

## Day -58 of the #101 days of the coding challenge-----

**Problem:-** Finding the longest Common prefix into the string array.

**Ex:-**

**Input:** strs = ["flower", "flow", "flight"]


**Output:** "fl"

**Solution:-**

```
public String longestCommonPrefix(String[] strs) {  
    String prefix = strs[0]; // storing the initial string  
  
    for(int i = 1; i<strs.length; i++)  
    {  
        while(strs[i].indexOf(prefix)!=0) // finding the prefix(!= 0, means not  
matched)  
        {  
            prefix = prefix.substring(0, prefix.length() - 1); // if not match then  
length of the string will decrease  
        }  
    }  
  
    return prefix;  
}
```

 Accepted

 Editorial


 Solution

 Runtime

Details

**0** ms

Beats 100.00% of users with Java

 Memory

Details

**40.00** MB

Beats 89.67% of users with Java

## Problem:- Valid Parenthesis:-

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

An input string is valid if:

1. Open brackets must be closed by the same type of brackets.
2. Open brackets must be closed in the correct order.
3. Every close bracket has a corresponding open bracket of the same type.

Ex:-

|                                     |
|-------------------------------------|
| <b>Input:</b> <code>s = "()"</code> |
| <b>Output:</b> <code>true</code>    |

|                                    |
|------------------------------------|
| <b>Input:</b> <code>s = "("</code> |
| <b>Output:</b> <code>false</code>  |

Solutions:-

```
public boolean isValid(String s) {  
  
    Stack<Character> stack = new Stack();  
  
    // now converting string into the char array  
    for(char ch : s.toCharArray())  
    {  
        if(ch == '(' || ch == '{' || ch == '[')  
        {  
            stack.add(ch);  
        }  
        else  
        {  
            if(stack.isEmpty()) return false;  
            if(ch == '}' && stack.peek() != '{') return false;  
            if(ch == ']' && stack.peek() != '[') return false;  
            if(ch == ')' && stack.peek() != '(') return false;  
            stack.pop();  
        }  
    }  
}
```

```
        return stack.isEmpty();  
    }  
}
```

✓ Accepted

Editorial

+ Solution

⌚ Runtime Details

2 ms

Beats 83.13% of users with Java

💾 Memory Details

40.55 MB

Beats 37.64% of users with Java

## Problem:-

Given two strings **needle** and **haystack**, return the index of the first occurrence of **needle** in **haystack**, or -1 if **needle** is not part of **haystack**.

Solution:-

```
public int strStr(String haystack, String needle) {  
  
    int n;  
    if(haystack.contains(needle)) // if needle contains  
    {  
        n = haystack.indexOf(needle); // finds the firstOccurence of the string  
char.  
    }  
  
    else  
    {  
        return -1;  
    }  
  
    return n;  
}
```

✓ Accepted

📖 Editorial

+ Solution

🕒 Runtime

Details

0 ms

Beats 100.00% of users with Java

💾 Memory

Details

40.36 MB

Beats 59.20% of users with Java

Problem:-

Given a string *s* consisting of words and spaces, return *the length of the **last** word in the string.*

A **word** is a maximal

substring

consisting of non-space characters only.

**Example 1:**

**Input:** *s* = "Hello World"

**Output:** 5

**Explanation:** The last word is "World" with length 5.

**Example 2:**

**Input:** *s* = " fly me to the moon "

**Output:** 4

**Explanation:** The last word is "moon" with length 4.

Solution:-

```
public int lengthOfLastWord(String s) {  
  
    int count = 0;  
    s = s.trim();  
  
    for(int i = s.length()-1; i>=0; i--)  
    {
```

```

        if(s.charAt(i) == ' ')
        {
            break;
        }
        else
        {
            count++;
        }
    }

    return count;
}

```

✓ Accepted

Editorial

+ Solution

⌚ Runtime

Details

0 ms

Beats 100.00% of users with Java

💾 Memory

Details

40.61 MB

Beats 52.54% of users with Java

Problem:-

A phrase is a **palindrome** if, after converting all uppercase letters into lowercase letters and removing all non-alphanumeric characters, it reads the same forward and backward. Alphanumeric characters include letters and numbers.

Given a string *s*, return true *if it is a **palindrome**, or* false *otherwise*.

Ex:-

**Input:** *s* = "A man, a plan, a canal: Panama"

**Output:** true

**Explanation:** "amanaplanacanalpanama" is a palindrome.

**Input:** *s* = ""

**Output:** true

**Explanation:** *s* is an empty string "" after removing non-alphanumeric characters. Since an empty string reads the same forward and backward, it is a palindrome.

Solution:-

```
public boolean isPalindrome(String s) {

    String str = "", strrv = "";
    int k = 0;

    // removing alphanumerics
    for(int i = 0; i<s.length(); i++)
    {
        char c = s.charAt(i);
        if(Character.isLetterOrDigit(c)) // it checks if alphabet then only true
        {
            str += Character.toLowerCase(c); // converting in lower case to avoid
the case sensitivity
        }
    }

    // reversing the string
    for(int i = str.length() - 1; i>=0; i--)
    {
        strrv += str.charAt(i);
    }

    // comparing the
    return str.equals(strrv);
}
```

✓ Accepted

📖 Editorial

+ Solution

🕒 Runtime

Details

**237** ms

**Beats 5.16%** of users with Java

💾 Memory

Deta

**44.84** MB

**Beats 10.92%** of users with Java