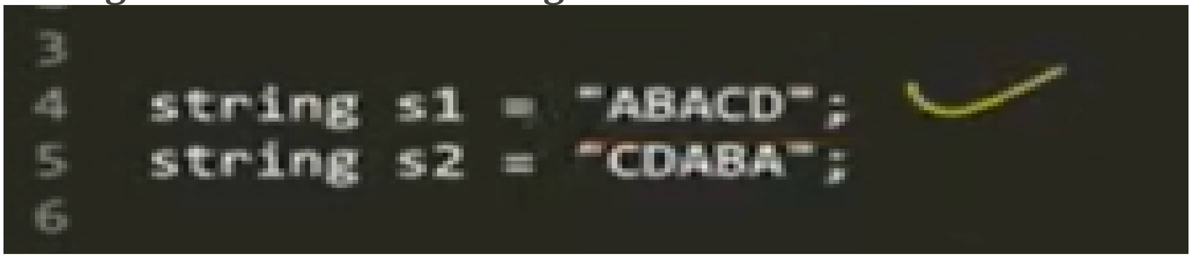
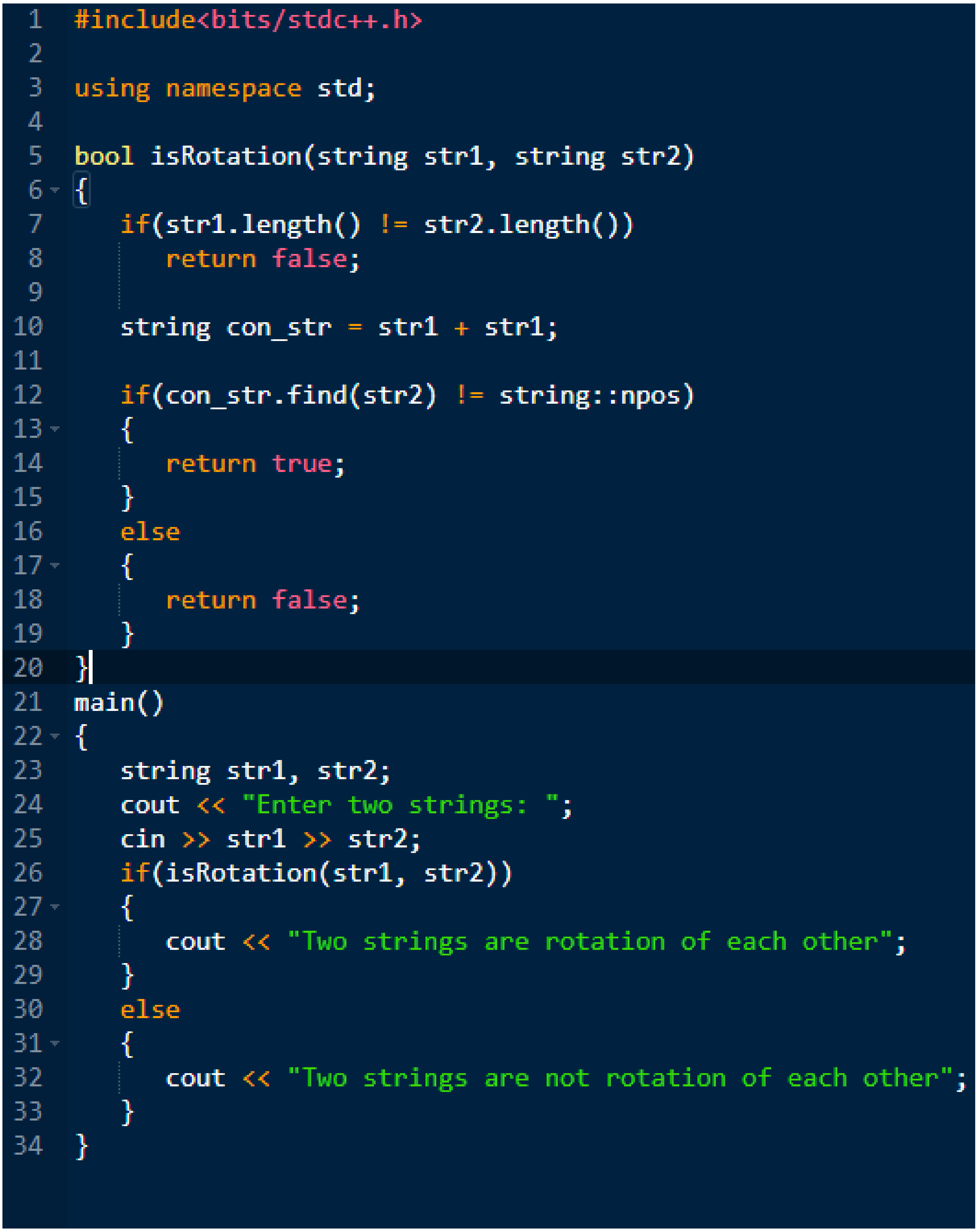
STRING

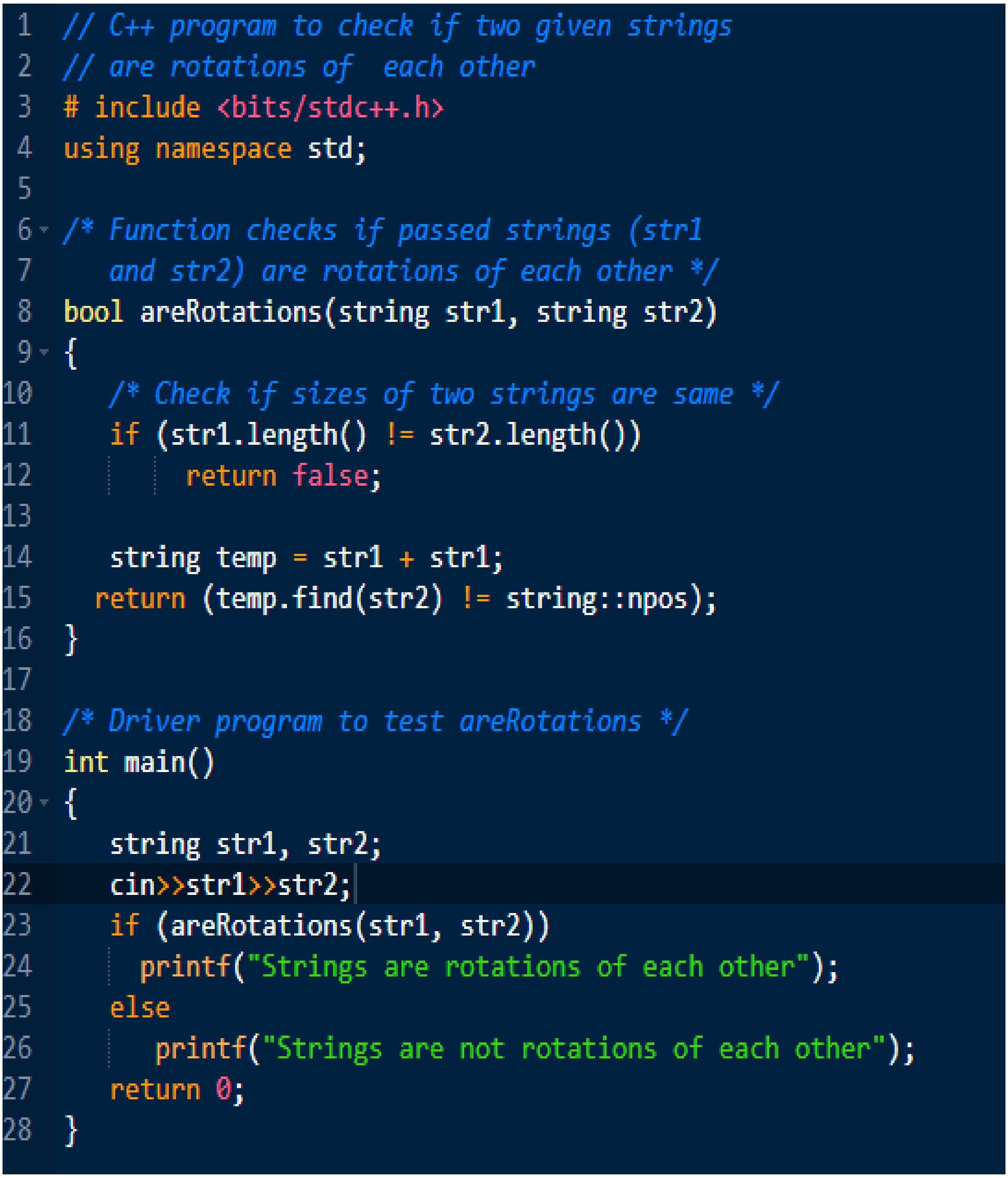
1-Program to check that strings are rotation of each other or not-



```
string s1 = "ABACD";
string s2 = "CDABA";
```

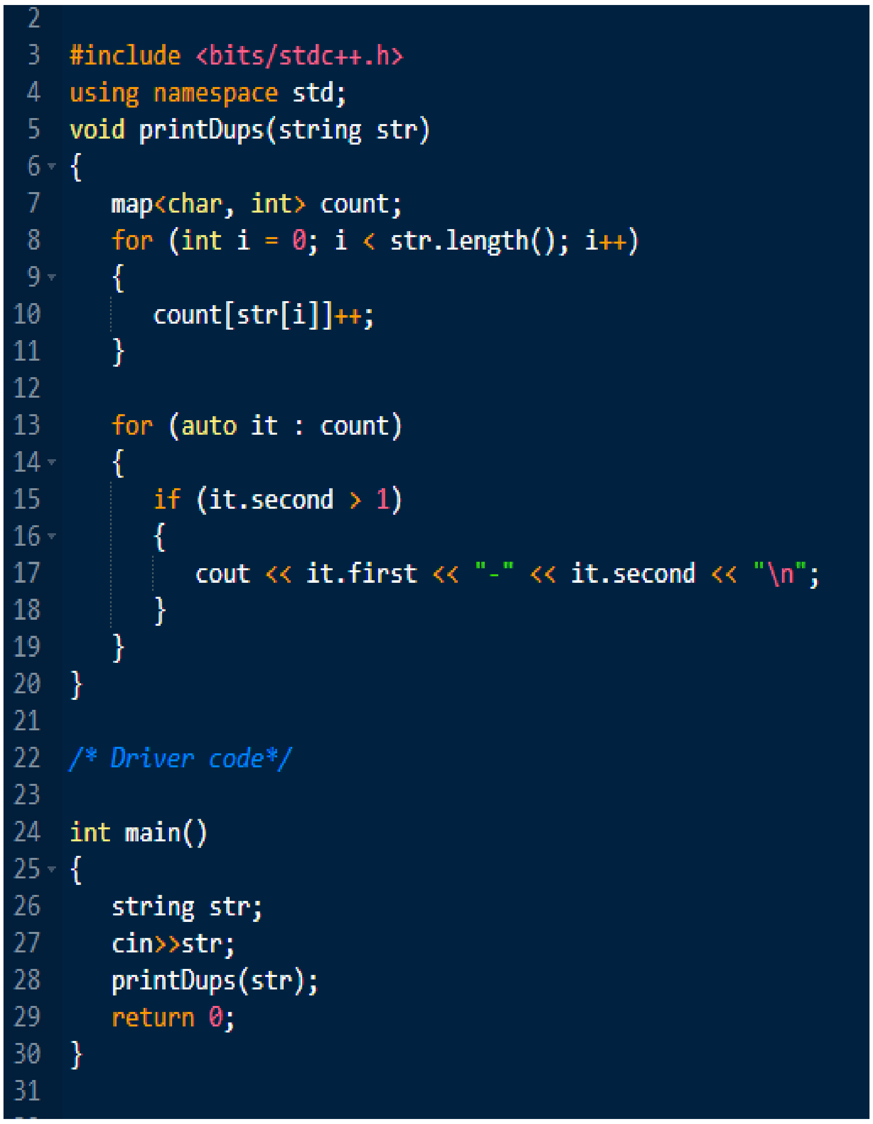


```cpp
#include<bits/stdc++.h>

using namespace std;

bool isRotation(string str1, string str2)
{
    if(str1.length() != str2.length())
        return false;

    string con_str = str1 + str1;

    if(con_str.find(str2) != string::npos)
    {
        return true;
    }
    else
    {
        return false;
    }
}
main()
{
    string str1, str2;
    cout << "Enter two strings: ";
    cin >> str1 >> str2;
    if(isRotation(str1, str2))
    {
        cout << "Two strings are rotation of each other";
    }
    else
    {
        cout << "Two strings are not rotation of each other";
    }
}
```



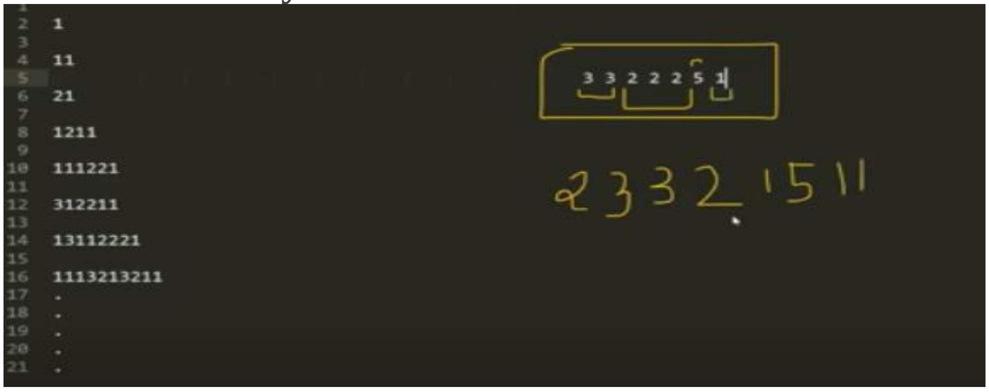
```cpp
// C++ program to check if two given strings
// are rotations of  each other
# include <bits/stdc++.h>
using namespace std;

/* Function checks if passed strings (str1
   and str2) are rotations of each other */
bool areRotations(string str1, string str2)
{
   /* Check if sizes of two strings are same */
   if (str1.length() != str2.length())
        return false;

   string temp = str1 + str1;
  return (temp.find(str2) != string::npos);
}

/* Driver program to test areRotations */
int main()
{
   string str1, str2;
   cin>>str1>>str2;
   if (areRotations(str1, str2))
     printf("Strings are rotations of each other");
   else
      printf("Strings are not rotations of each other");
   return 0;
}
```

2-Find all the duplicate characters from a given string-

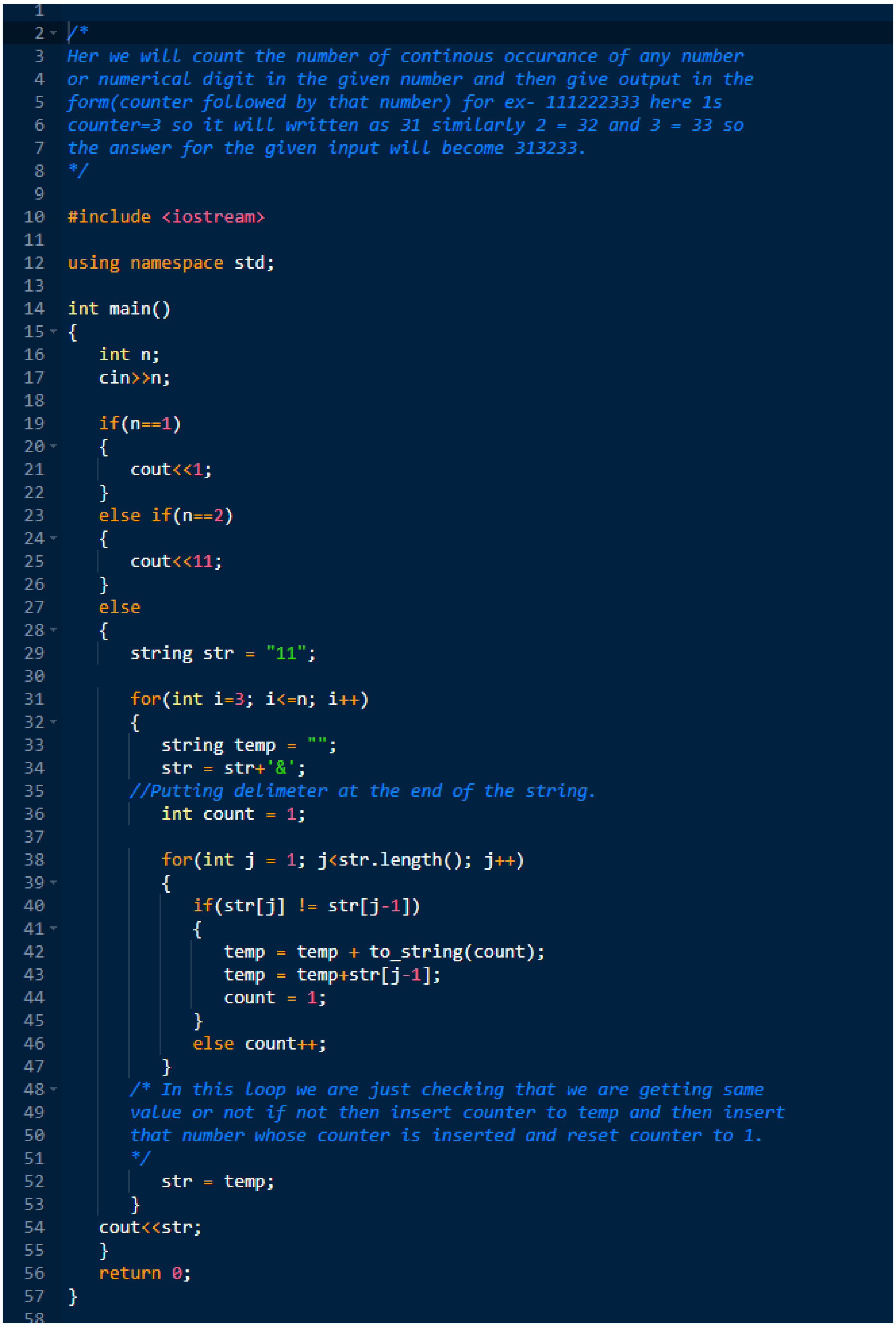


```cpp

#include <bits/stdc++.h>
using namespace std;
void printDups(string str)
{
    map<char, int> count;
    for (int i = 0; i < str.length(); i++)
    {
        count[str[i]]++;
    }

    for (auto it : count)
    {
        if (it.second > 1)
        {
            cout << it.first << "-" << it.second << "\n";
        }
    }
}

/* Driver code*/

int main()
{
    string str;
    cin>>str;
    printDups(str);
    return 0;
}

```

3-Count and say-

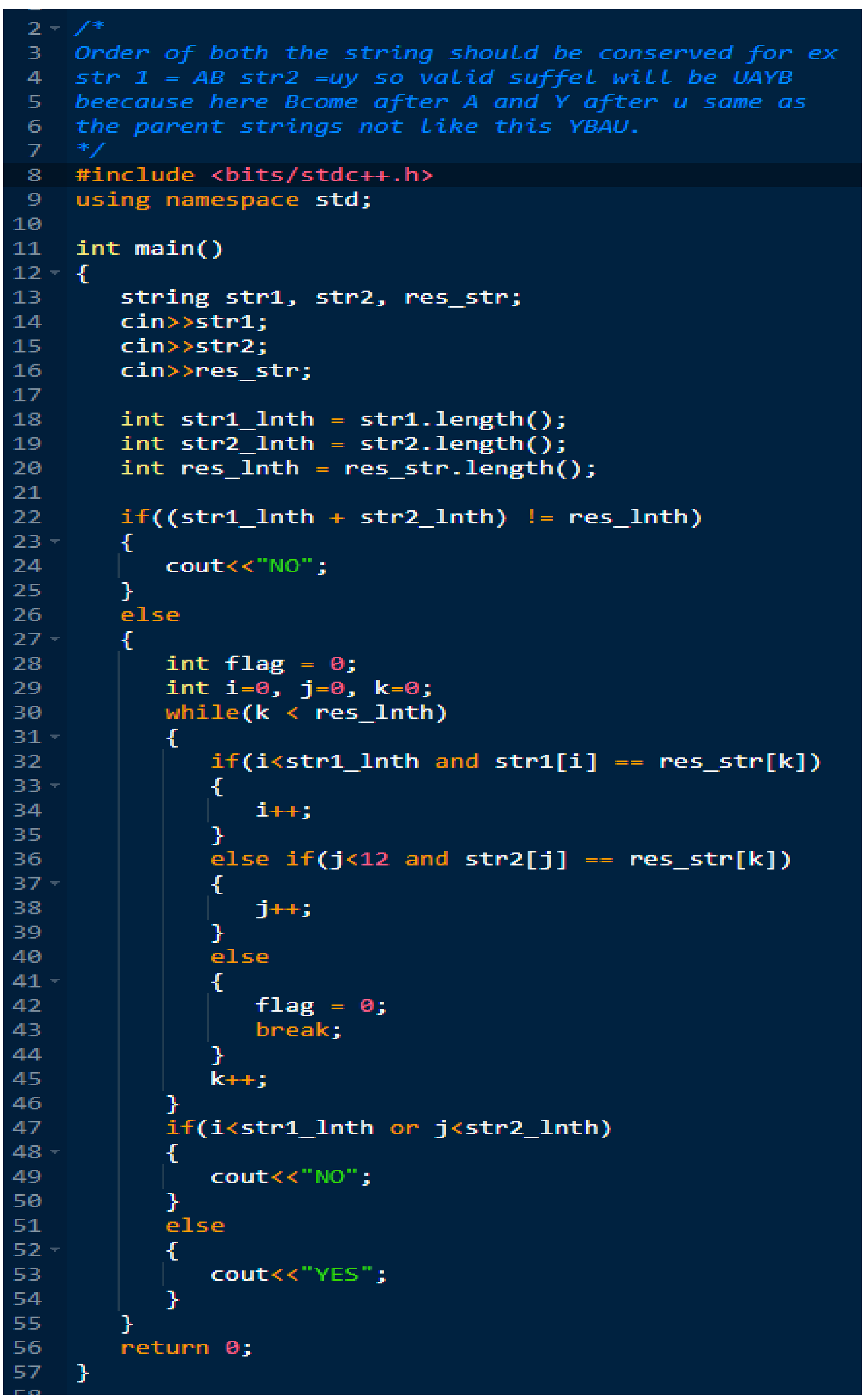


```
1

11

21

1211

111221

312211

13112221

1113213211
.
.
.
.
.
```



