Strings

Assignment Solutions

PW SKILLS

Problem 1: Given a string str, count and print the number of vowels and consonants in the string.
Example: —› **Input1:** physicswallah
Output1: vowels = 3 , consonants = 10
Explanation: i,a,a are vowels rest other characters are consonants.

Input2: coders
Output2: vowels = 2 , consonants = 4

Code Link: https://pastebin.com/i29d7xk8

Output:

```
enter the string: pwcoders
vowels = 2 , consonants = 6
```

Approach:
- We have taken a string as input and traversed the string using a for loop.
- We have checked, if any particular character is one of "a , e , i , o , u" then we need to increase the count of vowels otherwise it is consonant hence we will increase the count of consonants.
- Once we have completely traversed the string we can print the results as observed in the values vowels and consonants.

Problem 2: Given a string str, check whether it is a palindrome or not.
Example: —› **input 1:** coding
Ouput1: No

Input 2: ola
Output2: Yes

Explanation: A palindrome is a string that reads the same from start to end or end to start.

Code link: https://pastebin.com/4E6FhCKY

Output:

```
enter the string: return
No
```

```
enter the string: palap
Yes
```

Approach: A palindrome is a string that reads the same if read from the beginning or from the end.
- We have taken a string as input from the user. We have assigned two pointers **i** and **j**, one at the start and one at the end of the string respectively.
- If at any point the value of ith and jth character does not match then it is a clear indication that this string is not a palindrome.
- Hence we have raised a **flag** and break from the loop because at any cost now the string cannot be considered as a palindrome.
- After the termination of the while loop we checked if the value of flag variable is 1 that means the given string is not a palindrome otherwise it is a palindrome.

Problem 3: How do you reverse a given string in place without using an inbuilt function?
Example 1: —> **Input:** "pwskills"
Output: "sllikswp"
Example 2: **Input:** "PWCoders"
Output: "sredoCWP"

Code link: **https://pastebin.com/cfnhnL6x**

Output:

```
enter the string: pwcoders
the reversed string is: sredocwp
```

Approach:

- In the main function, we have taken a string input from the user.
- We calculated its size, and we traversed till half the length of the string.
- And we swapped the first and last characters, then second and second last characters and so on.
- After the loop terminates we have printed the reversed string.

Problem 4: Given a string str, and a character x. Find the occurrence of character 'x' in the given string.
Example: —> **Input 1:** abyybfurhdr x = 'y'
Output 1: 2
Input 2: codes x = 'a'
Output 2: 0

Code link: **https://pastebin.com/4d4qBHvr**
Output:

```
enter the string: physicswallah
enter the character: a
the character 'a' occurred 2 times in our given string.
```

Approach:

- In the main function we have taken the input of the string and the character.
- We have traversed the string, and if the ith character is exactly equal to the given character then we can increment the value of the count variable.
- After complete traversal of the loop, we can print the value of count.

Problem 5: Given a string that consists of only 0s, 1s and 2s, count the number of substrings that have an equal number of 0s, 1s, and 2s.

Example: -> **Input:** str = "0102010"
Output: 2

Explanation: Substring str[2, 4] = "102" and substring str[4, 6] = "201" has equal number of 0, 1 and 2

Input: str = "102100211"
Output: 5

Code link: https://pastebin.com/R5L7iqFF

Output:

```
enter the string: 0102010
The given string has 2 substrings with equal count of 0 , 1 and 2.
```

Approach:

- In the main function we have taken a string input from the user and called the function "stringWithEqual012" that accepts one parameter which is the string itself.
- In this function we have created a vector of strings with the intent to store all the possible substrings of the given string.
- Once all substrings are generated we have iterated on every substring and checked the count of 0 1 and 2 in a particular substring.

Problem 6: given a string s. Convert all lowercase letters to uppercase and vice versa and print the string.

