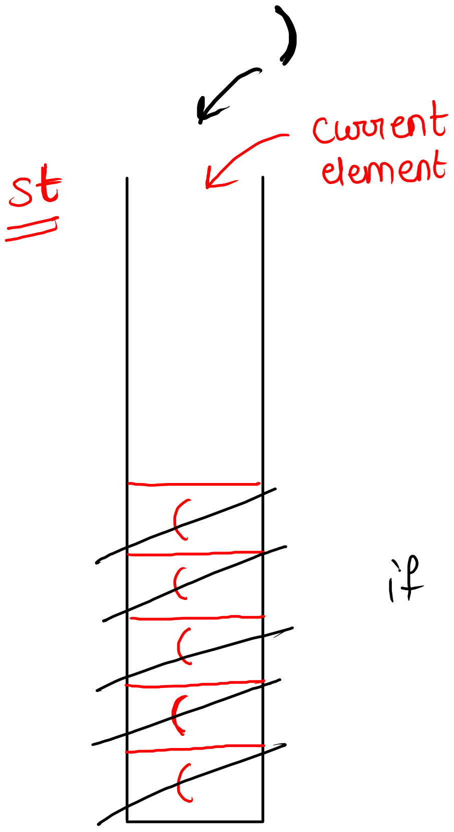
'(', ')'



assign meaning to stack

St :- stack will contain only invalid parenthesis st = "()((()()))"

0 1 2 3 4 5 6 7 8 9



psudo code

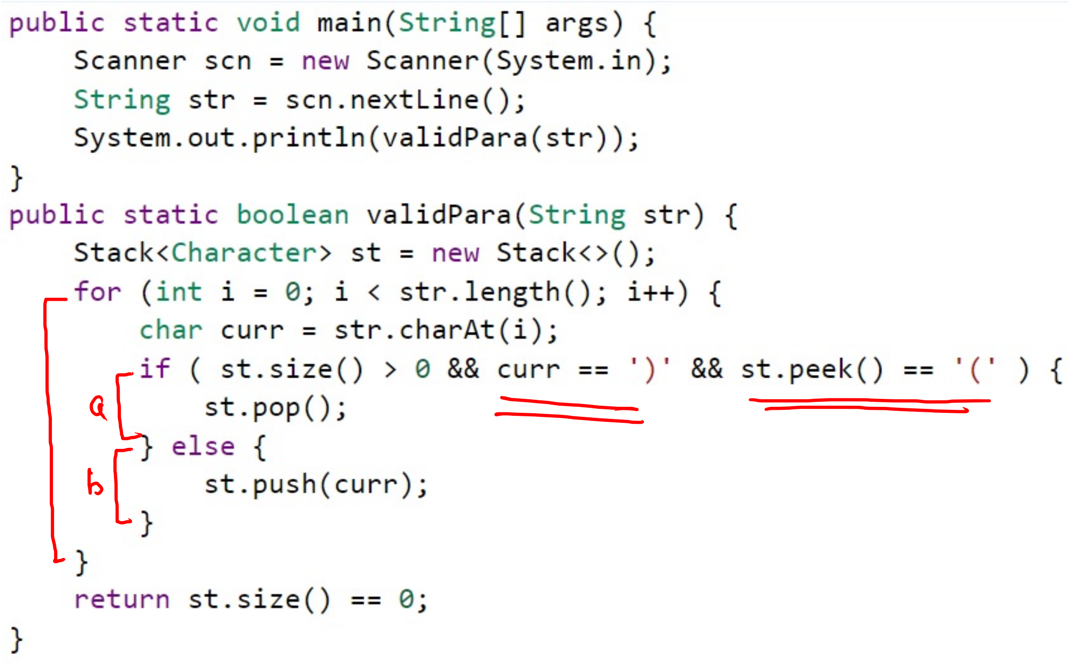
```
1) traverse from start to end
    1.1) if we get open para.
              push open para.
    1.2) else if curr == ')' &&
                 peek == '('

         pop
```

if size of st == 0
        valid
else
        invalid

code

when we have only paranthesis in given string



```
public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);
    String str = scn.nextLine();
    System.out.println(validPara(str));
}
public static boolean validPara(String str) {
    Stack<Character> st = new Stack<>();
    for (int i = 0; i < str.length(); i++) {
        char curr = str.charAt(i);
        if ( st.size() > 0 && curr == ')' && st.peek() == '(' ) {
            st.pop();
        } else {
            st.push(curr);
        }
    }
    return st.size() == 0;
}
```