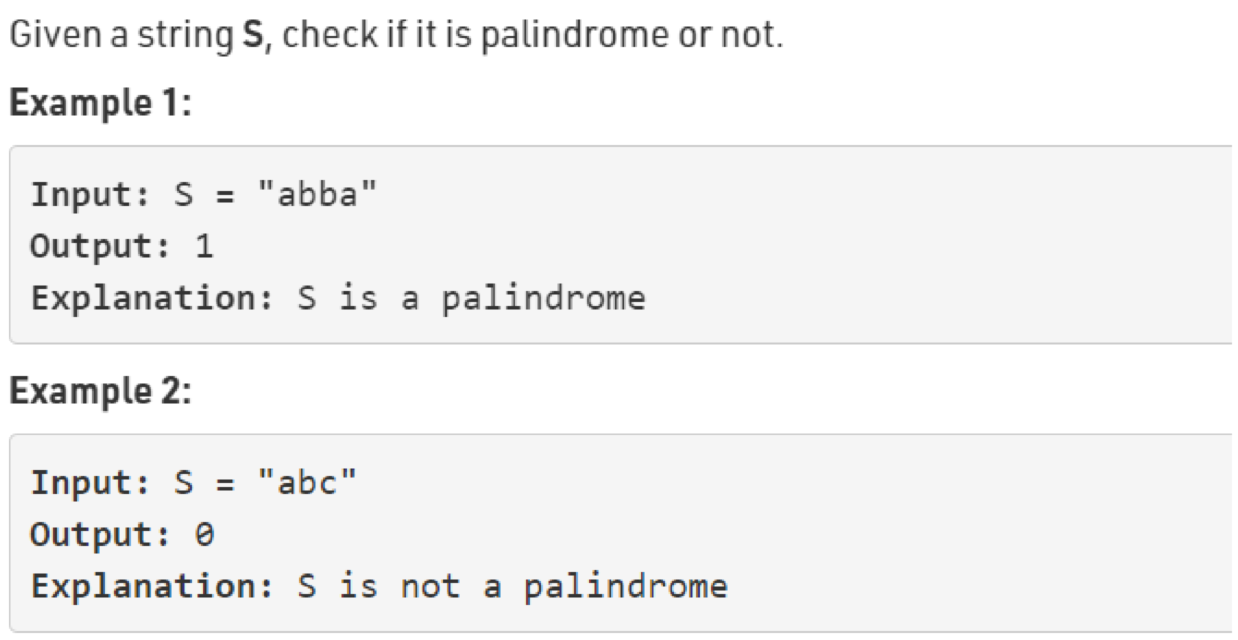
```cpp

/*
Her we will count the number of continous occurance of any number
or numerical digit in the given number and then give output in the
form(counter followed by that number) for ex- 111222333 here 1s
counter=3 so it will written as 31 similarly 2 = 32 and 3 = 33 so
the answer for the given input will become 313233.
*/

#include <iostream>

using namespace std;

int main()
{
    int n;
    cin>>n;

    if(n==1)
    {
        cout<<1;
    }
    else if(n==2)
    {
        cout<<11;
    }
    else
    {
        string str = "11";

        for(int i=3; i<=n; i++)
        {
            string temp = "";
            str = str+'&';
        //Putting delimeter at the end of the string.
            int count = 1;

            for(int j = 1; j<str.length(); j++)
            {
                if(str[j] != str[j-1])
                {
                    temp = temp + to_string(count);
                    temp = temp+str[j-1];
                    count = 1;
                }
                else count++;
            }
        /* In this loop we are just checking that we are getting same
        value or not if not then insert counter to temp and then insert
        that number whose counter is inserted and reset counter to 1.
        */
            str = temp;
        }
    cout<<str;
    }
    return 0;
}

```

4- Check if a str. is a valid shuffle of two distinct str.



```cpp
/*
Order of both the string should be conserved for ex
str 1 = AB str2 =uy so valid suffel will be UAYB
beecause here Bcome after A and Y after u same as
the parent strings not like this YBAU.
*/
#include <bits/stdc++.h>
using namespace std;

int main()
{
    string str1, str2, res_str;
    cin>>str1;
    cin>>str2;
    cin>>res_str;

    int str1_lnth = str1.length();
    int str2_lnth = str2.length();
    int res_lnth = res_str.length();

    if((str1_lnth + str2_lnth) != res_lnth)
    {
        cout<<"NO";
    }
    else
    {
        int flag = 0;
        int i=0, j=0, k=0;
        while(k < res_lnth)
        {
            if(i<str1_lnth and str1[i] == res_str[k])
            {
                i++;
            }
            else if(j<12 and str2[j] == res_str[k])
            {
                j++;
            }
            else
            {
                flag = 0;
                break;
            }
            k++;
        }
        if(i<str1_lnth or j<str2_lnth)
        {
            cout<<"NO";
        }
        else
        {
            cout<<"YES";
        }
    }
    return 0;
}
```

5-Palindrome string-



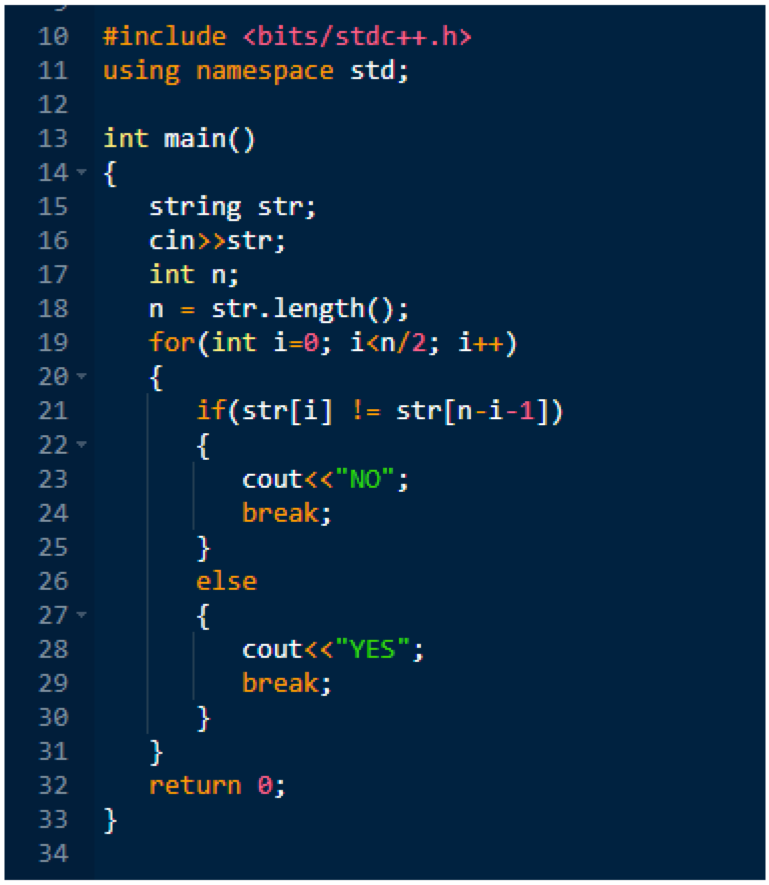
Given a string **S**, check if it is palindrome or not.

Example 1:

```
Input: S = "abba"
Output: 1
Explanation: S is a palindrome
```

Example 2:

```
Input: S = "abc"
Output: 0
Explanation: S is not a palindrome
```



```cpp
#include <bits/stdc++.h>
using namespace std;

int main()
{
    string str;
    cin>>str;
    int n;
    n = str.length();
    for(int i=0; i<n/2; i++)
    {
        if(str[i] != str[n-i-1])
        {
            cout<<"NO";
            break;
        }
        else
        {
            cout<<"YES";
            break;
        }
    }
    return 0;
}

```

6-Pallindromic substring in a string-

Longest Palindrome in a String

Medium   Accuracy: 49.2%   Submissions: 31377   Points: 4

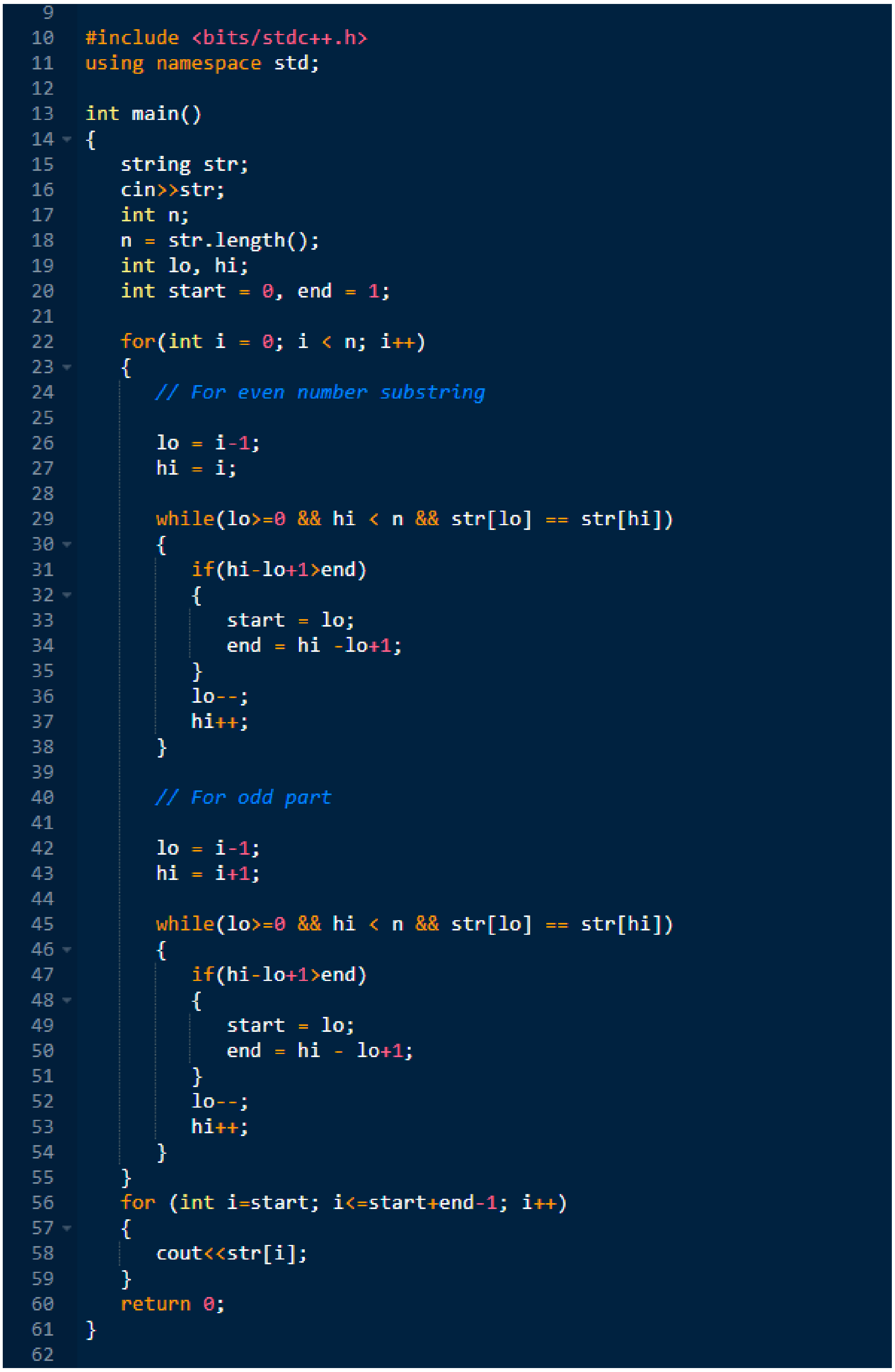
Given a string S, find the longest palindromic substring in S. **Substring of string S:** S[i . . . . j] where 0 ≤ i ≤ j < len(S). **Palindrome string:** A string which reads the same backwards. More formally, S is palindrome if reverse(S) = S. **Incase of conflict**, return the substring which occurs first (with the least starting index).

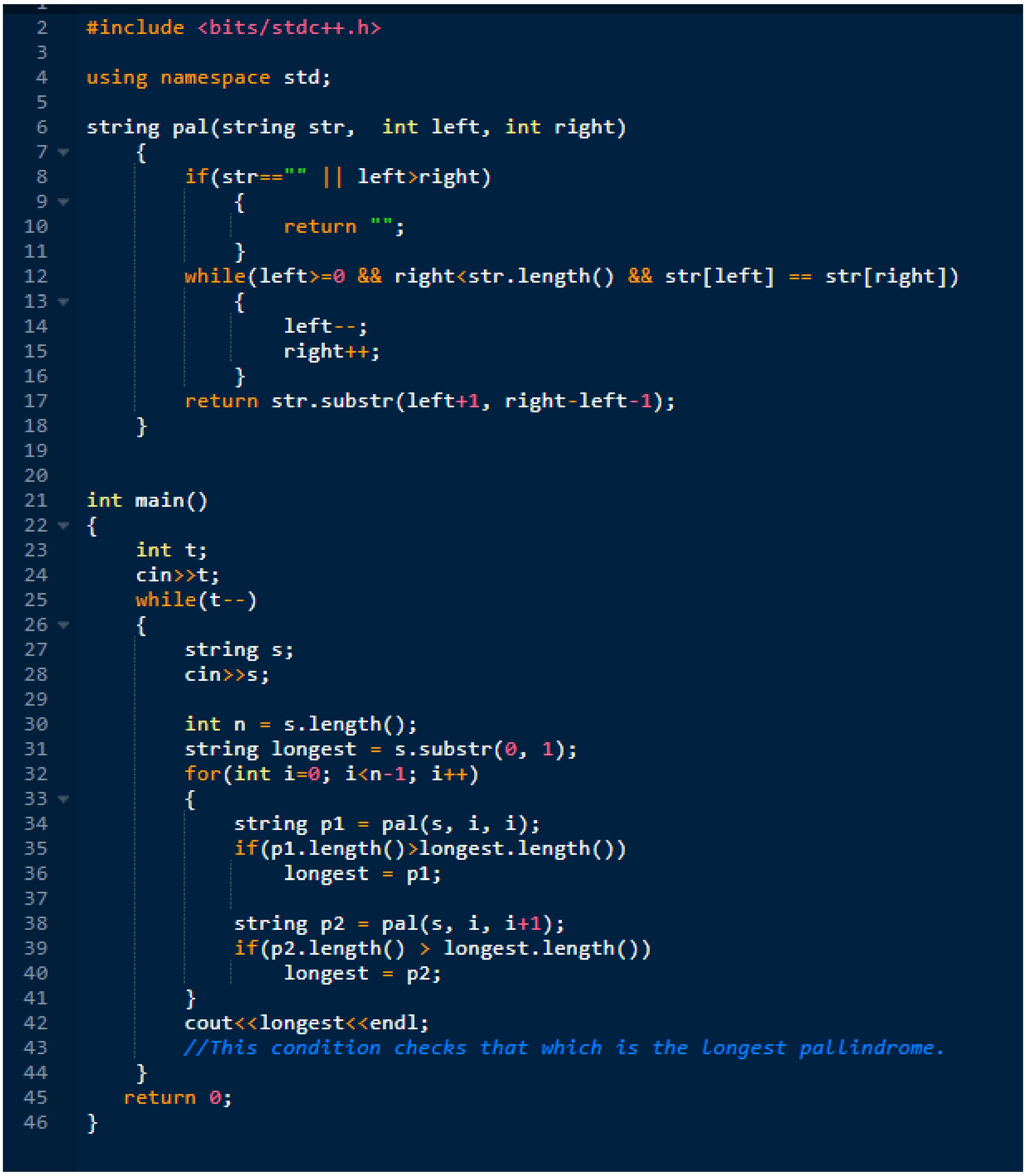
Example 1:

```
Input:
S = "aaaabbaa"
Output: aabbaa
Explanation: The longest Palindromic
substring is "aabbaa".
```

Example 2:

```
Input:
S = "abc"
Output: a
Explanation: "a", "b" and "c" are the
longest palindromes with same length.
The result is the one with the least
starting index.
```


```cpp

#include <bits/stdc++.h>
using namespace std;

int main()
{
    string str;
    cin>>str;
    int n;
    n = str.length();
    int lo, hi;
    int start = 0, end = 1;

    for(int i = 0; i < n; i++)
    {
        // For even number substring

        lo = i-1;
        hi = i;

        while(lo>=0 && hi < n && str[lo] == str[hi])
        {
            if(hi-lo+1>end)
            {
                start = lo;
                end = hi -lo+1;
            }
            lo--;
            hi++;
        }

        // For odd part

        lo = i-1;
        hi = i+1;

        while(lo>=0 && hi < n && str[lo] == str[hi])
        {
            if(hi-lo+1>end)
            {
                start = lo;
                end = hi - lo+1;
            }
            lo--;
            hi++;
        }
    }
    for (int i=start; i<=start+end-1; i++)
    {
        cout<<str[i];
    }
    return 0;
}

```



```cpp
#include <bits/stdc++.h>

using namespace std;

string pal(string str,  int left, int right)
    {
        if(str=="" || left>right)
            {
                return "";
            }
        while(left>=0 && right<str.length() && str[left] == str[right])
            {
                left--;
                right++;
            }
        return str.substr(left+1, right-left-1);
    }


int main()
{
    int t;
    cin>>t;
    while(t--)
    {
        string s;
        cin>>s;

        int n = s.length();
        string longest = s.substr(0, 1);
        for(int i=0; i<n-1; i++)
        {
            string p1 = pal(s, i, i);
            if(p1.length()>longest.length())
                longest = p1;

            string p2 = pal(s, i, i+1);
            if(p2.length() > longest.length())
                longest = p2;
        }
        cout<<longest<<endl;
        //This condition checks that which is the longest pallindrome.
    }
   return 0;
}
```

7-Split binary str. in to substrs with equal number of 0s and 1s-

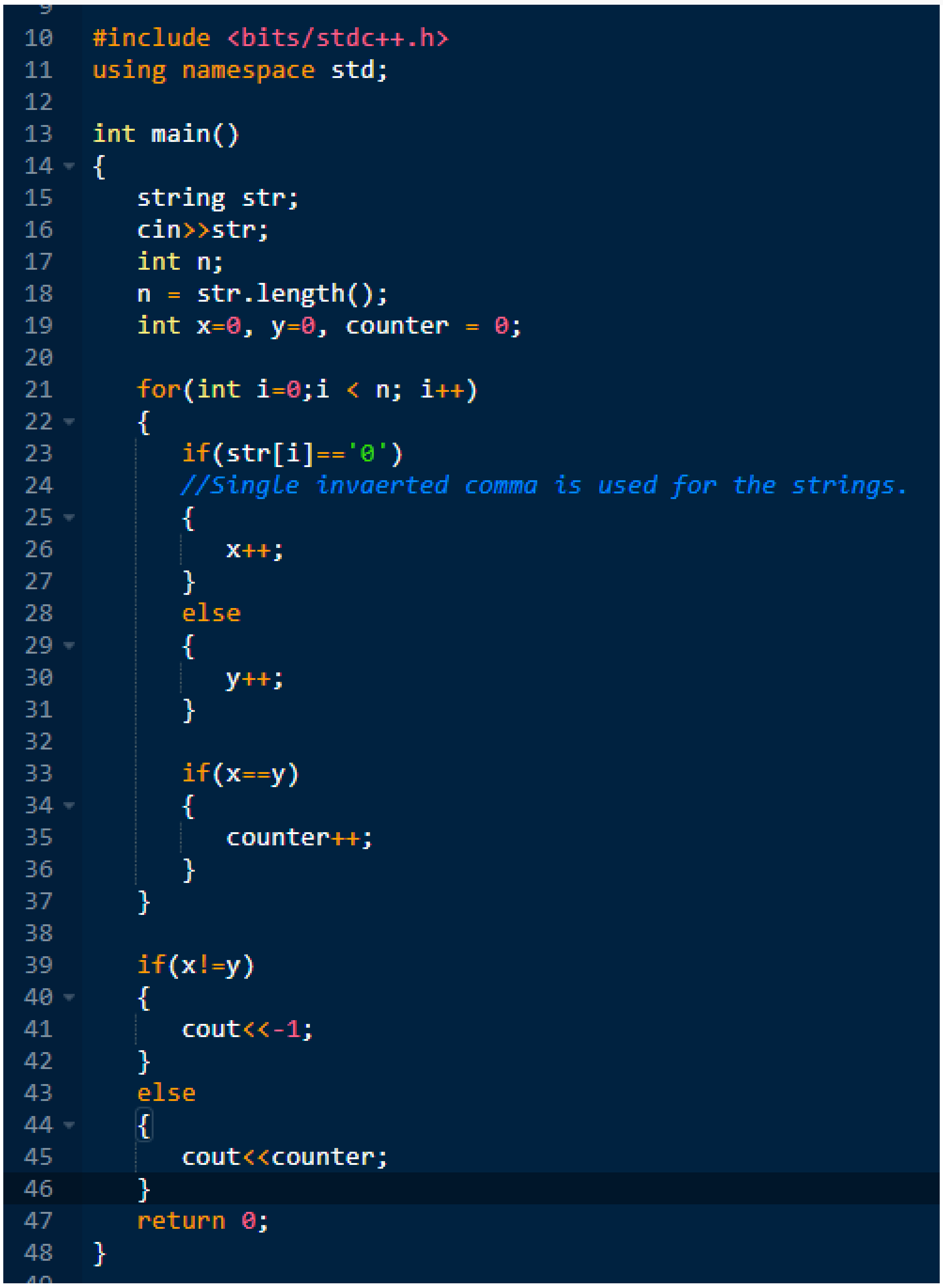
Given a binary string **str** of length **N**, the task is to find the maximum count of consecutive substrings **str** can be divided into such that all the substrings are balanced i.e. they have equal number of **0s** and **1s**. If it is not possible to split **str** satisfying the conditions then print **-1**.