Input 1: "PWskills"
Output 1: "pwSKILLS"

Input 2: "coDiNGwaLLah"
Output 2: "COdIngWAllAH"

Solution:

Code link: https://pastebin.com/WTYirLT4

Output:

```
Enter the string: PWskiLLs
The changed string will be: pwSKIlls
```

Approach:

- In the **main** function we have taken a string input from the user and made a call to the function **"changeCase"** that accepts the string as a parameter.
- We passed the string as reference because we want the changes to reflect in the string that we had originally.
- In the **"changeCase"** function we traversed the string completely and simply checked if any ith character is in uppercase then we changed it to lowercase and vice versa.
- Here we used three inbuilt functions of strings namely **isupper(), tolower(), toupper()**. All these functions accept a character as parameter and **isupper()** function returns boolean value i.e true if the passed character is in uppercase, false otherwise.
- **toupper()** function converts the passed character to uppercase and returns a character. If the character already is in uppercase then it won't change. Vice versa is the function of **tolower()** function. It will return a character in lowercase.
- Once traversal is complete we will return our string.

Time complexity: O(n) where n = length of the string.
Space complexity: space consumed will be O(1).

Problem 7: given an integer num, return the roman numeral of the integer.

Input 1: 23
Output 1: XXIII

Input 2: 49
Output 2: XLIX

Solution:

Code link: **https://pastebin.com/n1Wdbbym**

Output:

```
Enter the number : 23
The roman numeral for number 23 will be : XXIII
```

Approach:

- In the **main** function we have taken the input of the number and made a call to the function **"intToRoman"** and passed the number as the parameter.
- This function will return the string consisting of roman numerals corresponding to that number.
- In this function we have created two arrays namely **"normal"** and **"roman"** where **"normal"** array consists of those numbers for which in roman either we have special reserved characters or those numbers which have next roman character preceding the previous character. For example 50 , 9 , 40 , 10 etc.
- The **"roman"** array contains corresponding special characters for the above mentioned numbers.
- We have traversed the array till 13 because max value will be in thousands only.[max - 3999]
- Our main idea here is to subtract as greater value as possible in one go and award a corresponding roman numeral against that value.
- We have used here the **"append()"** function that takes a string as a parameter and simply concatenates at the end of the string calling the append function.
- Once the roman part for a particular value is organized we will subtract that amount from the original number and we will repeat this process until the original number turns out to be 0.
- This way we will have our answer stored in the string **"result"** and at last we will return this result.

Time complexity: O(number) .
Space complexity: O(1).

Problem 8: Given a list of words followed by two words, the task is to find the minimum distance between the given two words in the list of words.
Note: both word1 and word2 will exist in the list of the words.

Input 1: S = { "the", "quick", "brown", "fox", "quick"}, word1 = "the", word2 = "fox"
Output 1: 3

Explanation: Minimum distance between the words "the" and "fox" is 3

Input 2: S = {"geeks", "for", "geeks", "contribute", "practice"}, word1 = "geeks", word2 = "practice"
Output 2: 2

Explanation: Minimum distance between the words "geeks" and "practice" is 2

Solution:

Code link: https://pastebin.com/6gufydeY

Output:

```
enter the number of strings : 5
Enter the strings : the quick brown fox quick
Enter first word : the
Enter second word : fox
The  shortest distance between the and fox is : 3
```

Approach:

- In the **main** function we have taken the array of strings and the two words as input and called the function **"shortestDistance"** accepting the inputted values as parameters.
- The return type of **"shortestDistance"** function is of **"int"**.
- In the function **"shortestDistance"** we initialize the two **"distance1"** and **"distance2"** variables as -1 since we didn't find out the exact indices of the two words as of now.
- We have initialized the **"answer"** variable as **INT_MAX** currently as we need to find the minimum distance between those two words.
- We traverse on the vector and once we find out the location of both the words we keep on storing the possible lowest distance because there can be many occurrences of the same word.
- On complete traversal of the vector we have our minimum most answer stored in the **"answer"** variable and we will return that value.

Time complexity: O(n) where n = length of the vector of strings.

Space complexity: we have declared a few variables only apart from the input that will cost us almost constant space. Therefore space consumed will be O(1).

Problem 9: Given a length n, count the number of strings of length n that can be made using 'a', 'b' and 'c' with at most one 'b' and two 'c's allowed.