now, we have all brackets

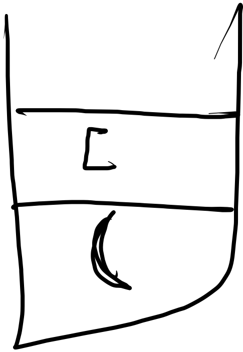
top curr

[[({ })]] valid

([ )] invalid

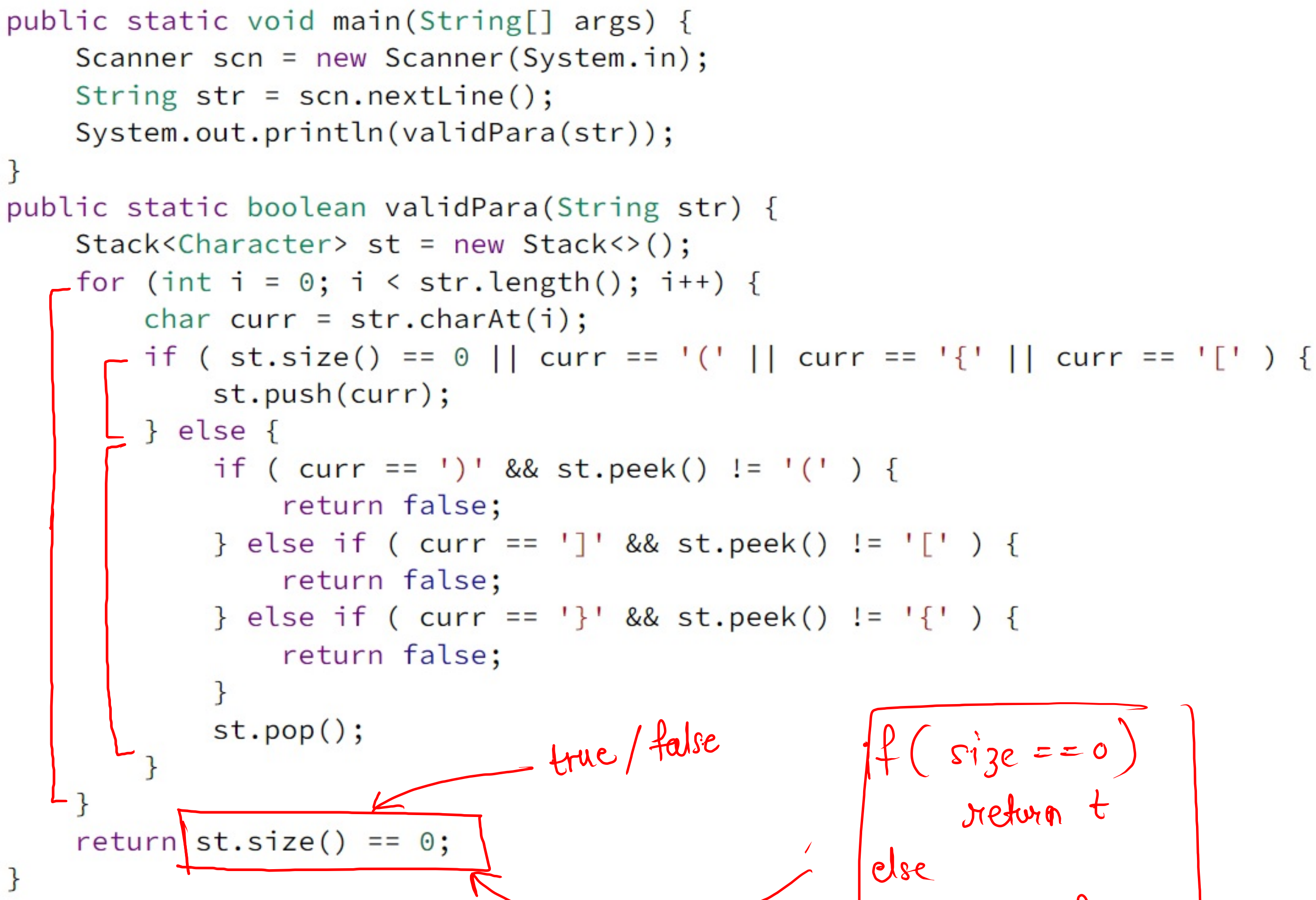
[{ } ( ) { } invalid

[{ } ( ) { }] valid



code

T.C = O(N), N = str size



```java
public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);
    String str = scn.nextLine();
    System.out.println(validPara(str));
}
public static boolean validPara(String str) {
    Stack<Character> st = new Stack<>();
    for (int i = 0; i < str.length(); i++) {
        char curr = str.charAt(i);
        if ( st.size() == 0 || curr == '(' || curr == '{' || curr == '[' ) {
            st.push(curr);
        } else {
            if ( curr == ')' && st.peek() != '(' ) {
                return false;
            } else if ( curr == ']' && st.peek() != '[' ) {
                return false;
            } else if ( curr == '}' && st.peek() != '{' ) {
                return false;
            }
            st.pop();
        }
    }
    return st.size() == 0;
}
```

true / false

```
if ( size == 0 )
    return t
else
    retn f
```