Example:

Input: *str = "0100110101"*
Output: *4*
The required substrings are "01", "0011", "01" and "01".
Input: *str = "0111100010"*
Output: *3*

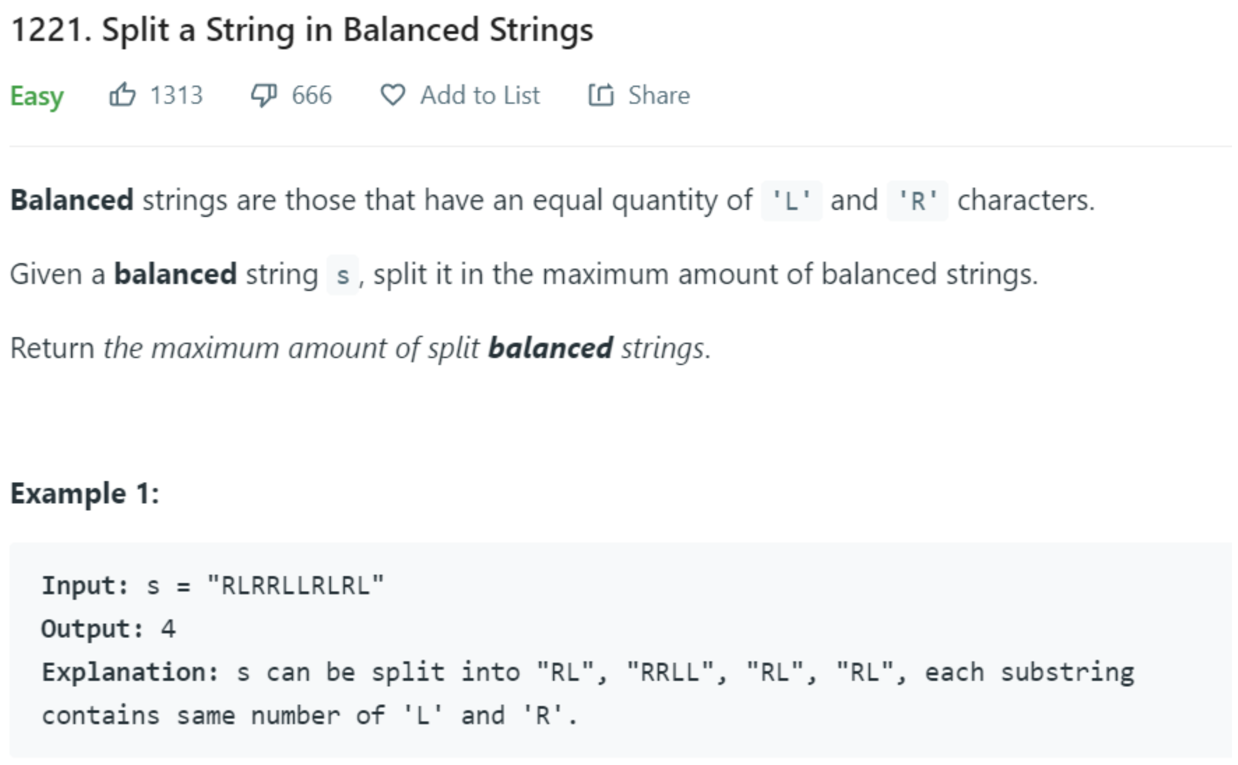
Input: *str = "0000000000"*

Output: *-1*



```
#include <bits/stdc++.h>
using namespace std;

int main()
{
    string str;
    cin>>str;
    int n;
    n = str.length();
    int x=0, y=0, counter = 0;

    for(int i=0;i < n; i++)
    {
        if(str[i]=='0')
        //Single invaerted comma is used for the strings.
        {
            x++;
        }
        else
        {
            y++;
        }

        if(x==y)
        {
            counter++;
        }
    }

    if(x!=y)
    {
        cout<<-1;
    }
    else
    {
        cout<<counter;
    }
    return 0;
}
```

OR



1221. Split a String in Balanced Strings

Easy 1313 666 Add to List Share

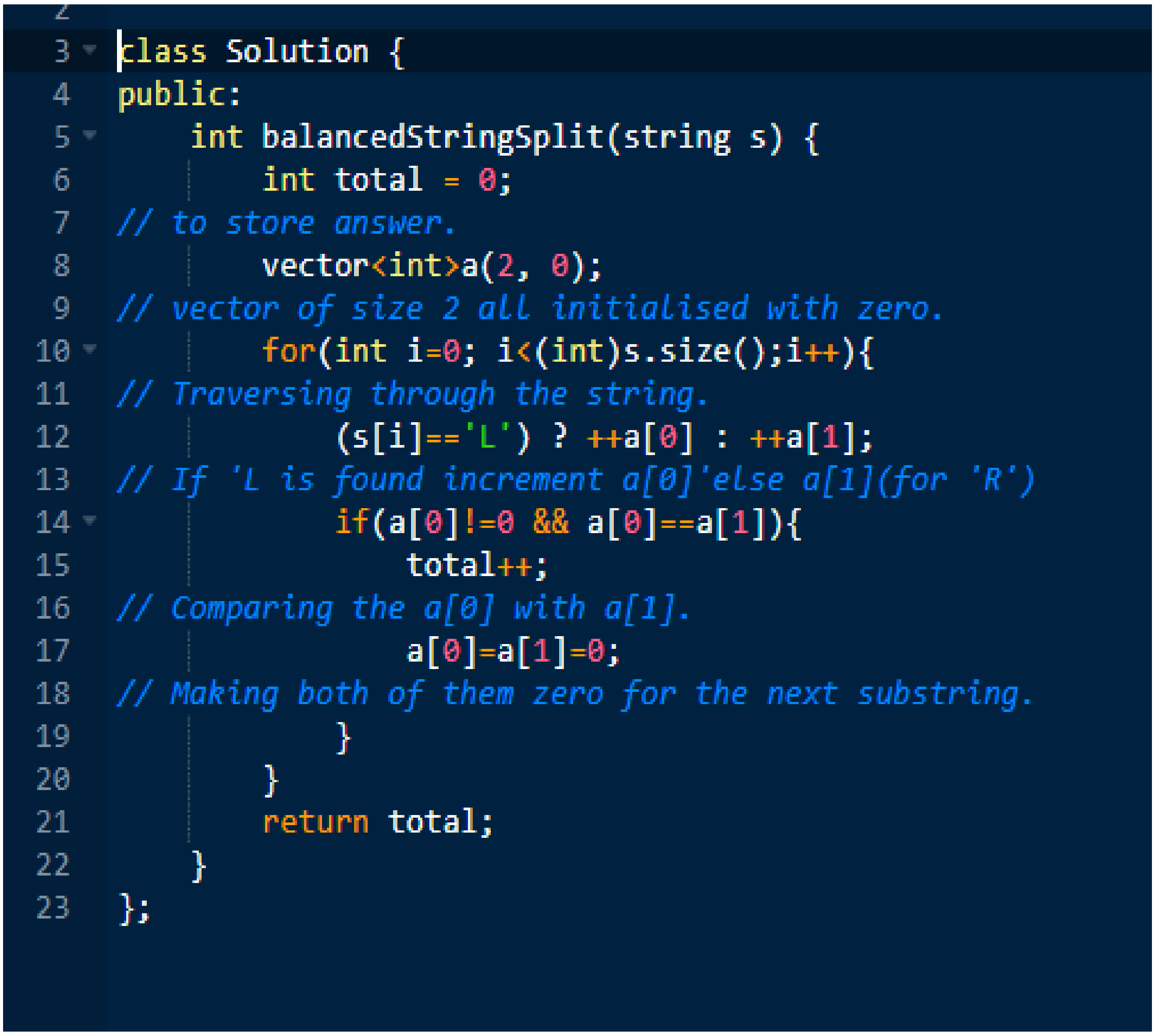
Balanced strings are those that have an equal quantity of `'L'` and `'R'` characters.

Given a **balanced** string `s`, split it in the maximum amount of balanced strings.

Return *the maximum amount of split **balanced** strings.*

Example 1:

```
Input: s = "RLRRLLRLRL"
Output: 4
Explanation: s can be split into "RL", "RRLL", "RL", "RL", each substring
contains same number of 'L' and 'R'.
```



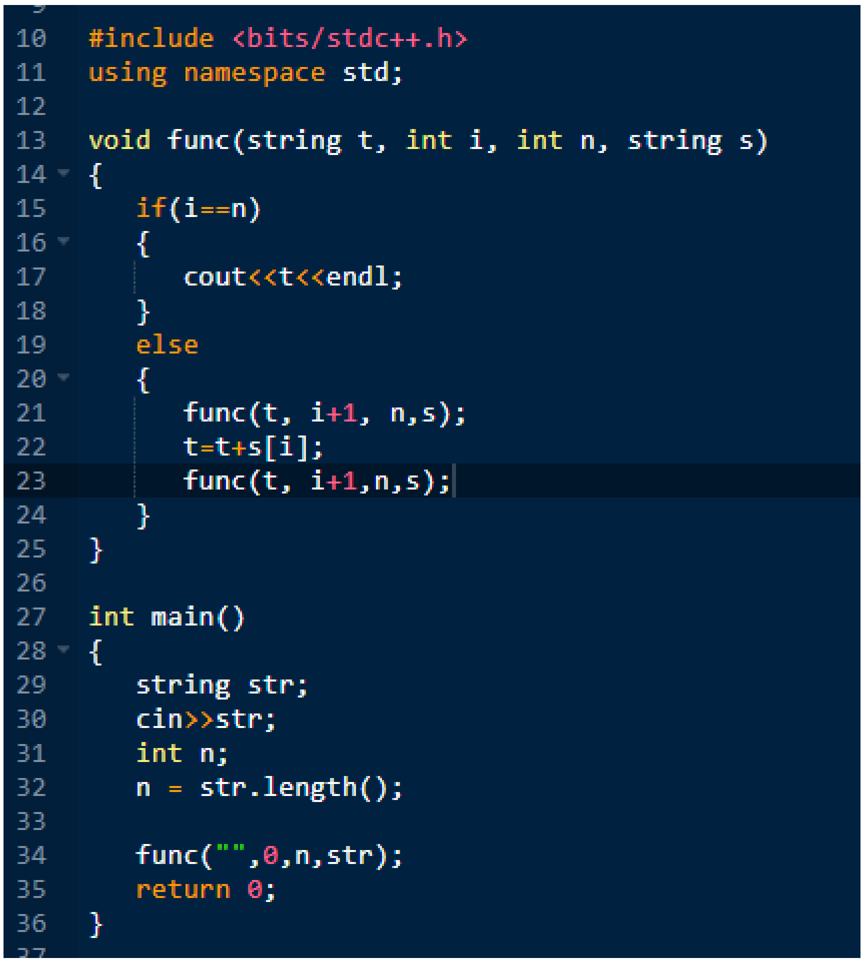
```
class Solution {
public:
    int balancedStringSplit(string s) {
        int total = 0;
// to store answer.
        vector<int>a(2, 0);
// vector of size 2 all initialised with zero.
        for(int i=0; i<(int)s.size();i++){
// Traversing through the string.
            (s[i]=='L') ? ++a[0] : ++a[1];
// If 'L is found increment a[0]'else a[1](for 'R')
            if(a[0]!=0 && a[0]==a[1]){
                total++;
// Comparing the a[0] with a[1].
                a[0]=a[1]=0;
// Making both of them zero for the next substring.
            }
        }
        return total;
    }
};
```

8-Print all subsequence of a string-

Given a string, we have to find out all subsequences of it. A String is a subsequence of a given String, that is generated by deleting some character of a given string without changing its order.

Examples:

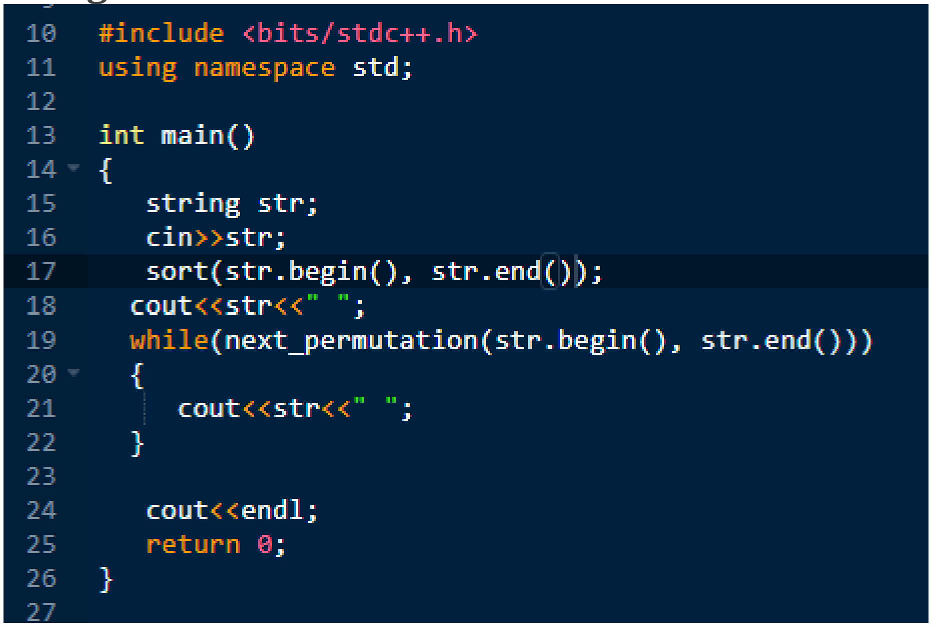
```
Input : abc
Output : a, b, c, ab, bc, ac, abc

Input : aaa
Output : a, aa, aaa
```



```cpp
#include <bits/stdc++.h>
using namespace std;

void func(string t, int i, int n, string s)
{
    if(i==n)
    {
        cout<<t<<endl;
    }
    else
    {
        func(t, i+1, n,s);
        t=t+s[i];
        func(t, i+1,n,s);
    }
}

int main()
{
    string str;
    cin>>str;
    int n;
    n = str.length();

    func("",0,n,str);
    return 0;
}
```

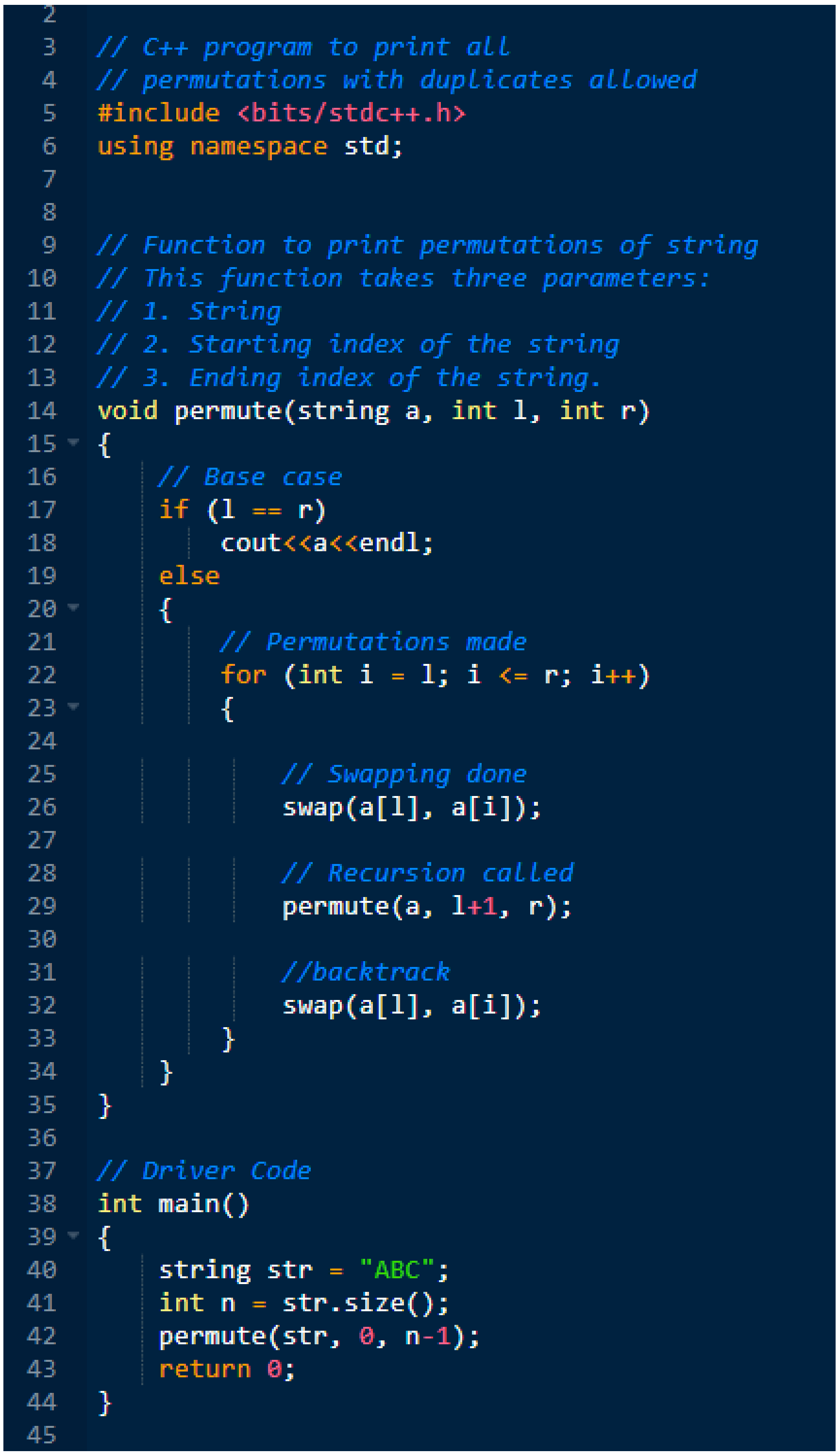
9-Find all the permutations of the given string-

Using stl-



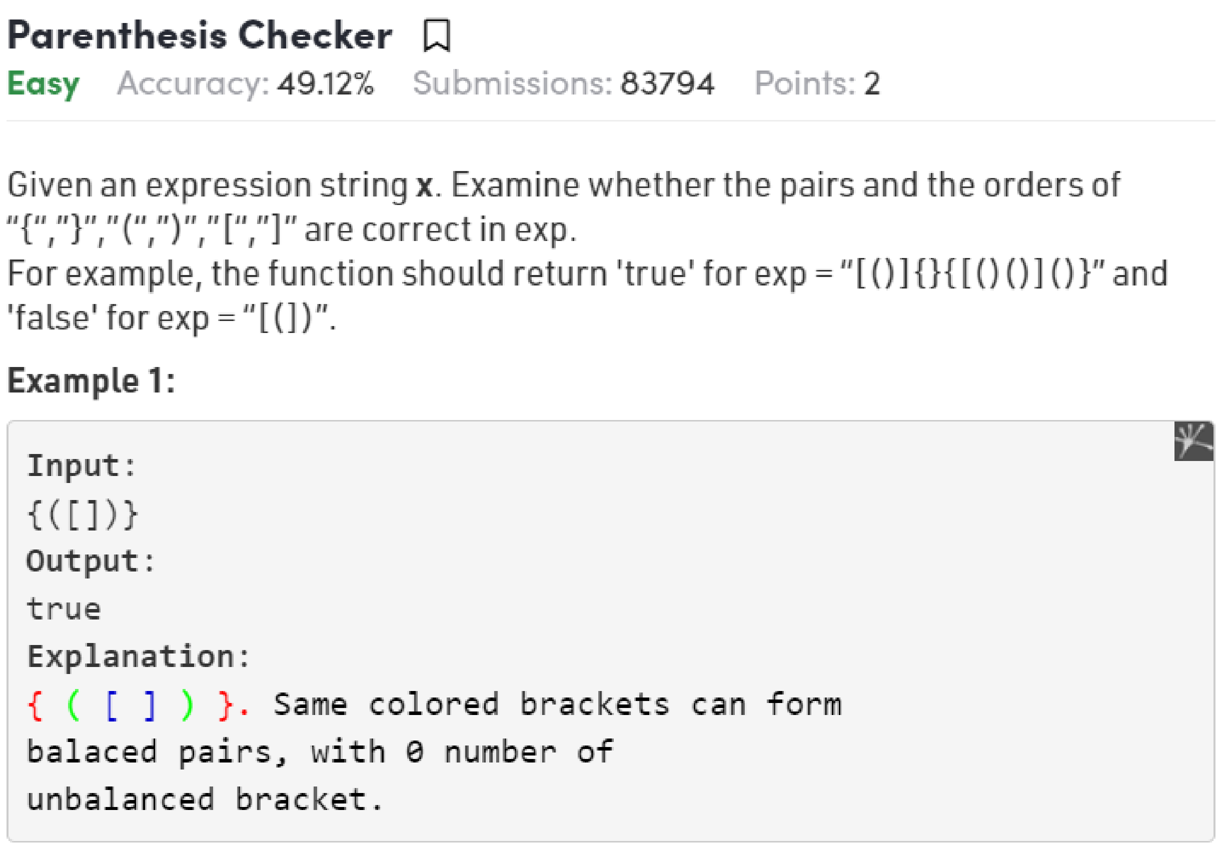
```cpp
#include <bits/stdc++.h>
using namespace std;

int main()
{
    string str;
    cin>>str;
    sort(str.begin(), str.end());
   cout<<str<<" ";
   while(next_permutation(str.begin(), str.end()))
   {
      cout<<str<<" ";
   }

    cout<<endl;
    return 0;
}
```

Recursive approach-



```cpp

// C++ program to print all
// permutations with duplicates allowed
#include <bits/stdc++.h>
using namespace std;


// Function to print permutations of string
// This function takes three parameters:
// 1. String
// 2. Starting index of the string
// 3. Ending index of the string.
void permute(string a, int l, int r)
{
    // Base case
    if (l == r)
        cout<<a<<endl;
    else
    {
        // Permutations made
        for (int i = l; i <= r; i++)
        {

            // Swapping done
            swap(a[l], a[i]);

            // Recursion called
            permute(a, l+1, r);

            //backtrack
            swap(a[l], a[i]);
        }
    }
}

// Driver Code
int main()
{
    string str = "ABC";
    int n = str.size();
    permute(str, 0, n-1);
    return 0;
}

```

10-Valid parenthesis-



Parenthesis Checker

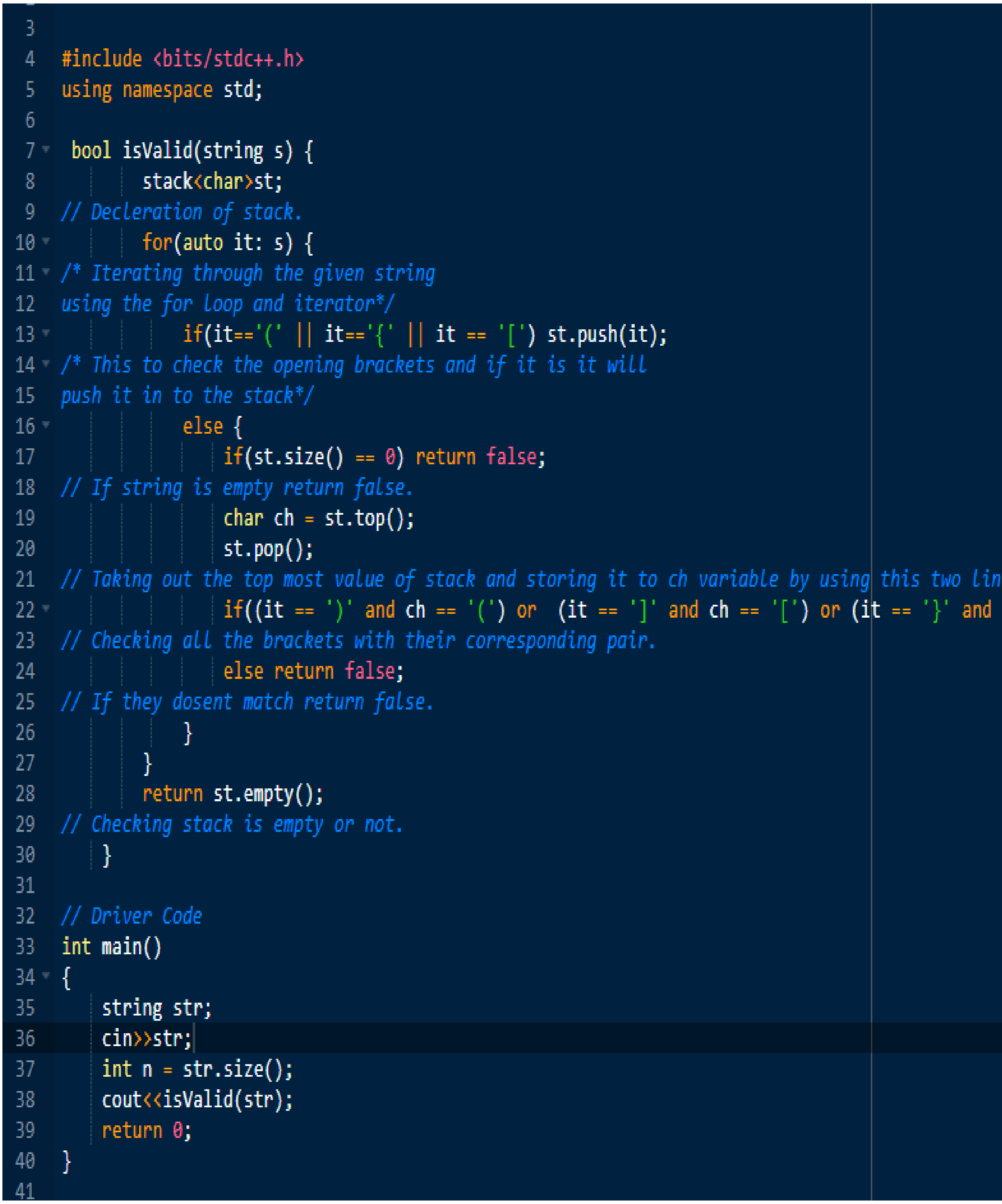
Easy   Accuracy: 49.12%   Submissions: 83794   Points: 2

Given an expression string **x**. Examine whether the pairs and the orders of “{“,”}”,”(“,”)”,”[“,”]” are correct in exp.
For example, the function should return 'true' for exp = “[()]{}{[()()]()}” and 'false' for exp = “[(])”.