Input 1: n = 3
Output 1: 19

Explanation: Below strings follow given constraints:
aaa aab aac aba abc aca acb acc baa
bac bca bcc caa cab cac cba cbc cca ccb

Input 2: n = 4
Output 2: 39

Solution:

Code link: https://pastebin.com/K4PicigW

Output:

```
enter the number : 4
The number of strings possible are 39
```

Approach:

- In the **main** function we have taken the length as input and called for the function **"countStrings"** that accepts 3 parameters, one is the length of the string, second is the number of b's allowed and the third one is number of c's allowed.
- In the **"countStrings"** function the base condition should be that if the **"length"** is equal to 0 that means we have found at least one possible string under the given constraints.
- Now arises three recursive calls.
- The first one where we are not at all using any value of **'b'** and **'c'**, the resultant string will only contain **'a'** because there is no restriction on the number of a's to be used.
- In this case we have passed parameters **length - 1 , bAllowed , cAllowed,** length will decrease by 1, since we are not using any **'b', 'c'** therefore values of bAllowed and cAllowed did not change.
- The second case is if we have at least one value of **'b'** remaining we can fill the next character in the string as **'b'**. Though we are not creating the resultant string as it is not required in this question, we are just visualizing the string formation.
- The parameters for this call will be **length - 1 , bAllowed - 1 , cAllowed**.
- The third and last call will be if there is at least one value of **'c'** remaining we can again make a recursive call with parameters **length - 1 , bAllowed , cAllowed - 1**.
- Since we are counting total answers we will sum up all these possibilities and we will return the sum.

Time complexity: in the worst case scenario, every time 3 recursive calls are made, so if the length is n the time complexity would be O(3^n).

Space complexity: there will be a space consumed by a call stack of recursion equivalent to the maximum length of the string i.e n.

Therefore space complexity will be O(n).

Problem 10: Given two positive numbers as strings. The numbers may be very large (may not fit in long long int), the task is to find product of these two numbers.

Input 1: num1 = 4154
num2 = 51454

Output 1: 213739916

Input 2: num1 = 6541541541514545454154154 54
num2 = 6351656156315631654514514 6514654

Output 2: 4154962260395530977724371606999799700762043993771150906 2916

Solution:

Code link: **https://pastebin.com/grmZG0kq**

Output:

```
Enter first number : 23
Enter second number : 23
The multiplicative product of these two numbers is : 529
```

Approach:

- We start from the last digit of the second number and multiply it with the first number.
- Then we multiply the second digit of the second number with the first number, and so on.
- We add all these multiplications. While adding, we put i-th multiplication shifted.
- The approach used in the solution below is to keep only one array for the result.
- We traverse all digits first and second numbers in a loop and add the result at the appropriate position.

Time complexity: O(m*n) where m and n are lengths of two numbers.
Space Complexity: O(m+n) where m and n are lengths of two numbers.

Problem 11: The k-beauty of an integer num is defined as the number of substrings of num when it is read as a string that meet the following conditions:

It has a length of k.
It is a divisor of num.
Given integers num and k, return the k-beauty of num.

Note:
Leading zeros are allowed.
0 is not a divisor of any value.
A substring is a contiguous sequence of characters in a string.

Example 1:

Input: num = 240, k = 2
Output: 2

Explanation: The following are the substrings of num of length k:
- "24" from "240": 24 is a divisor of 240.
- "40" from "240": 40 is a divisor of 240.

Therefore, the k-beauty is 2.

Example 2:

Input: num = 430043, k = 2
Output: 2

Explanation: The following are the substrings of num of length k:
- "43" from "430043": 43 is a divisor of 430043.
- "30" from "430043": 30 is not a divisor of 430043.
- "00" from "430043": 0 is not a divisor of 430043.
- "04" from "430043": 4 is not a divisor of 430043.
- "43" from "430043": 43 is a divisor of 430043.

Therefore, the k-beauty is 2.

Solution:

Code link: https://pastebin.com/xP1wZELd

Output:

```
Enter the number : 240
Enter the size of substring : 2
Desired output is: 2
```

Approach:

- Generate all possible substrings of the given number(convert it to string) of length k.
- Divide the number by each of the substrings and increment count if the remainder of the division is zero (perfectly divisible). Also, check that the integer value of the resultant substring is not zero, as division by zero is invalid.
- In the **"divisorSubstrings"** function following things are happening :
- Convert the number into String for easy traversal.
- Now take two pointers i & j for handling the window boundaries.
- run a while loop till j < string.length()
- now till we do not get the window size increment j.
- as we get the window size i.e j - i + 1 == k;
- extract the string using substr() func and then convert that string to integer using stoi() function.
- Check whether that number is divisible by k.
- If yes, increase the **"answer"**.
- return **"answer"**.