Example 1:

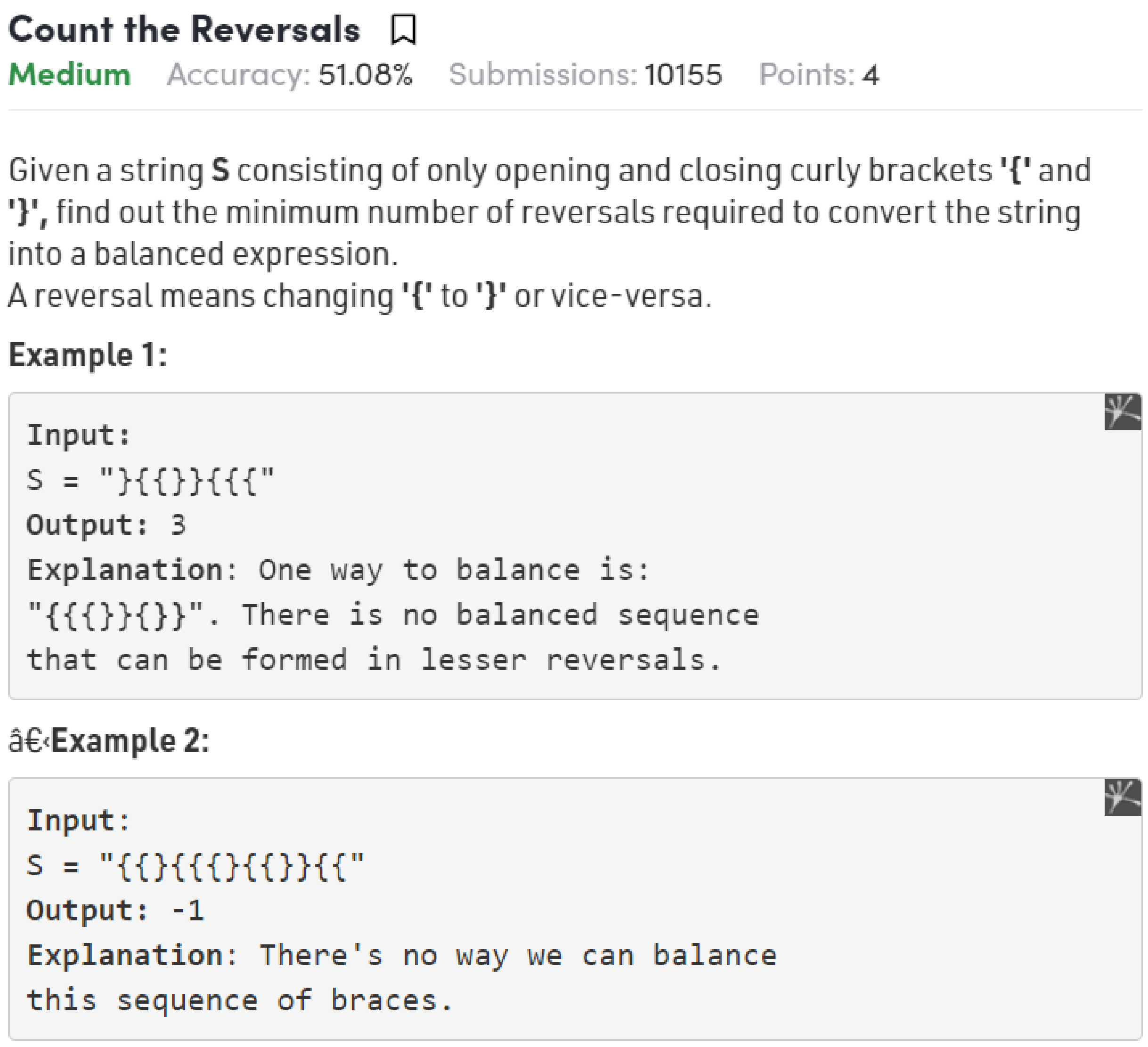
```
Input:
{([])}
Output:
true
Explanation:
{ ( [ ] ) }. Same colored brackets can form
balaced pairs, with 0 number of
unbalanced bracket.
```

USING STACK-



```
#include <bits/stdc++.h>
using namespace std;

 bool isValid(string s) {
        stack<char>st;
// Decleration of stack.
        for(auto it: s) {
/* Iterating through the given string
using the for loop and iterator*/
            if(it=='(' || it=='{' || it == '[') st.push(it);
/* This to check the opening brackets and if it is it will
push it in to the stack*/
            else {
                if(st.size() == 0) return false;
// If string is empty return false.
                char ch = st.top();
                st.pop();
// Taking out the top most value of stack and storing it to ch variable by using this two lin
                if((it == ')' and ch == '(') or  (it == ']' and ch == '[') or (it == '}' and
// Checking all the brackets with their corresponding pair.
                else return false;
// If they dosent match return false.
            }
        }
        return st.empty();
// Checking stack is empty or not.
    }

// Driver Code
int main()
{
    string str;
    cin>>str;
    int n = str.size();
    cout<<isValid(str);
    return 0;
}
```

OR



Count the Reversals

Medium Accuracy: 51.08% Submissions: 10155 Points: 4

Given a string **S** consisting of only opening and closing curly brackets **'{'** and **'}',** find out the minimum number of reversals required to convert the string into a balanced expression.
A reversal means changing **'{'** to **'}'** or vice-versa.

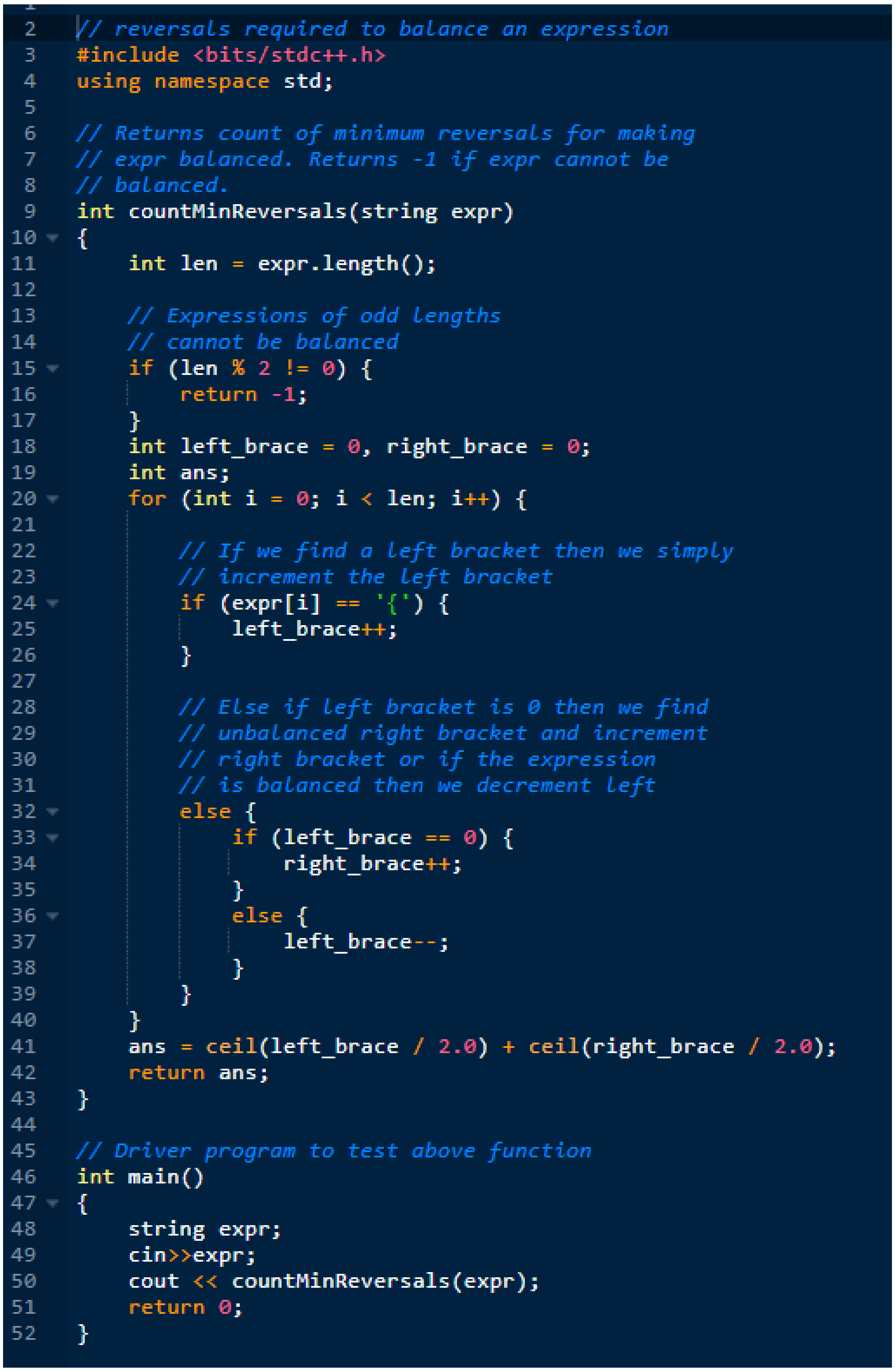
Example 1:

```
Input:
S = "}{{}}{{{"
Output: 3
Explanation: One way to balance is:
"{{{}}{}}". There is no balanced sequence
that can be formed in lesser reversals.
```

â€‹**Example 2:**

```
Input:
S = "{{}{{{}{{}}{{"
Output: -1
Explanation: There's no way we can balance
this sequence of braces.
```

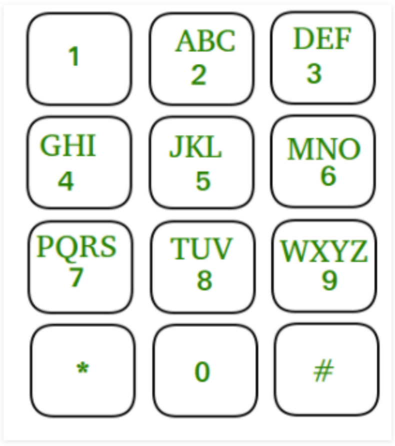
O(1) space solution can be used in both que-



```cpp
// reversals required to balance an expression
#include <bits/stdc++.h>
using namespace std;

// Returns count of minimum reversals for making
// expr balanced. Returns -1 if expr cannot be
// balanced.
int countMinReversals(string expr)
{
    int len = expr.length();

    // Expressions of odd lengths
    // cannot be balanced
    if (len % 2 != 0) {
        return -1;
    }
    int left_brace = 0, right_brace = 0;
    int ans;
    for (int i = 0; i < len; i++) {

        // If we find a left bracket then we simply
        // increment the left bracket
        if (expr[i] == '{') {
            left_brace++;
        }

        // Else if left bracket is 0 then we find
        // unbalanced right bracket and increment
        // right bracket or if the expression
        // is balanced then we decrement left
        else {
            if (left_brace == 0) {
                right_brace++;
            }
            else {
                left_brace--;
            }
        }
    }
    ans = ceil(left_brace / 2.0) + ceil(right_brace / 2.0);
    return ans;
}

// Driver program to test above function
int main()
{
    string expr;
    cin>>expr;
    cout << countMinReversals(expr);
    return 0;
}
```

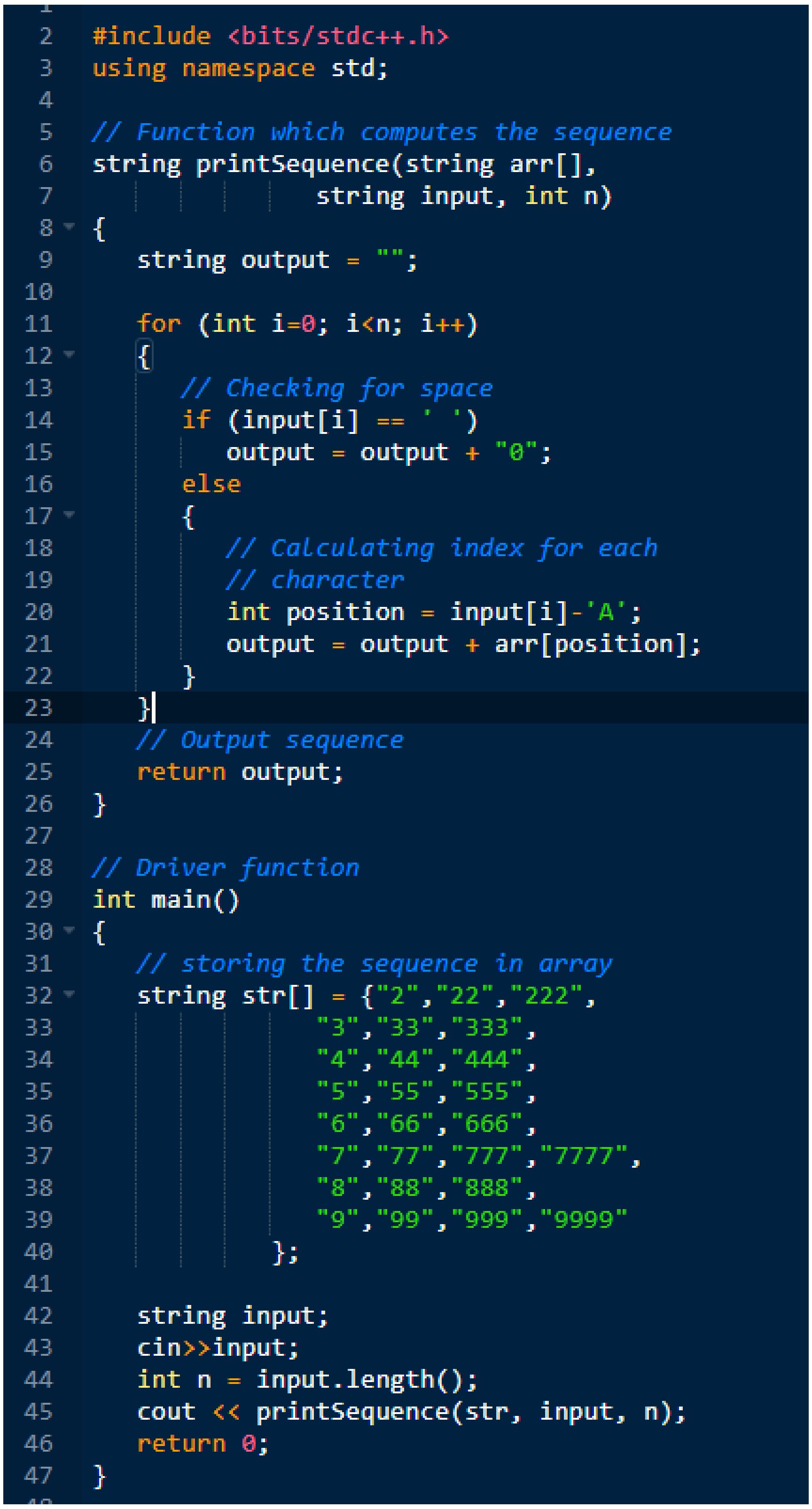
11-Convert a sentence into its equivalent mobile numeric keypad sequence-

Given a sentence in the form of a string, convert it into its equivalent mobile numeric keypad sequence.



Examples :

```
Input : GEEKSFORGEEKS
Output : 4333355777733366677743333557777
For obtaining a number, we need to press a
number corresponding to that character for
number of times equal to position of the
character. For example, for character C,
we press number 2 three times and accordingly.
```



```cpp
#include <bits/stdc++.h>
using namespace std;

// Function which computes the sequence
string printSequence(string arr[],
                     string input, int n)
{
    string output = "";

    for (int i=0; i<n; i++)
    {
        // Checking for space
        if (input[i] == ' ')
            output = output + "0";
        else
        {
            // Calculating index for each
            // character
            int position = input[i]-'A';
            output = output + arr[position];
        }
    }
    // Output sequence
    return output;
}

// Driver function
int main()
{
    // storing the sequence in array
    string str[] = {"2","22","222",
                    "3","33","333",
                    "4","44","444",
                    "5","55","555",
                    "6","66","666",
                    "7","77","777","7777",
                    "8","88","888",
                    "9","99","999","9999"
                   };

    string input;
    cin>>input;
    int n = input.length();
    cout << printSequence(str, input, n);
    return 0;
}
```

12-Min num. of flips-

Min Number of Flips

Easy Accuracy: 54.6% Submissions: 9161 Points: 2

Given a binary string, that is it contains only 0s and 1s. We need to make this string a sequence of alternate characters by flipping some of the bits, our goal is to minimize the number of bits to be flipped.

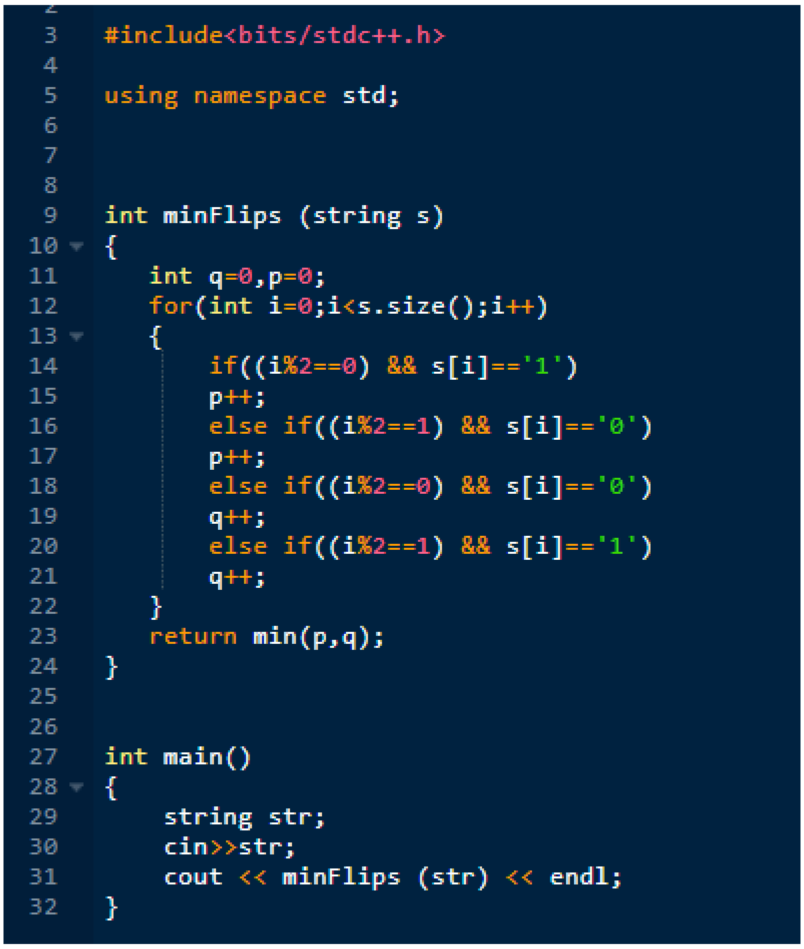
Example 1:

```
Input:
S = "001"
Output: 1
Explanation: We can flip the 0th bit to 1
to have "101".
```

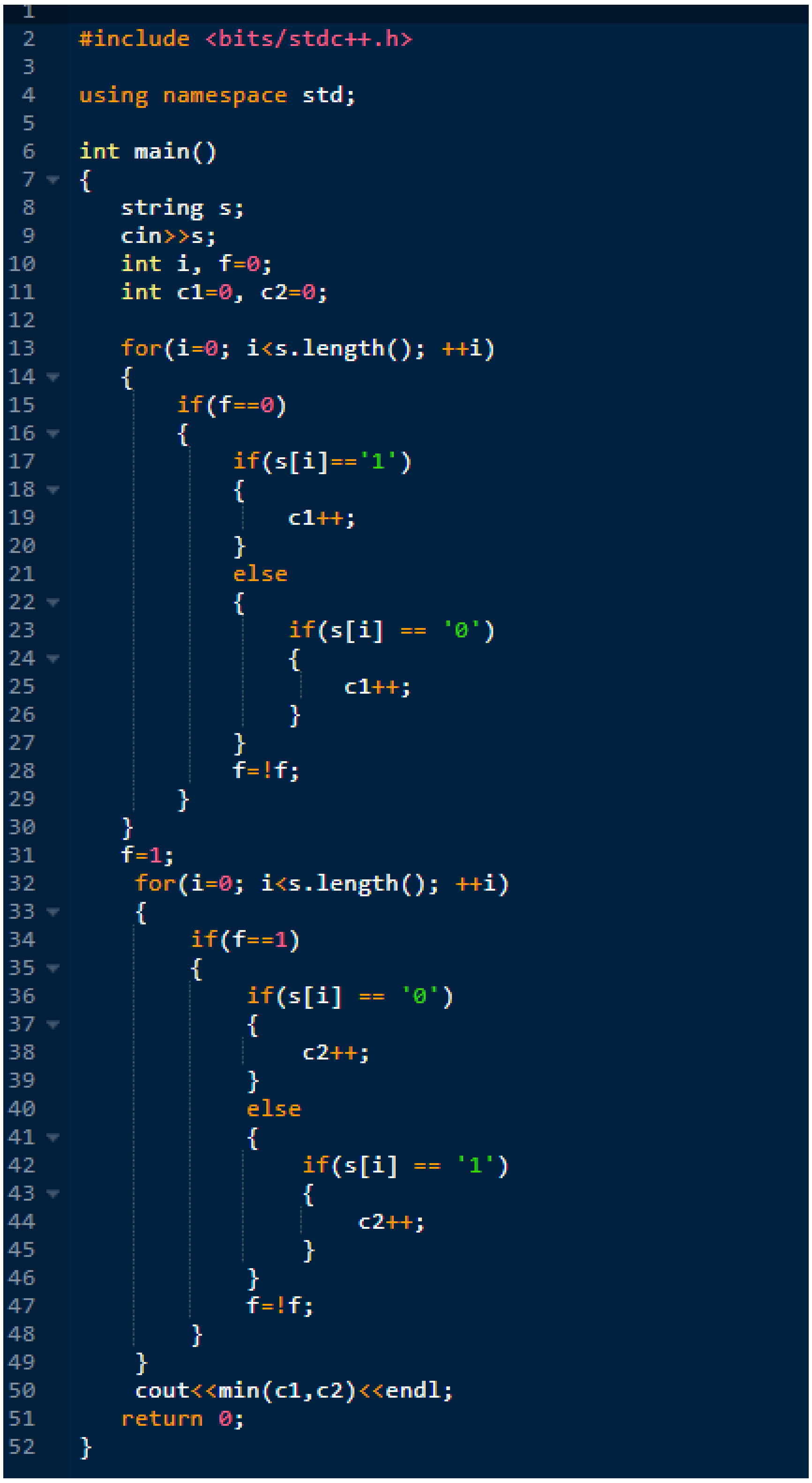
â€‹**Example 2:**

```
Input:
S = "0001010111"
Output: 2
Explanation: We can flip the 1st and 8th bit
to have "0101010101".
```

Your Task:
You don't need to read input or print anything. Your task is to complete the function **minFlips()** which takes the string S as input and returns the minimum number of flips required.



```cpp
#include<bits/stdc++.h>

using namespace std;



int minFlips (string s)
{
    int q=0,p=0;
    for(int i=0;i<s.size();i++)
    {
        if((i%2==0) && s[i]=='1')
        p++;
        else if((i%2==1) && s[i]=='0')
        p++;
        else if((i%2==0) && s[i]=='0')
        q++;
        else if((i%2==1) && s[i]=='1')
        q++;
    }
    return min(p,q);
}


int main()
{
     string str;
     cin>>str;
     cout << minFlips (str) << endl;
}
```



```cpp

#include <bits/stdc++.h>

using namespace std;

int main()
{
    string s;
    cin>>s;
    int i, f=0;
    int c1=0, c2=0;

    for(i=0; i<s.length(); ++i)
    {
        if(f==0)
        {
            if(s[i]=='1')
            {
                c1++;
            }
            else
            {
                if(s[i] == '0')
                {
                    c1++;
                }
            }
            f=!f;
        }
    }
    f=1;
     for(i=0; i<s.length(); ++i)
     {
         if(f==1)
         {
             if(s[i] == '0')
             {
                 c2++;
             }
             else
             {
                 if(s[i] == '1')
                 {
                     c2++;
                 }
             }
             f=!f;
         }
     }
     cout<<min(c1,c2)<<endl;
    return 0;
}
```

13-Second most repeated string –

Second most repeated string in a sequence

Easy Accuracy: 50.47% Submissions: 12311 Points: 2

Given a sequence of strings, the task is to find out the second most repeated (or frequent) string in the given sequence.

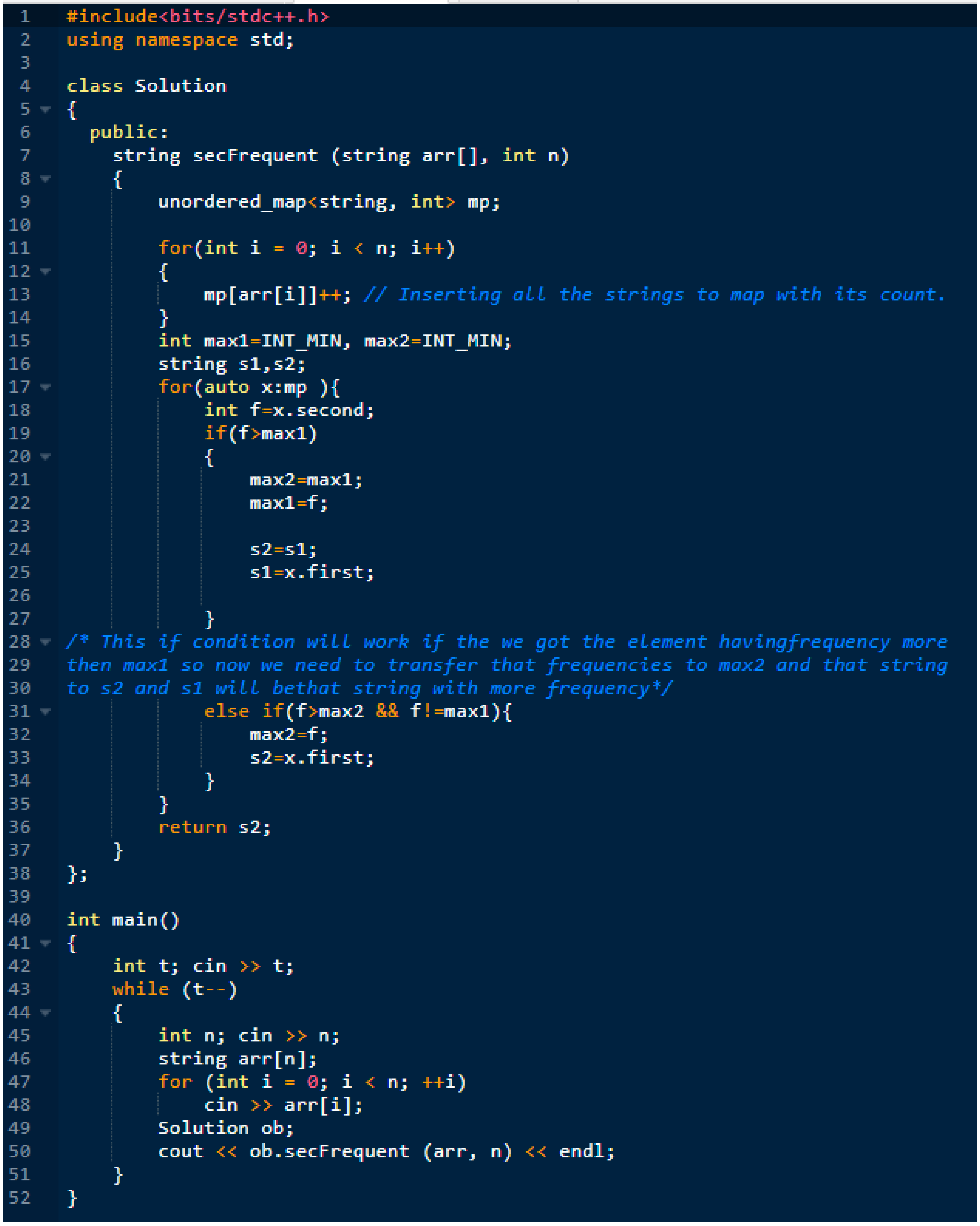
Note: No two strings are the second most repeated, there will be always a single string.

Example 1:

```
Input:
N = 6
arr[] = {aaa, bbb, ccc, bbb, aaa, aaa}
Output: bbb
Explanation: "bbb" is the second most
occurring string with frequency 2.
```

â€‹**Example 2:**

```
Input:
N = 6
arr[] = {geek, for, geek, for, geek, aaa}
Output: for
Explanation: "for" is the second most
occurring string with frequency 2.
```



```cpp
#include<bits/stdc++.h>
using namespace std;

class Solution
{
  public:
    string secFrequent (string arr[], int n)
    {
        unordered_map<string, int> mp;

        for(int i = 0; i < n; i++)
        {
            mp[arr[i]]++; // Inserting all the strings to map with its count.
        }
        int max1=INT_MIN, max2=INT_MIN;
        string s1,s2;
        for(auto x:mp ){
            int f=x.second;
            if(f>max1)
            {
                max2=max1;
                max1=f;

                s2=s1;
                s1=x.first;

            }
/* This if condition will work if the we got the element havingfrequency more
then max1 so now we need to transfer that frequencies to max2 and that string
to s2 and s1 will bethat string with more frequency*/
            else if(f>max2 && f!=max1){
                max2=f;
                s2=x.first;
            }
        }
        return s2;
    }
};

int main()
{
    int t; cin >> t;
    while (t--)
    {
        int n; cin >> n;
        string arr[n];
        for (int i = 0; i < n; ++i)
            cin >> arr[i];
        Solution ob;
        cout << ob.secFrequent (arr, n) << endl;
    }
}
```

14-Min. swaps for bracket balancing-

Minimum Swaps for Bracket Balancing

Easy   Accuracy: 41.16%   Submissions: 8052   Points: 2

You are given a string S of 2N characters consisting of N '[' brackets and N ']' brackets. A string is considered balanced if it can be represented in the for S2[S1] where S1 and S2 are balanced strings. We can make an unbalanced string balanced by swapping **adjacent** characters. Calculate the minimum number of swaps necessary to make a string balanced.
Note - Strings S1 and S2 can be empty.

Example 1:

```
Input  : []][][
Output : 2
Explanation :
First swap: Position 3 and 4
[][]][
Second swap: Position 5 and 6
[][][]
```

Example 2:

```
Input : [[][]]
Output : 0
Explanation:
String is already balanced.
```

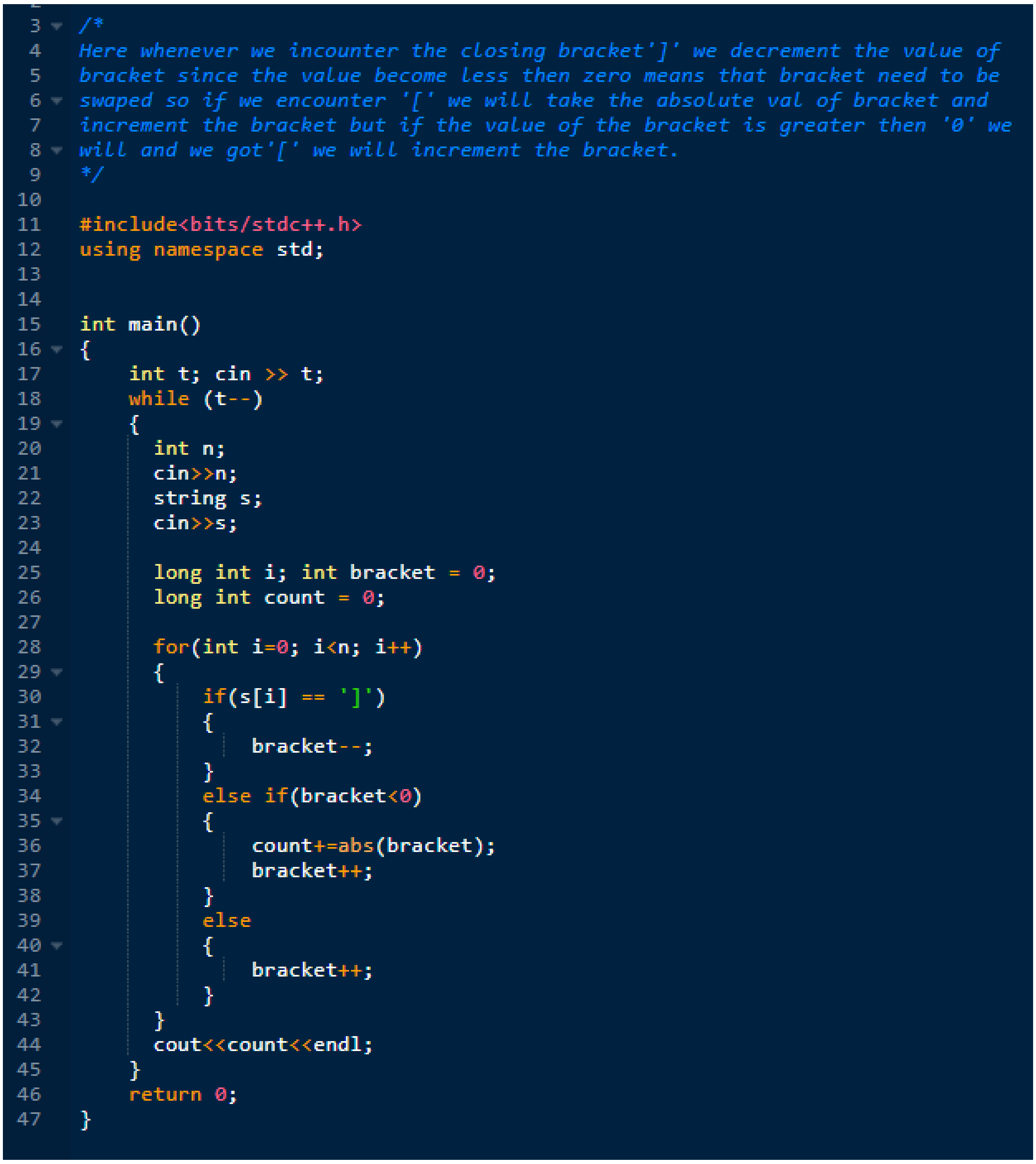
Your Task:

You don't need to read input or print anything. Your task is to complete the function **minimumNumberOfSwaps()** which takes the string S and return minimum number of operations required to balance the bracket sequence.

Expected Time Complexity: O(N).
Expected Auxiliary Space: O(1).

Constraints:

1<=|S|<=100000



```
/*
Here whenever we incounter the closing bracket']' we decrement the value of
bracket since the value become less then zero means that bracket need to be
swaped so if we encounter '[' we will take the absolute val of bracket and
increment the bracket but if the value of the bracket is greater then '0' we
will and we got'[' we will increment the bracket.
*/

#include<bits/stdc++.h>
using namespace std;


int main()
{
    int t; cin >> t;
    while (t--)
    {
        int n;
        cin>>n;
        string s;
        cin>>s;

        long int i; int bracket = 0;
        long int count = 0;

        for(int i=0; i<n; i++)
        {
            if(s[i] == ']')
            {
                bracket--;
            }
            else if(bracket<0)
            {
                count+=abs(bracket);
                bracket++;
            }
            else
            {
                bracket++;
            }
        }
        cout<<count<<endl;
    }
    return 0;
}
```

15-Rearrange characters-

Rearrange characters

Accuracy: 49.98% Submissions: 28139 Points: 60

Given a string S with repeated characters (only lowercase). The task is to rearrange characters in a string such that no two adjacent characters are same.

Note : It may be assumed that the string has only lowercase English alphabets.

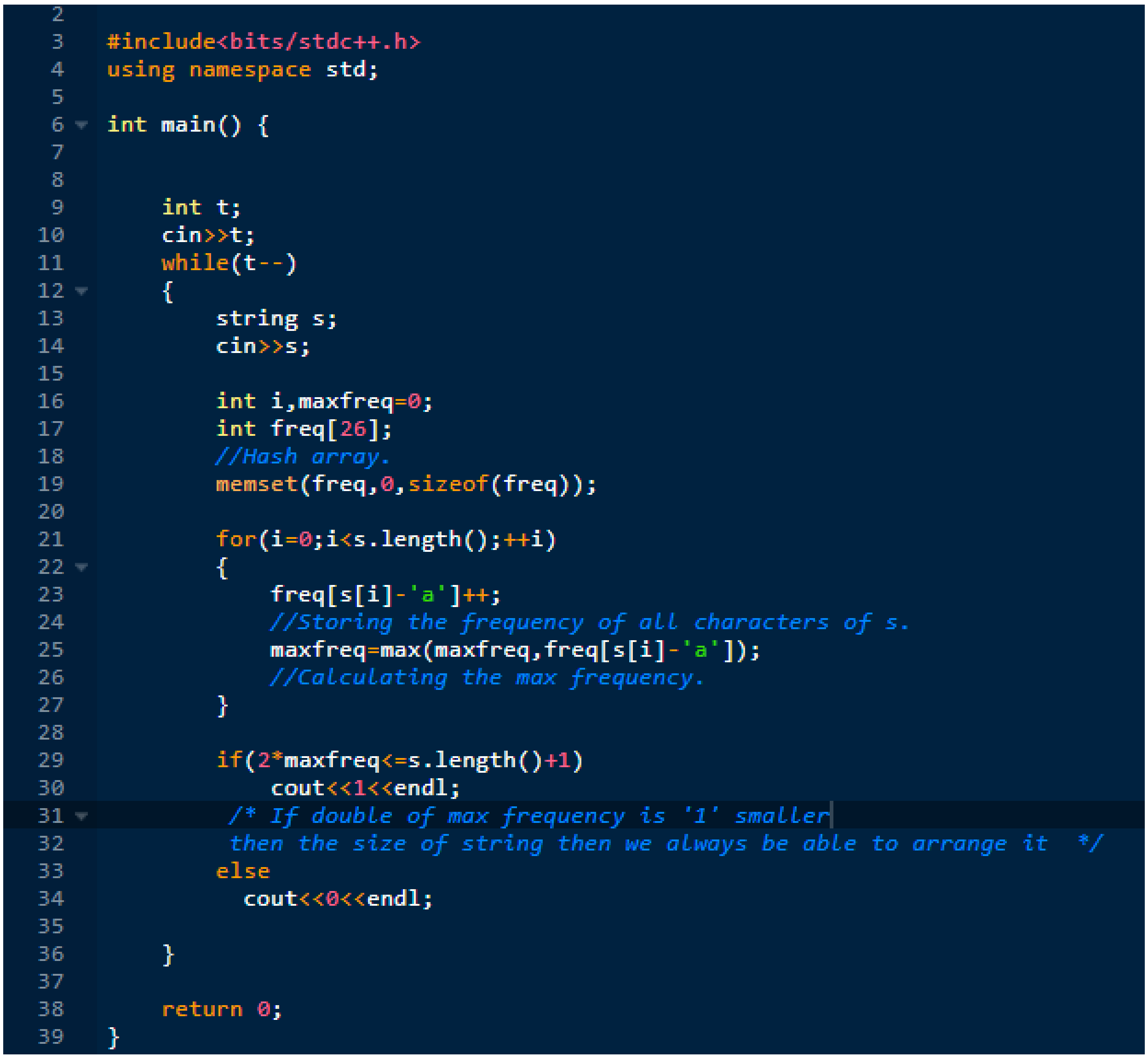
Input:
The first line of input contains an integer T denoting the number of test cases. Then T test cases follow. Each test case contains a single line containing a string of lowercase english alphabets.

Output:
For each test case in a new line print "1" (without quotes) if the generated string doesn't contains any same adjacent characters, else if no such string is possible to be made print "0" (without quotes).

Constraints:
1 <= T <= 100
1 <= length of string <= 10^4

Input:
```
3
geeksforgeeks
bbbabaaacd
bbbbb
```

Output:
```
1
1
0
```



```

#include<bits/stdc++.h>
using namespace std;

int main() {


    int t;
    cin>>t;
    while(t--)
    {
        string s;
        cin>>s;

        int i,maxfreq=0;
        int freq[26];
        //Hash array.
        memset(freq,0,sizeof(freq));

        for(i=0;i<s.length();++i)
        {
            freq[s[i]-'a']++;
            //Storing the frequency of all characters of s.
            maxfreq=max(maxfreq,freq[s[i]-'a']);
            //Calculating the max frequency.
        }

        if(2*maxfreq<=s.length()+1)
            cout<<1<<endl;
         /* If double of max frequency is '1' smaller
         then the size of string then we always be able to arrange it  */
        else
          cout<<0<<endl;

    }

    return 0;
}
```

16-Remove consecutive duplicates-

Remove Consecutive Characters

Basic Accuracy: 62.41% Submissions: 8766 Points: 1

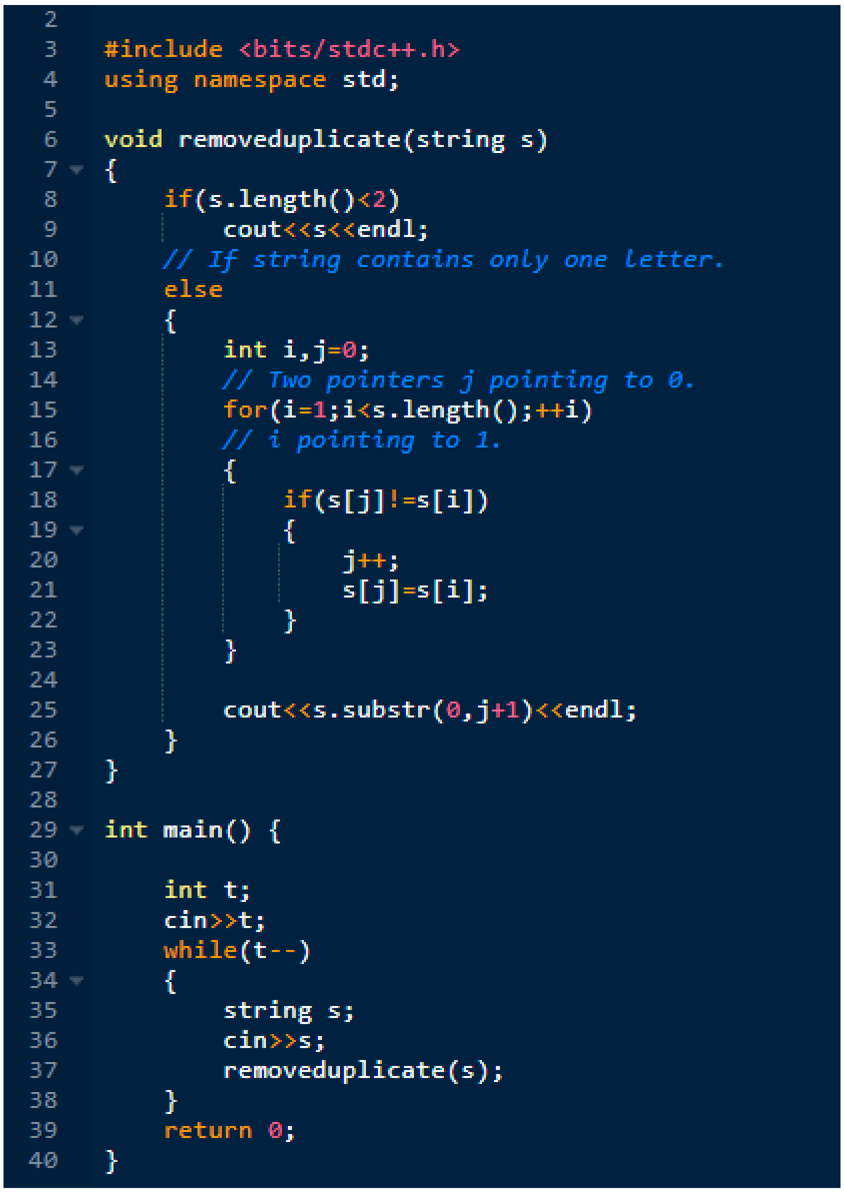
Given a string **S** delete the characters which are appearing more than once consecutively.