Time complexity: O(n) where n = length of the string
Space complexity: O(1)

Problem 12: Given n pairs of parentheses, write a function to generate all combinations of well-formed parentheses.

Example 1:

Input: n = 3
Output: ["((()))","(()())","(())()","()(())","()()()"]

Example 2:

Input: n = 1
Output: ["()"]

Solution:

Code link: https://pastebin.com/qRDGqKzt

Output:

```
Enter the number : 3
Desired output is: ((())) (()()) (())() ()(()) ()()()
```

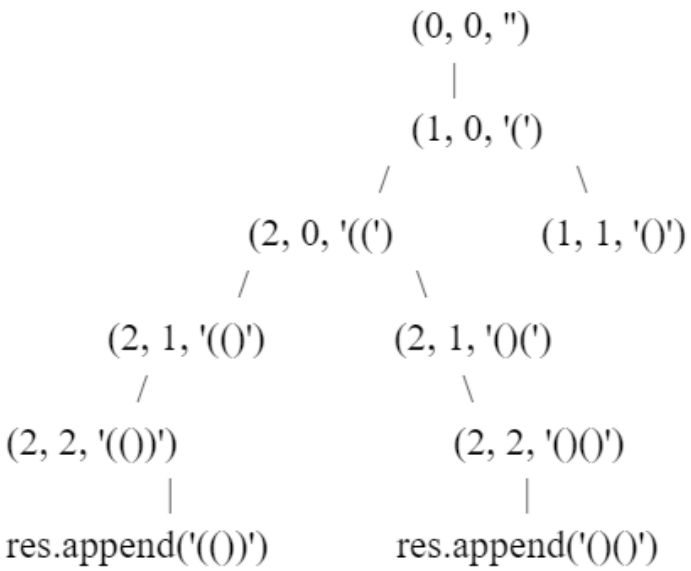
Approach:

- The idea is to add ')' only after valid '('
- We use two integer variables left & right to see how many '(' & ')' are in the current string
- **'left'**, represents how many left parentheses remain; **'right'** represents how many right parentheses remain. The remaining right parentheses should be larger than left ones.
- If **left < n** then we can add '(' to the current string
- If **right < left** then we can add ')' to the current string

Time complexity: O(2^n)

Space complexity: O(n*2^n)

For n = 2, the recursion tree will be something like this,



Problem 13: You are given a string s, where every two consecutive vertical bars '|' are grouped into a pair. In other words, the 1st and 2nd '|' make a pair, the 3rd and 4th '|' make a pair, and so forth.Return the number of '*' in s, excluding the '*' between each pair of '|'.
Note that each '|' will belong to exactly one pair.

Example 1:
Input: s = "l|*e*et|c**o|*de|"
Output: 2

Explanation: The considered characters are underlined: "l|*e*et|c**o|*de|".
The characters between the first and second '|' are excluded from the answer.
Also, the characters between the third and fourth '|' are excluded from the answer.
There are 2 asterisks considered. Therefore, we return 2.

Example 2:
Input: s = "iamprogrammer"
Output: 0

Explanation: In this example, there are no asterisks in s. Therefore, we return 0.

Example 3:
Input: s = "yo|uar|e**|b|e***au|tiful|l"
Output: 5

Explanation: The considered characters are underlined: "yo|uar|e**|b|e***au|tiful|l". There are 5 asterisks considered. Therefore, we return 5.

Constraints:
1 <= s.length <= 1000
s consists of lowercase English letters, vertical bars '|', and asterisks '*'.
s contains an even number of vertical bars '|'.

Solution:

Code link: **https://pastebin.com/Chr7JeSD**

Output:

```
Enter the string : Ph|**ysi|***csW|all|****a|h**|
Desired output is: 7
```

Approach:

- We will just count * for the odd bar string let's say 1,3,5,7...
- For example:
- "l|*e*et|c**o|*de|"
- the odd bar sequence is : 1,3 & inside of that sequence => l & c**o string exist. so total number of * inside of that string is 2.
- Similarly,
- for example
- "yo|uar|e**|b|e***au|tiful|l"
- the odd bar sequence is : 1,3,5,7 & inside of that sequence => yo , e** , e***au , l strings are existing. so total number of * inside of that string is 0+2+3=5.