Example 1:

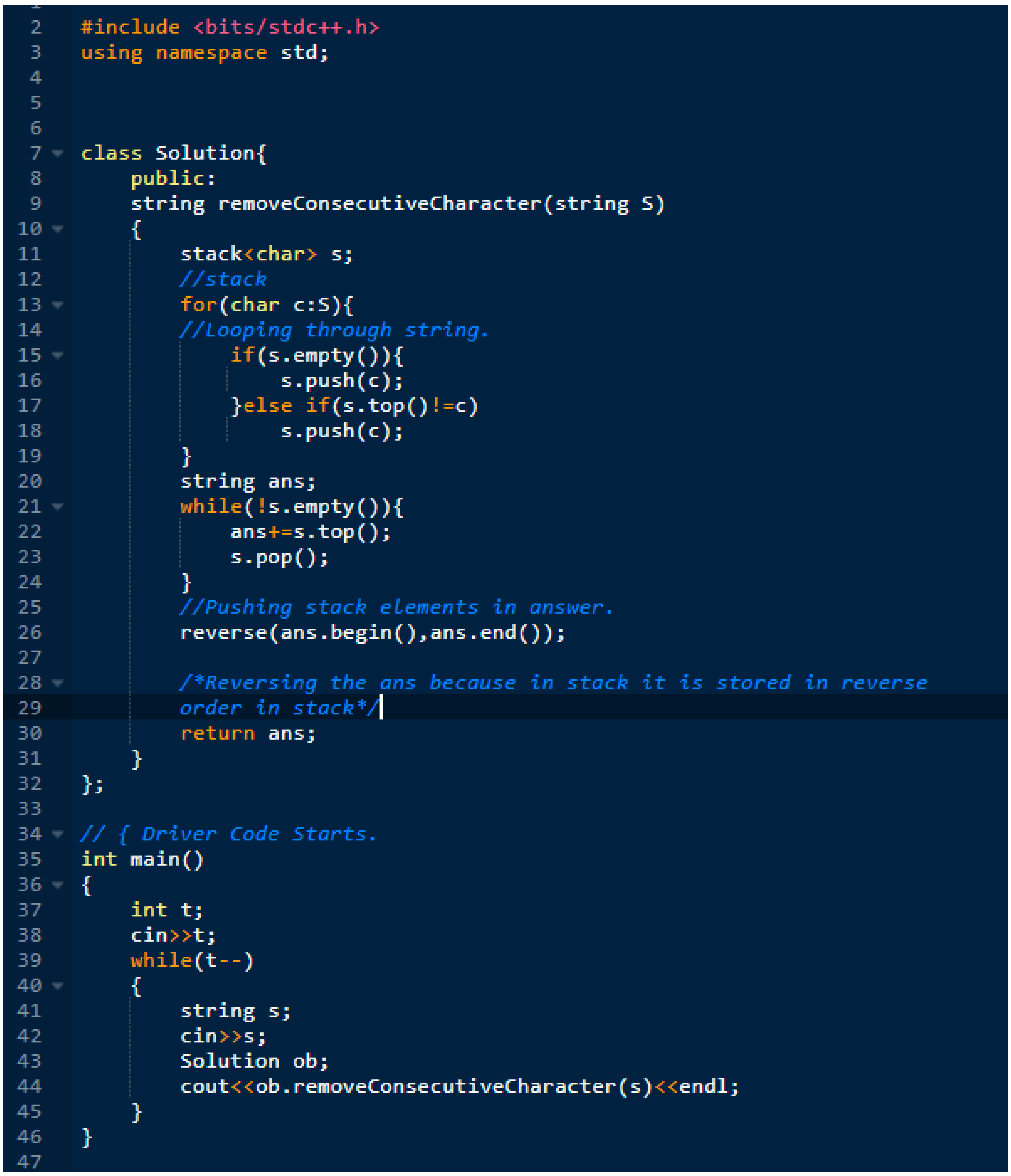
```
Input:
S = aabb
Output:  ab
Explanation: 'a' at 2nd position is
appearing 2nd time consecutively.
Similiar explanation for b at
4th position.
```

Example 2:

```
Input:
S = aabaa
Output:  aba
Explanation: 'a' at 2nd position is
appearing 2nd time consecutively.
'a' at fifth position is appearing
2nd time consecutively.
```



```cpp

#include <bits/stdc++.h>
using namespace std;

void removeduplicate(string s)
{
    if(s.length()<2)
        cout<<s<<endl;
    // If string contains only one letter.
    else
    {
        int i,j=0;
        // Two pointers j pointing to 0.
        for(i=1;i<s.length();++i)
        // i pointing to 1.
        {
            if(s[j]!=s[i])
            {
                j++;
                s[j]=s[i];
            }
        }

        cout<<s.substr(0,j+1)<<endl;
    }
}

int main() {

    int t;
    cin>>t;
    while(t--)
    {
        string s;
        cin>>s;
        removeduplicate(s);
    }
    return 0;
}
```



```
#include <bits/stdc++.h>
using namespace std;



class Solution{
    public:
    string removeConsecutiveCharacter(string S)
    {
        stack<char> s;
        //stack
        for(char c:S){
        //Looping through string.
            if(s.empty()){
                s.push(c);
            }else if(s.top()!=c)
                s.push(c);
        }
        string ans;
        while(!s.empty()){
            ans+=s.top();
            s.pop();
        }
        //Pushing stack elements in answer.
        reverse(ans.begin(),ans.end());

        /*Reversing the ans because in stack it is stored in reverse
        order in stack*/
        return ans;
    }
};

// { Driver Code Starts.
int main()
{
    int t;
    cin>>t;
    while(t--)
    {
        string s;
        cin>>s;
        Solution ob;
        cout<<ob.removeConsecutiveCharacter(s)<<endl;
    }
}

```

17-Print anagrams together-

Print Anagrams Together

Medium Accuracy: 56.1% Submissions: 18089 Points: 4

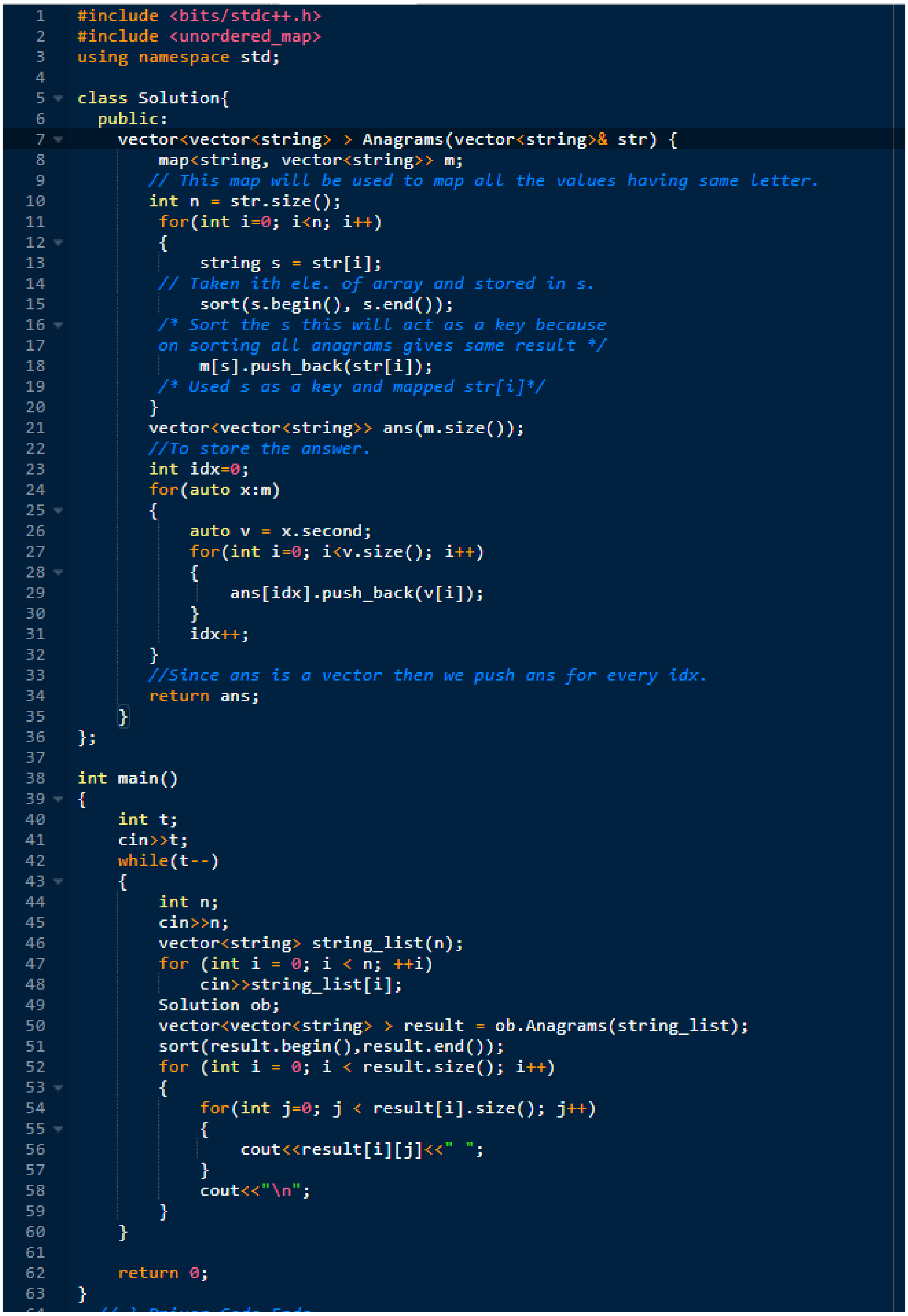
Given an array of strings, return all groups of strings that are anagrams. The groups must be created in order of their appearance in the original array. Look at the sample case for clarification.

Example 1:

```
Input:
N = 5
words[] = {act,god,cat,dog,tac}
Output:
god dog
act cat tac
Explanation:
There are 2 groups of
anagrams "god", "dog" make group 1.
"act", "cat", "tac" make group 2.
```

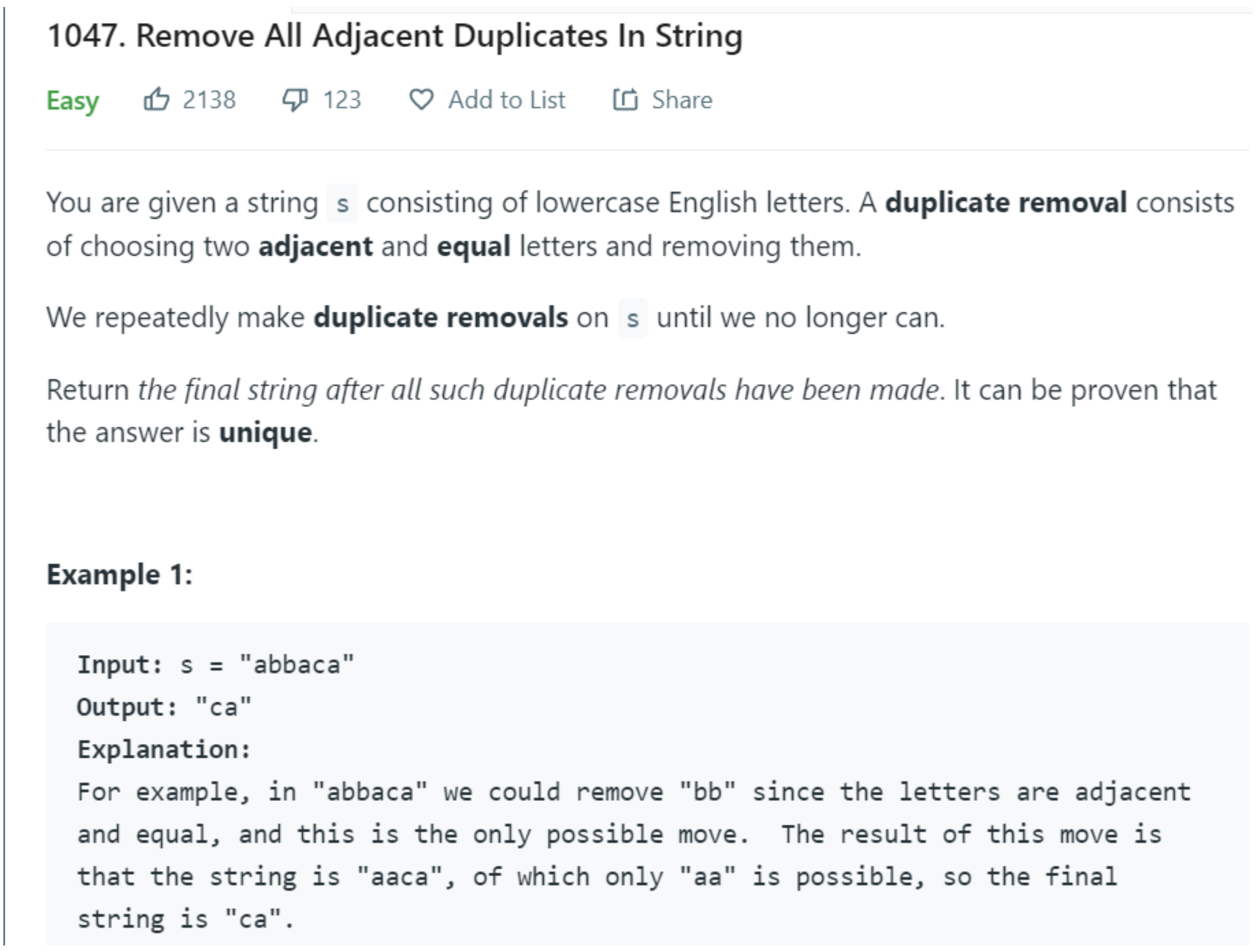
Example 2:

```
Input:
N = 3
words[] = {no,on,is}
Output:
no on
is
```



```cpp
#include <bits/stdc++.h>
#include <unordered_map>
using namespace std;

class Solution{
  public:
    vector<vector<string> > Anagrams(vector<string>& str) {
        map<string, vector<string>> m;
       // This map will be used to map all the values having same letter.
       int n = str.size();
        for(int i=0; i<n; i++)
        {
            string s = str[i];
        // Taken ith ele. of array and stored in s.
            sort(s.begin(), s.end());
        /* Sort the s this will act as a key because
        on sorting all anagrams gives same result */
            m[s].push_back(str[i]);
        /* Used s as a key and mapped str[i]*/
       }
       vector<vector<string>> ans(m.size());
       //To store the answer.
       int idx=0;
       for(auto x:m)
       {
           auto v = x.second;
           for(int i=0; i<v.size(); i++)
           {
               ans[idx].push_back(v[i]);
           }
           idx++;
       }
       //Since ans is a vector then we push ans for every idx.
       return ans;
    }
};

int main()
{
    int t;
    cin>>t;
    while(t--)
    {
        int n;
        cin>>n;
        vector<string> string_list(n);
        for (int i = 0; i < n; ++i)
            cin>>string_list[i];
        Solution ob;
        vector<vector<string> > result = ob.Anagrams(string_list);
        sort(result.begin(),result.end());
        for (int i = 0; i < result.size(); i++)
        {
            for(int j=0; j < result[i].size(); j++)
            {
                cout<<result[i][j]<<" ";
            }
            cout<<"\n";
        }
    }

    return 0;
}
```

18-Remove all adjacent duplicates in strings-



1047. Remove All Adjacent Duplicates In String

Easy 2138 123 Add to List Share

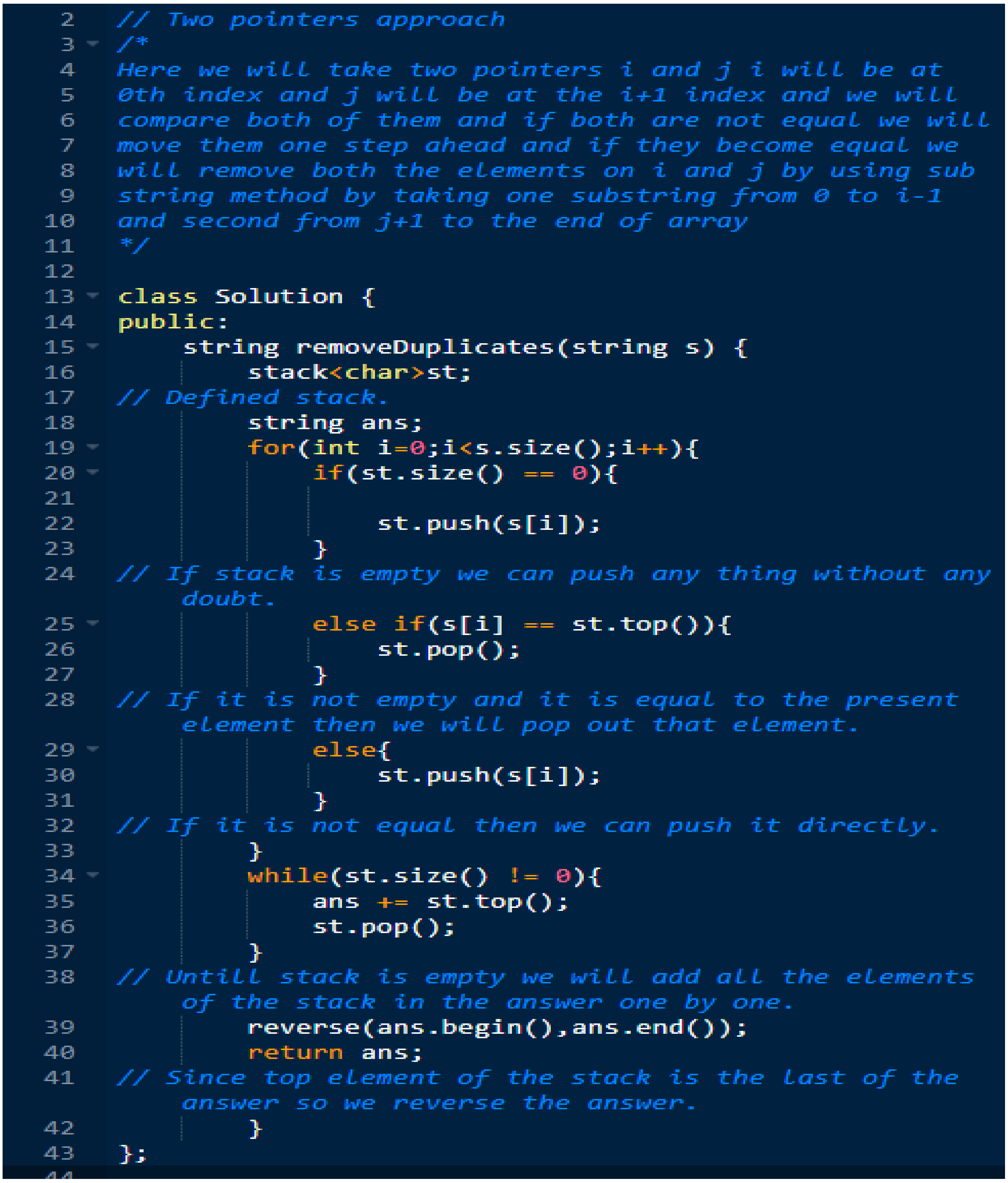
You are given a string `s` consisting of lowercase English letters. A **duplicate removal** consists of choosing two **adjacent** and **equal** letters and removing them.

We repeatedly make **duplicate removals** on `s` until we no longer can.

Return *the final string after all such duplicate removals have been made*. It can be proven that the answer is **unique**.

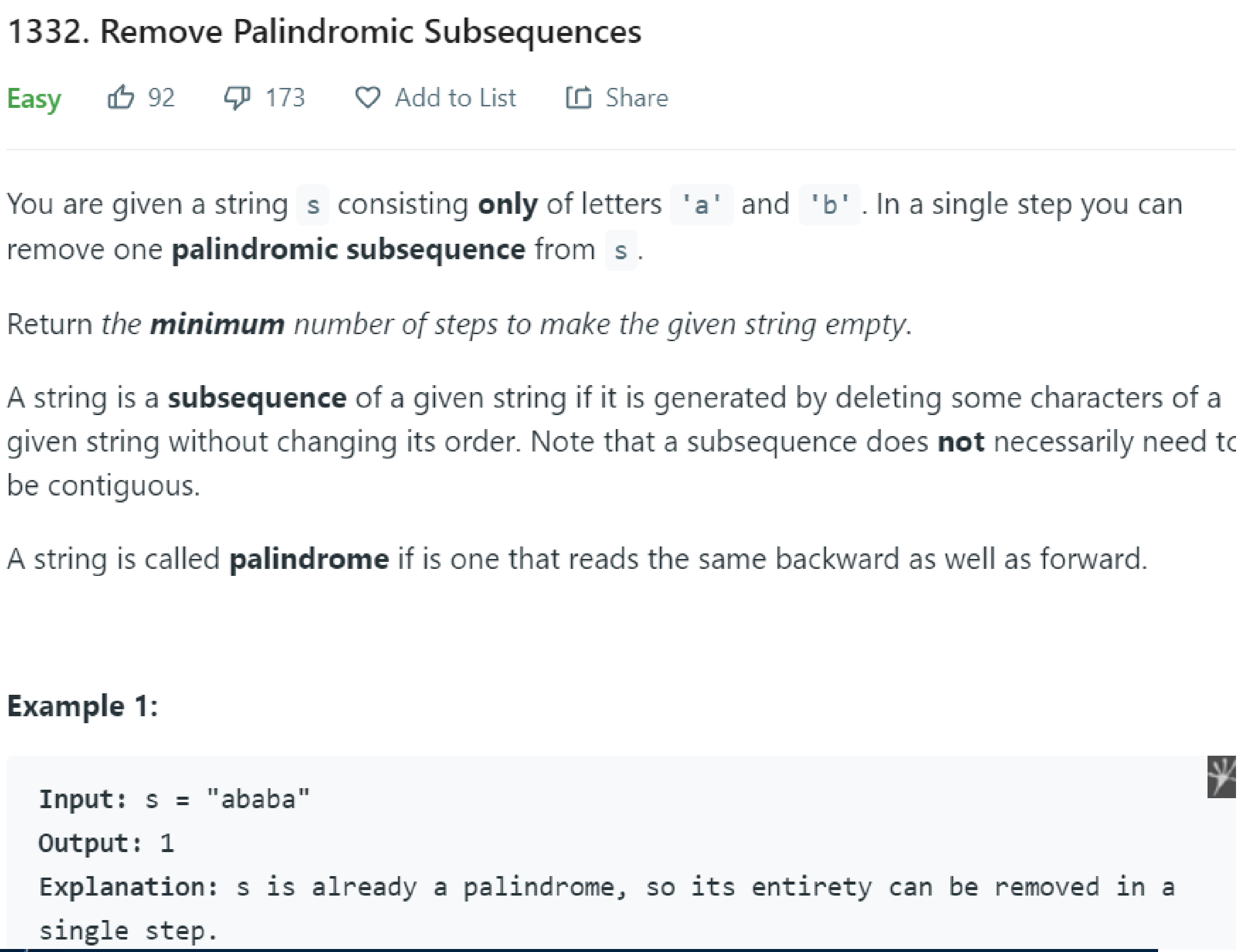
Example 1:

```
Input: s = "abbaca"
Output: "ca"
Explanation:
For example, in "abbaca" we could remove "bb" since the letters are adjacent
and equal, and this is the only possible move.  The result of this move is
that the string is "aaca", of which only "aa" is possible, so the final
string is "ca".
```



```
// Two pointers approach
/*
Here we will take two pointers i and j i will be at
0th index and j will be at the i+1 index and we will
compare both of them and if both are not equal we will
move them one step ahead and if they become equal we
will remove both the elements on i and j by using sub
string method by taking one substring from 0 to i-1
and second from j+1 to the end of array
*/

class Solution {
public:
    string removeDuplicates(string s) {
        stack<char>st;
// Defined stack.
        string ans;
        for(int i=0;i<s.size();i++){
            if(st.size() == 0){

                st.push(s[i]);
            }
// If stack is empty we can push any thing without any
    doubt.
            else if(s[i] == st.top()){
                st.pop();
            }
// If it is not empty and it is equal to the present
    element then we will pop out that element.
            else{
                st.push(s[i]);
            }
// If it is not equal then we can push it directly.
        }
        while(st.size() != 0){
            ans += st.top();
            st.pop();
        }
// Untill stack is empty we will add all the elements
    of the stack in the answer one by one.
        reverse(ans.begin(),ans.end());
        return ans;
// Since top element of the stack is the last of the
    answer so we reverse the answer.
    }
};
```

19-Remove palindromic sub string to make string empty-



1332. Remove Palindromic Subsequences

Easy 92 173 Add to List Share

You are given a string `s` consisting **only** of letters `'a'` and `'b'`. In a single step you can remove one **palindromic subsequence** from `s`.

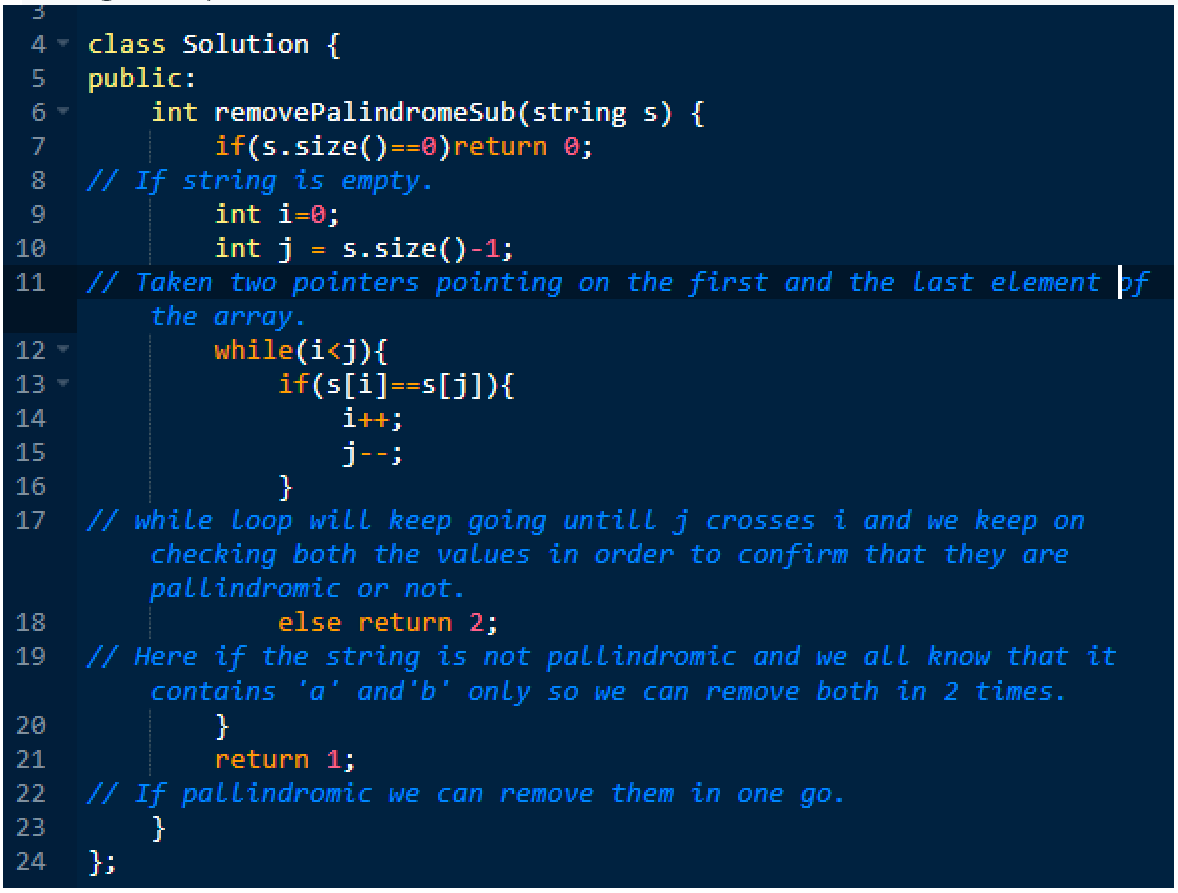
Return *the **minimum** number of steps to make the given string empty.*

A string is a **subsequence** of a given string if it is generated by deleting some characters of a given string without changing its order. Note that a subsequence does **not** necessarily need t be contiguous.

A string is called **palindrome** if is one that reads the same backward as well as forward.

Example 1:

```
Input: s = "ababa"
Output: 1
Explanation: s is already a palindrome, so its entirety can be removed in a single step.
```



```
class Solution {
public:
    int removePalindromeSub(string s) {
        if(s.size()==0)return 0;
// If string is empty.
        int i=0;
        int j = s.size()-1;
// Taken two pointers pointing on the first and the last element of the array.
        while(i<j){
            if(s[i]==s[j]){
                i++;
                j--;
            }
// while loop will keep going untill j crosses i and we keep on checking both the values in order to confirm that they are pallindromic or not.
            else return 2;
// Here if the string is not pallindromic and we all know that it contains 'a' and'b' only so we can remove both in 2 times.
        }
        return 1;
// If pallindromic we can remove them in one go.
    }
};
```

20-Destination cities-

1436. Destination City

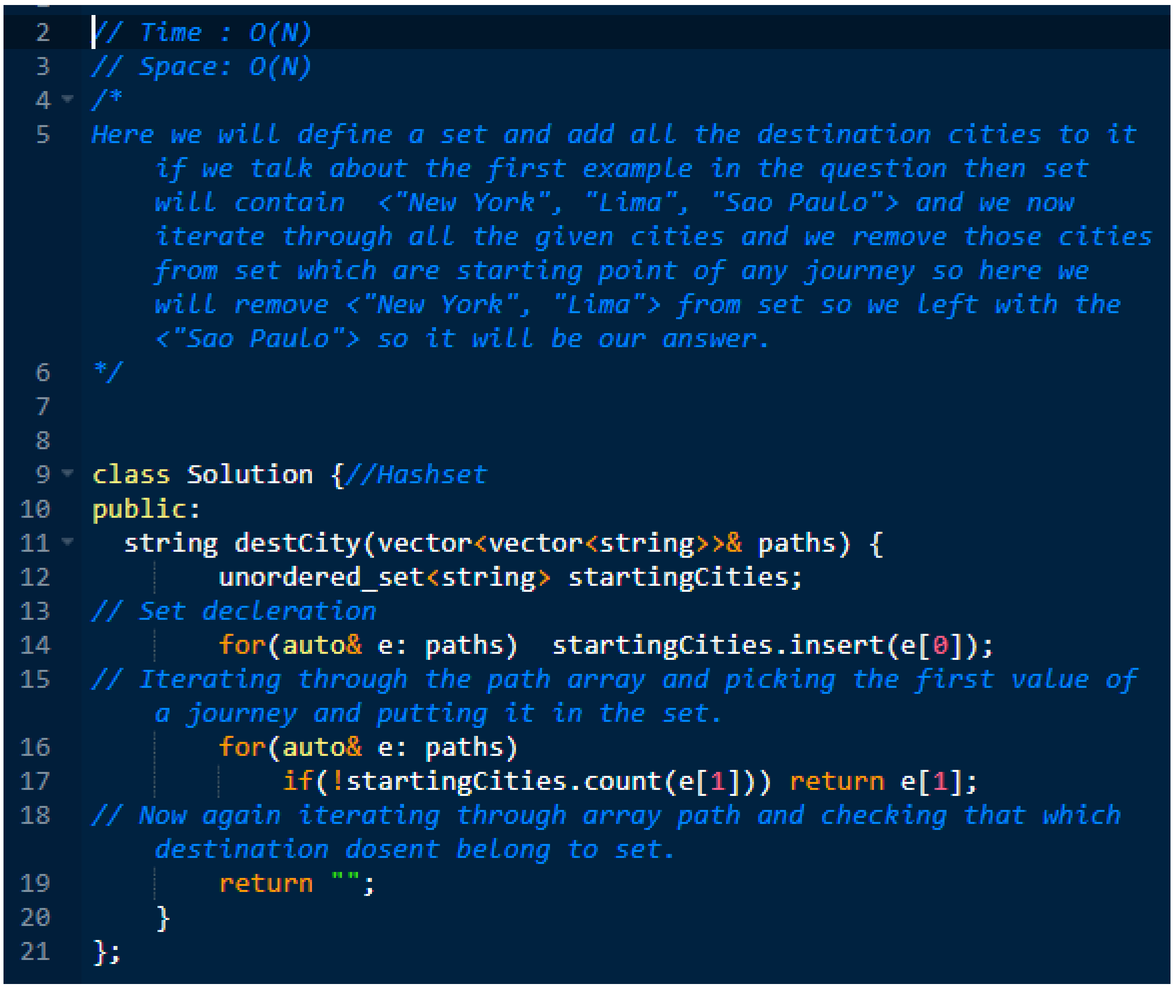
Easy 727 53 Add to List Share

You are given the array `paths`, where `paths[i] = [cityAi, cityBi]` means there exists a direct path going from `cityAi` to `cityBi`. *Return the destination city, that is, the city without any path outgoing to another city.*

It is guaranteed that the graph of paths forms a line without any loop, therefore, there will be exactly one destination city.

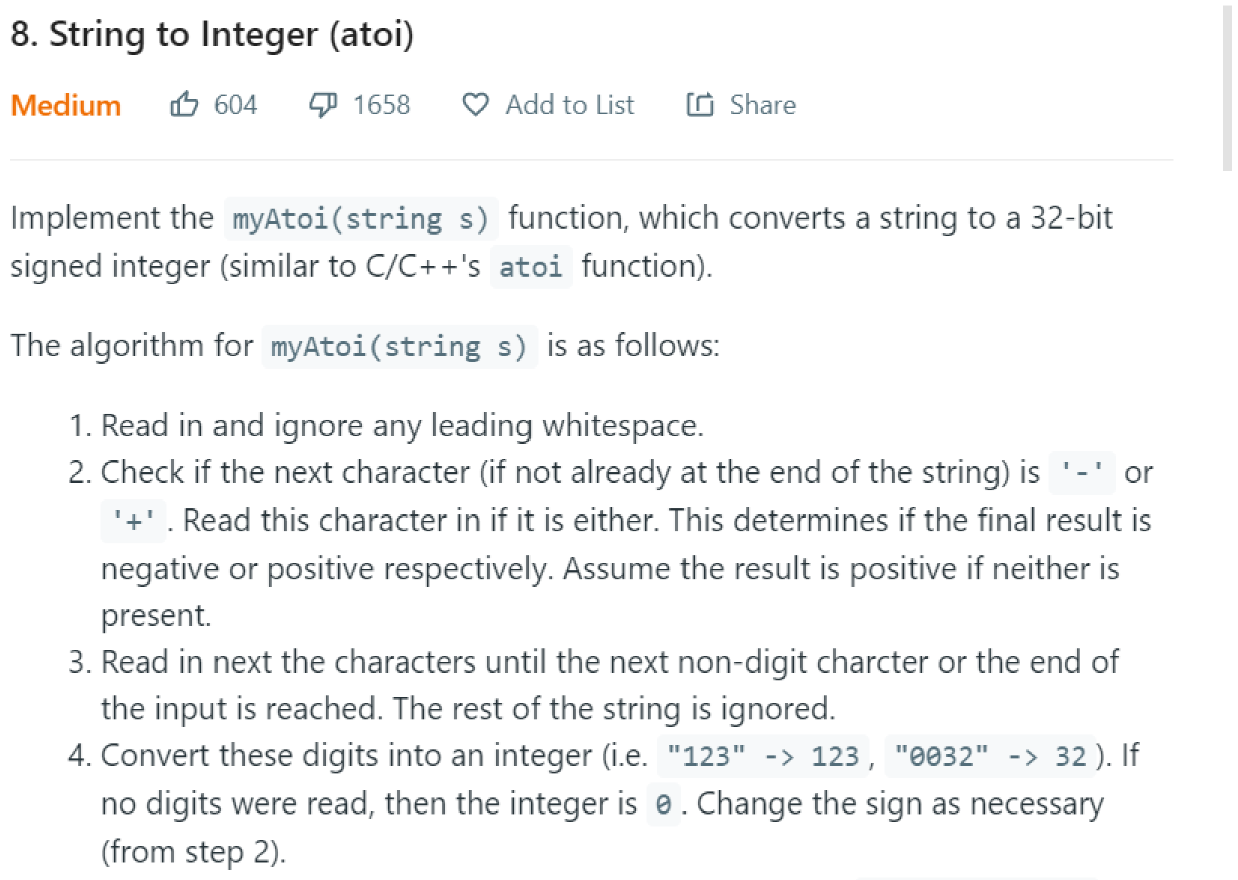
Example 1:

```
Input: paths = [["London","New York"],["New York","Lima"],["Lima","Sao
Paulo"]]
Output: "Sao Paulo"
Explanation: Starting at "London" city you will reach "Sao Paulo" city which
is the destination city. Your trip consist of: "London" -> "New York" ->
"Lima" -> "Sao Paulo".
```



```
// Time : O(N)
// Space: O(N)
/*
Here we will define a set and add all the destination cities to it
    if we talk about the first example in the question then set
    will contain  <"New York", "Lima", "Sao Paulo"> and we now
    iterate through all the given cities and we remove those cities
    from set which are starting point of any journey so here we
    will remove <"New York", "Lima"> from set so we left with the
    <"Sao Paulo"> so it will be our answer.
*/


class Solution {//Hashset
public:
  string destCity(vector<vector<string>>& paths) {
        unordered_set<string> startingCities;
// Set decleration
        for(auto& e: paths)  startingCities.insert(e[0]);
// Iterating through the path array and picking the first value of
    a journey and putting it in the set.
        for(auto& e: paths)
            if(!startingCities.count(e[1])) return e[1];
// Now again iterating through array path and checking that which
    destination dosent belong to set.
        return "";
    }
};
```

21-String to integer-



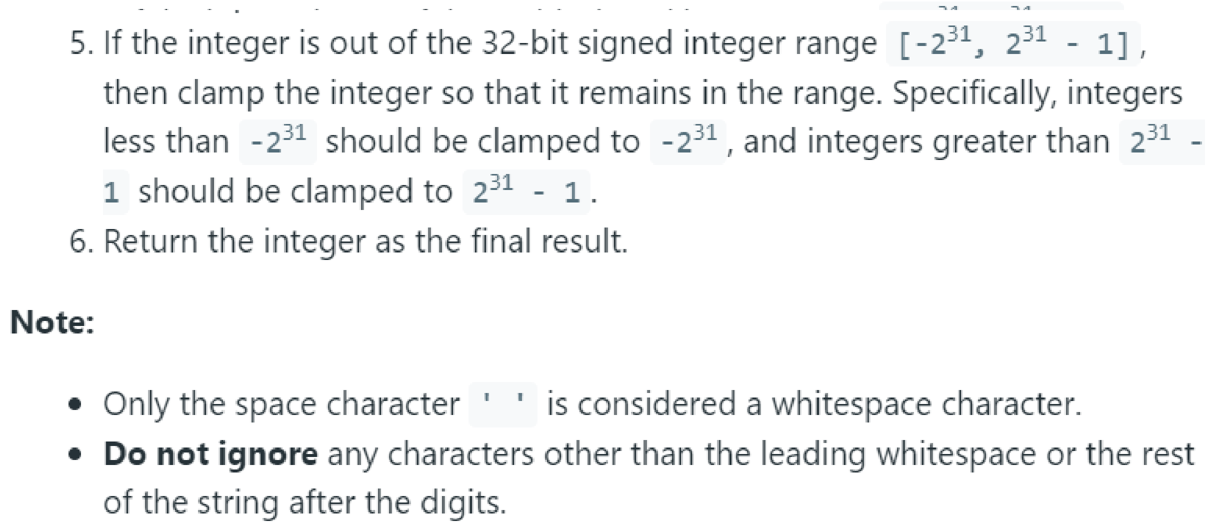
8. String to Integer (atoi)

Medium 604 1658 Add to List Share

Implement the `myAtoi(string s)` function, which converts a string to a 32-bit signed integer (similar to C/C++'s `atoi` function).

The algorithm for `myAtoi(string s)` is as follows:

1. Read in and ignore any leading whitespace.
2. Check if the next character (if not already at the end of the string) is `'-'` or `'+'`. Read this character in if it is either. This determines if the final result is negative or positive respectively. Assume the result is positive if neither is present.
3. Read in next the characters until the next non-digit charcter or the end of the input is reached. The rest of the string is ignored.
4. Convert these digits into an integer (i.e. `"123" -> 123`, `"0032" -> 32`). If no digits were read, then the integer is `0`. Change the sign as necessary (from step 2).



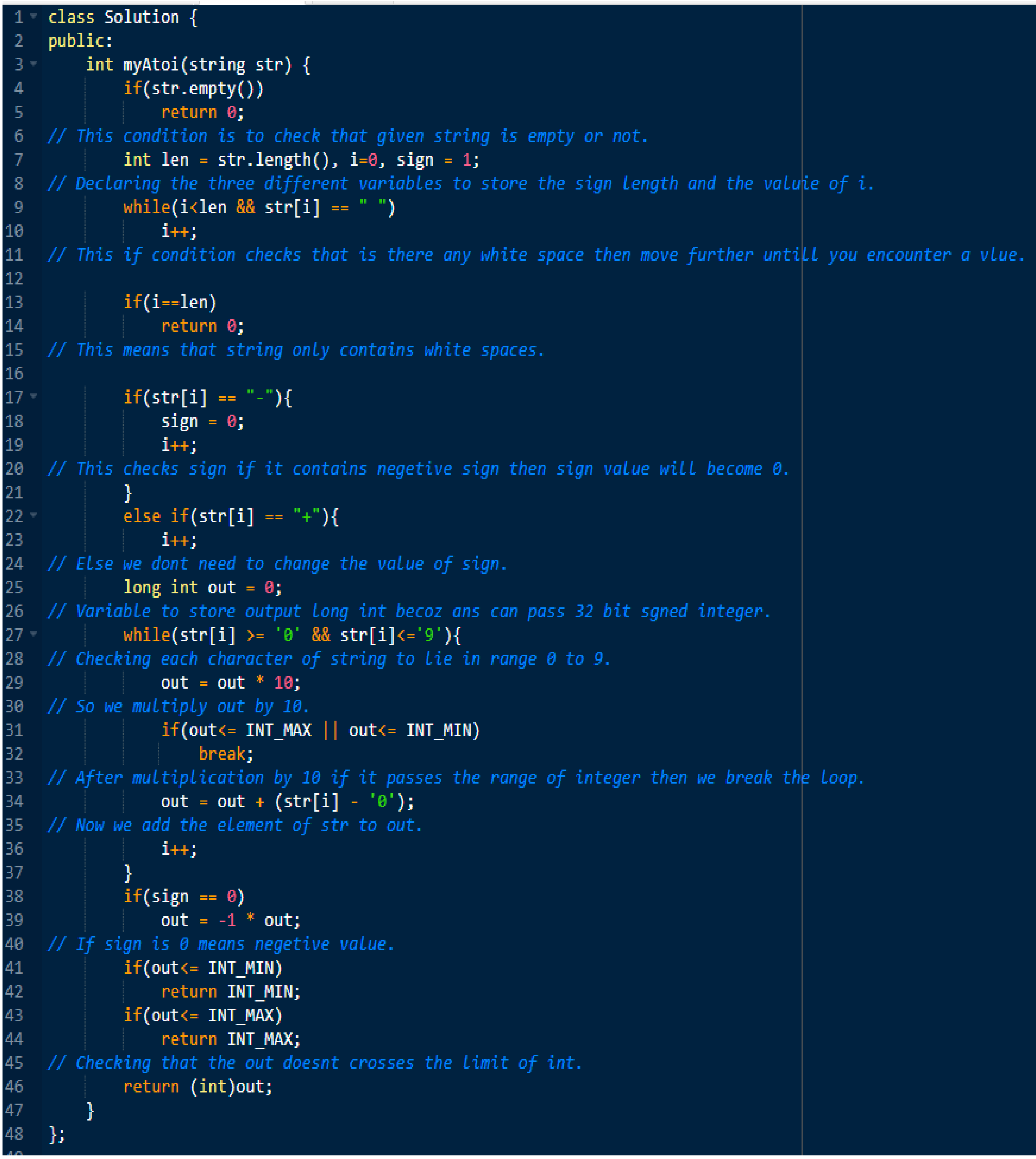
5. If the integer is out of the 32-bit signed integer range $[-2^{31}, 2^{31} - 1]$, then clamp the integer so that it remains in the range. Specifically, integers less than -2^{31} should be clamped to -2^{31}, and integers greater than $2^{31} - 1$ should be clamped to $2^{31} - 1$.
6. Return the integer as the final result.