Time complexity : O(n) where n = length of the string
Space complexity : O(1)

Problem 14: You are given a string s, which contains stars *.In one operation, you can:Choose a star in s.Remove the closest non-star character to its left, as well as remove the star itself. Return the string after all stars have been removed.

Note: The input will be generated such that the operation is always possible.
It can be shown that the resulting string will always be unique.

Example 1:
Input: s = "leet**cod*e"
Output: "lecoe"

Explanation: Performing the removals from left to right:
- The closest character to the 1st star is 't' in "leet**cod*e". s becomes "lee*cod*e".
- The closest character to the 2nd star is 'e' in "lee*cod*e". s becomes "lecod*e".
- The closest character to the 3rd star is 'd' in "lecod*e". s becomes "lecoe".

There are no more stars, so we return "lecoe".

Example 2:
Input: s = "erase*****"
Output: ""

Explanation: The entire string is removed, so we return an empty string.

Constraints:
1 <= s.length <= 1e5
s consists of lowercase English letters and stars *.
The operation above can be performed on s.

Solution:

Code link: **https://pastebin.com/hLJ8ZUzU**

Output:

```
Enter the string : pwb*sk*ikjh***lls
Desired output is: pwsills
```

Approach:
- Just keep inserting the characters in string “answer” and as soon as you find ‘*’ then just look into the ‘answer’ string that if it contains any character then pop one character from its back using the pop_back() function of the string.
- Keep repeating it until the last character of the string.

Time complexity: O(n) where n = length of string
Space complexity: O(n) because we have constructed a new string and in the worst case it can happen that the given string may not have even a single asterisk. So in that case a complete string will be replicated which will take space O(n).

Problem 15: Given a string s, consisting of words and spaces, return the length of the last word in the string.

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.

Example 3:
Input: s = "luffy is still joyboy"
Output: 6

Explanation: The last word is "joyboy" with length 6.

Constraints:
1 <= s.length <= 104
s consists of only English letters and spaces ' '.
There will be at least one word in s.

Solution:

Code link: **https://pastebin.com/qVni6p2v**

Output:

```
Enter the string : pwskills is a revolution
Desired output is: 10
```

Approach:

- We have simply used **stringstream** class to extract the words of the sentence given.
- Once the while loop will terminate the variable **"word"** will hold the last word of this sentence.
- We will return its size.

Time complexity: O(n)
Space complexity: O(1)

Problem 16: Given a string s, you can transform every letter individually to be lowercase or uppercase to create another string. Return a list of all possible strings we could create. Return the output in any order.

Example 1:
Input: s = "a1b2"
Output: ["a1b2","a1B2","A1b2","A1B2"]

Example 2:
Input: s = "3z4"
Output: ["3z4","3Z4"]

Constraints:
1 <= s.length <= 12
s consists of lowercase English letters, uppercase English letters, and digits.

Solution:

Code link: https://pastebin.com/sk0AaPgy

Output:

```
Enter the string : 3z4Y
Desired output is: 3z4Y 3z4y 3Z4Y 3Z4y
```

Approach:

- In the **"go"** function we have passed the parameters, the string , a vector of string holding all the possible values, an index starting from 0.
- Our first recursive call would be to pass the string on to the next index as it is without any change because we need to evaluate every single possibility.
- So we passed the parameters **s , result , i + 1**.
- The second alternative is if the **ith** character is an alphabet change its case and then pass on the new string.
- We used the function **isalpha()**, that takes a character as input and returns true if it is alphabet else false.
- If the character is in uppercase we used the function **tolower()** to convert it to lowercase and passed the new parameters **s , res , i + 1**. Here the note point is the string passed is already a changed version of the original string.
- So we need to make sure that further calls should be made or further changes should be made on the original string only, so to reverse the effect of change we simply put back the character with the original case back in the same position.
- If the character is in lowercase we used the function **toupper()** to convert it to uppercase and again passed the same parameters.

Time complexity: O(2^n) because in the worst case there can be a string of length n where every character is an alphabet so each character has 2 possibilities, one in uppercase and second in lowercase.
Space complexity: O(n*2^n) in the worst case there can be 2^n different strings all of size n.