Note:

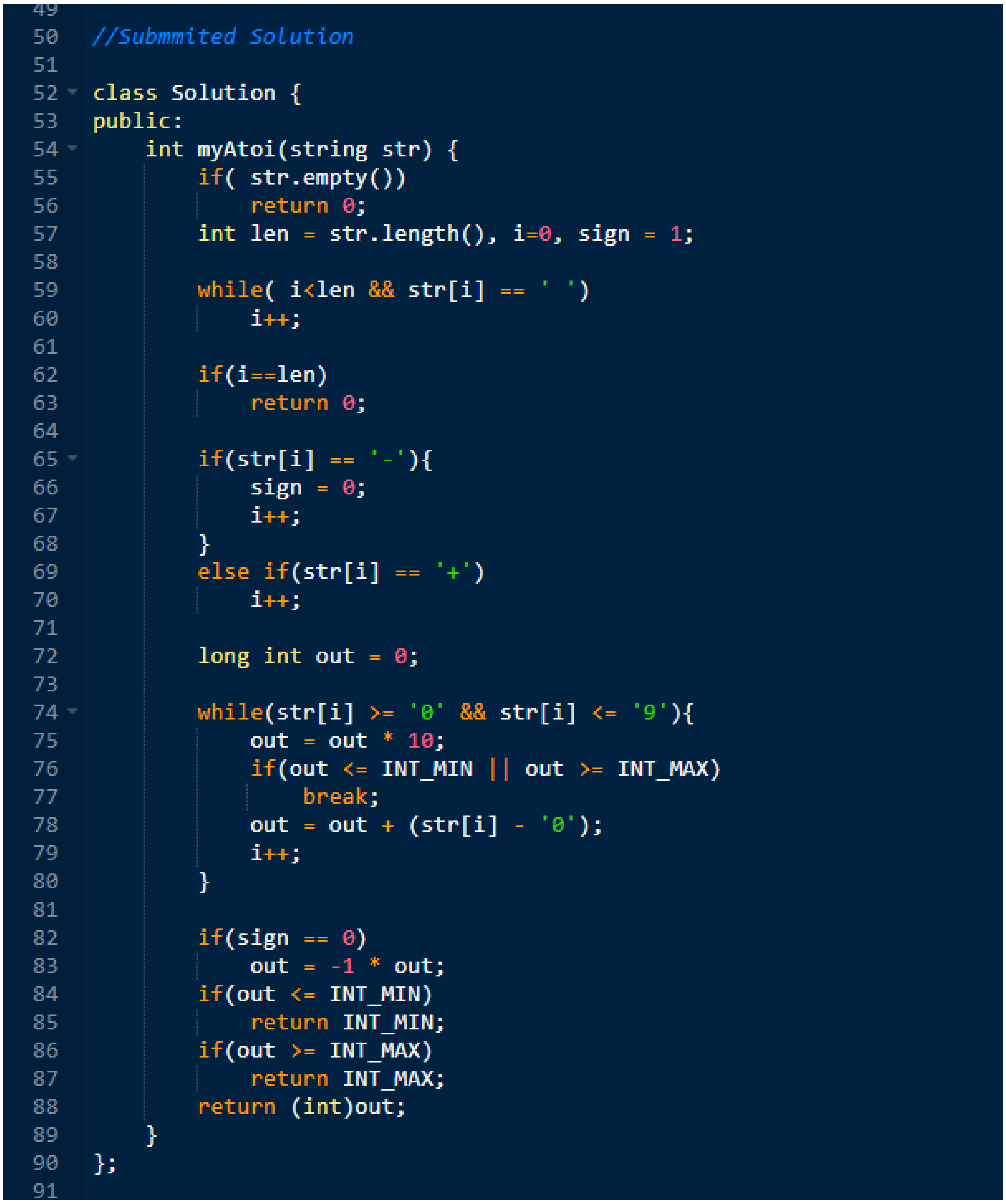
- Only the space character `' '` is considered a whitespace character.
- **Do not ignore** any characters other than the leading whitespace or the rest of the string after the digits.

Example 1:

```
Input: s = "42"
Output: 42
Explanation: The underlined characters are what is read in, the caret is the current reader position.
Step 1: "42" (no characters read because there is no leading whitespace)
         ^
Step 2: "42" (no characters read because there is neither a '-' nor '+')
         ^
Step 3: "42" ("42" is read in)
           ^
The parsed integer is 42.
Since 42 is in the range [$-2^{31}$, $2^{31}$ - 1], the final result is 42.
```



```cpp
class Solution {
public:
    int myAtoi(string str) {
        if(str.empty())
            return 0;
// This condition is to check that given string is empty or not.
        int len = str.length(), i=0, sign = 1;
// Declaring the three different variables to store the sign length and the valuie of i.
        while(i<len && str[i] == " ")
            i++;
// This if condition checks that is there any white space then move further untill you encounter a vlue.

        if(i==len)
            return 0;
// This means that string only contains white spaces.

        if(str[i] == "-"){
            sign = 0;
            i++;
// This checks sign if it contains negetive sign then sign value will become 0.
        }
        else if(str[i] == "+"){
            i++;
// Else we dont need to change the value of sign.
        long int out = 0;
// Variable to store output long int becoz ans can pass 32 bit sgned integer.
        while(str[i] >= '0' && str[i]<='9'){
// Checking each character of string to lie in range 0 to 9.
            out = out * 10;
// So we multiply out by 10.
            if(out<= INT_MAX || out<= INT_MIN)
                break;
// After multiplication by 10 if it passes the range of integer then we break the loop.
            out = out + (str[i] - '0');
// Now we add the element of str to out.
            i++;
        }
        if(sign == 0)
            out = -1 * out;
// If sign is 0 means negetive value.
        if(out<= INT_MIN)
            return INT_MIN;
        if(out<= INT_MAX)
            return INT_MAX;
// Checking that the out doesnt crosses the limit of int.
        return (int)out;
    }
};
```



```
//Submmited Solution

class Solution {
public:
    int myAtoi(string str) {
        if( str.empty())
            return 0;
        int len = str.length(), i=0, sign = 1;

        while( i<len && str[i] == ' ')
            i++;

        if(i==len)
            return 0;

        if(str[i] == '-'){
            sign = 0;
            i++;
        }
        else if(str[i] == '+')
            i++;

        long int out = 0;

        while(str[i] >= '0' && str[i] <= '9'){
            out = out * 10;
            if(out <= INT_MIN || out >= INT_MAX)
                break;
            out = out + (str[i] - '0');
            i++;
        }

        if(sign == 0)
            out = -1 * out;
        if(out <= INT_MIN)
            return INT_MIN;
        if(out >= INT_MAX)
            return INT_MAX;
        return (int)out;
    }
};
```

22- Custom sort of string (791)-

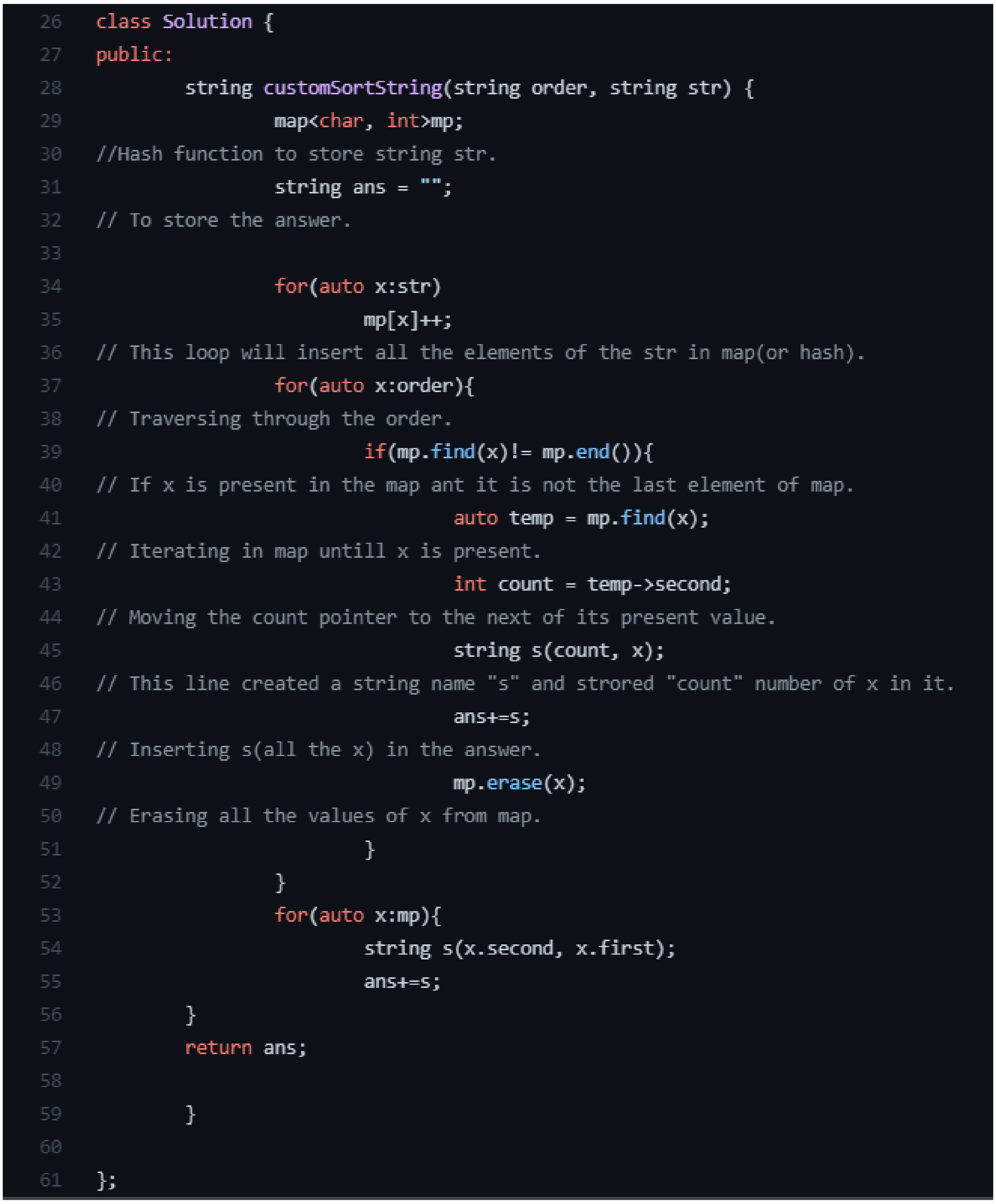
You are given two strings order and s. All the words of `order` are **unique** and were sorted in some custom order previously.

Permute the characters of `s` so that they match the order that `order` was sorted. More specifically, if a character `x` occurs before a character `y` in `order`, then `x` should occur before `y` in the permuted string.

Return *any permutation of* `s` *that satisfies this property.*

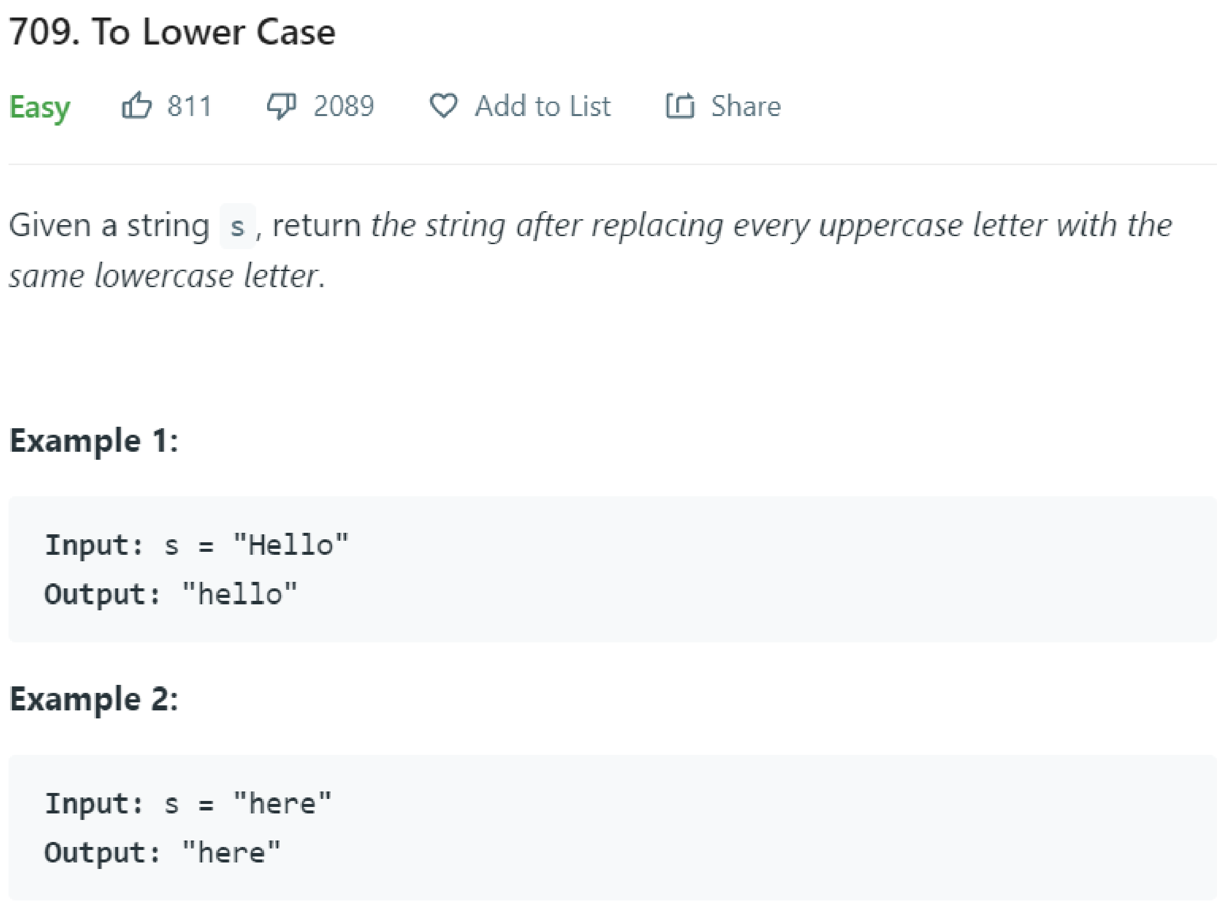
Example 1:

```
Input: order = "cba", s = "abcd"
Output: "cbad"
Explanation:
"a", "b", "c" appear in order, so the order of "a", "b", "c" should
be "c", "b", and "a".
Since "d" does not appear in order, it can be at any position in
the returned string. "dcba", "cdba", "cbda" are also valid outputs.
```



```cpp
class Solution {
public:
        string customSortString(string order, string str) {
                map<char, int>mp;
//Hash function to store string str.
                string ans = "";
// To store the answer.

                for(auto x:str)
                        mp[x]++;
// This loop will insert all the elements of the str in map(or hash).
                for(auto x:order){
// Traversing through the order.
                        if(mp.find(x)!= mp.end()){
// If x is present in the map ant it is not the last element of map.
                                auto temp = mp.find(x);
// Iterating in map untill x is present.
                                int count = temp->second;
// Moving the count pointer to the next of its present value.
                                string s(count, x);
// This line created a string name "s" and strored "count" number of x in it.
                                ans+=s;
// Inserting s(all the x) in the answer.
                                mp.erase(x);
// Erasing all the values of x from map.
                        }
                }
                for(auto x:mp){
                        string s(x.second, x.first);
                        ans+=s;
        }
        return ans;

        }

};
```

23-To lower case-



709. To Lower Case

Easy 811 2089 Add to List Share

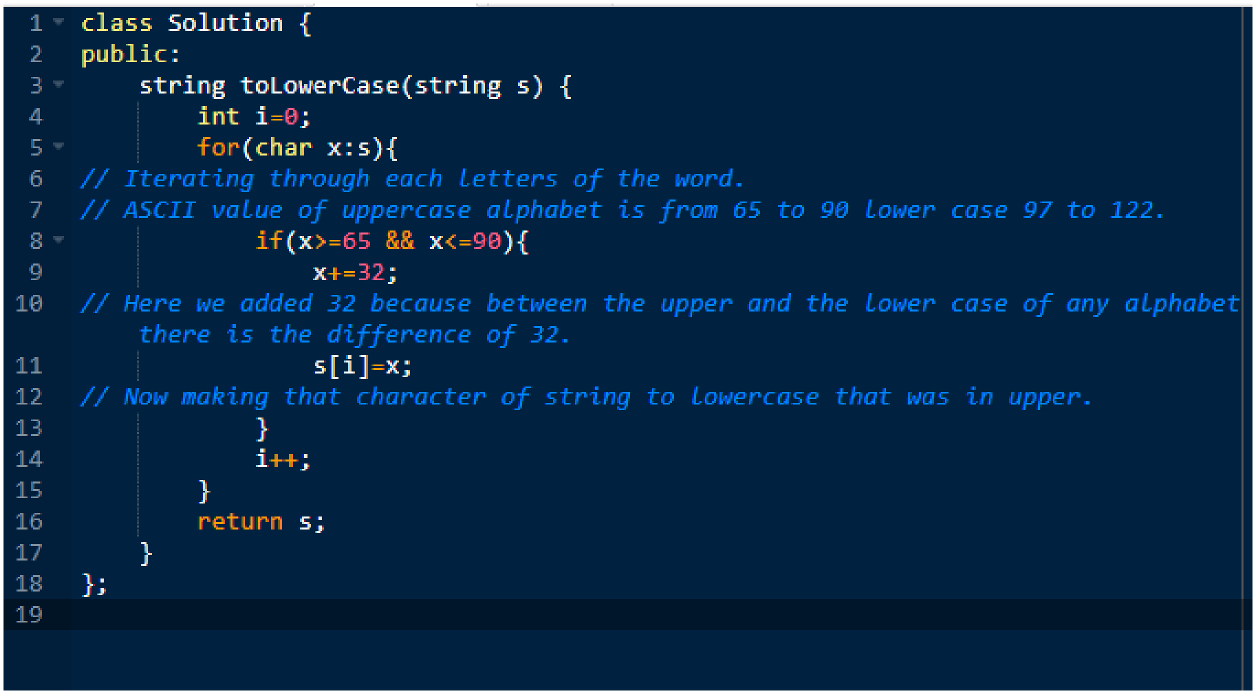
Given a string `s`, return *the string after replacing every uppercase letter with the same lowercase letter.*

Example 1:

```
Input: s = "Hello"
Output: "hello"
```

Example 2:

```
Input: s = "here"
Output: "here"
```



```
class Solution {
public:
    string toLowerCase(string s) {
        int i=0;
        for(char x:s){
// Iterating through each letters of the word.
// ASCII value of uppercase alphabet is from 65 to 90 lower case 97 to 122.
            if(x>=65 && x<=90){
                x+=32;
// Here we added 32 because between the upper and the lower case of any alphabet
    there is the difference of 32.
                s[i]=x;
// Now making that character of string to lowercase that was in upper.
            }
            i++;
        }
        return s;
    }
};
```

24-Unique characters of a strings-

387. First Unique Character in a String

Easy 3422 165 Add to List Share

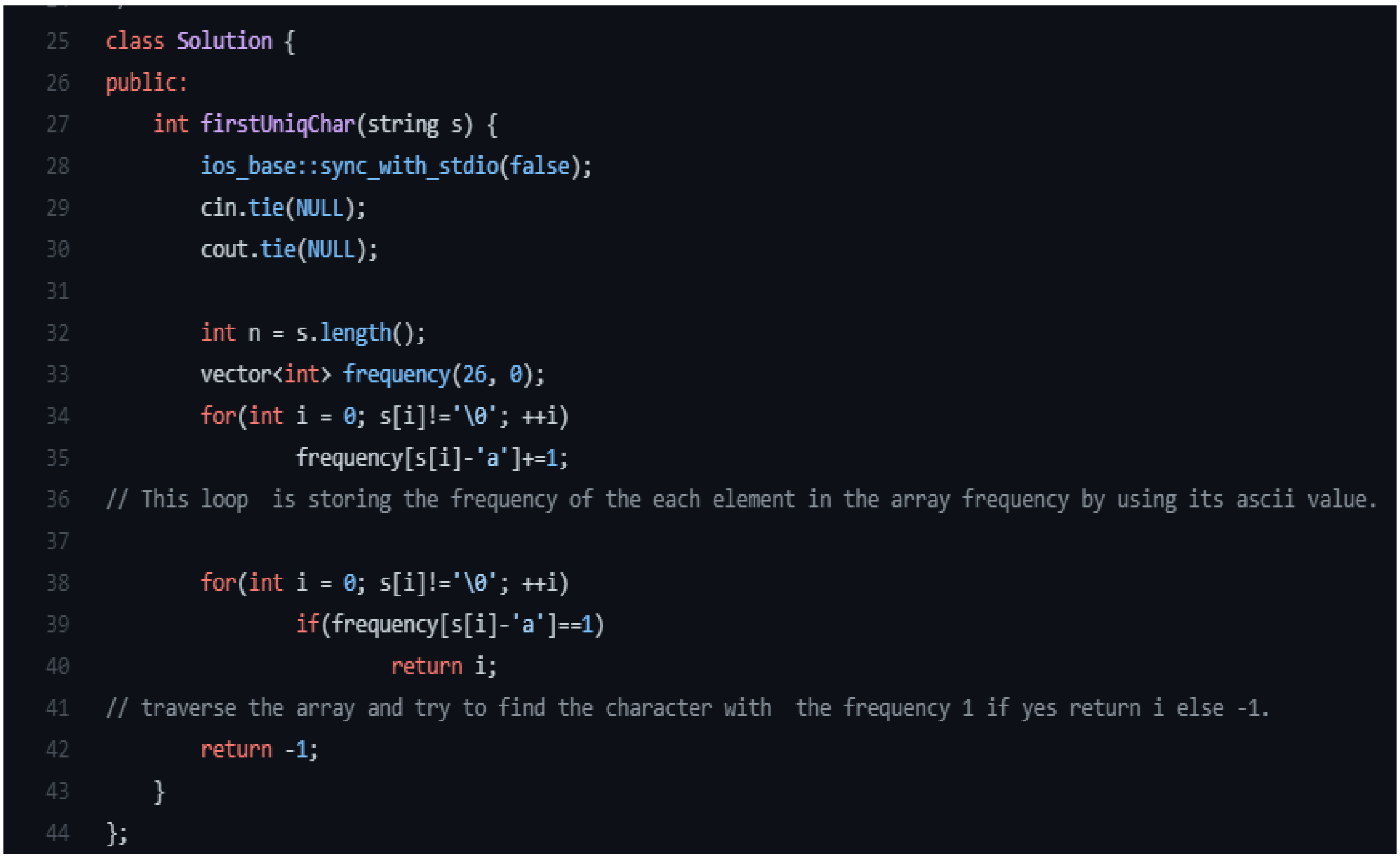
Given a string `s`, *find the first non-repeating character in it and return its index*. If it does not exist, return `-1`.

Example 1:

```
Input: s = "leetcode"
Output: 0
```

Example 2:

```
Input: s = "loveleetcode"
Output: 2
```



```cpp
class Solution {
public:
    int firstUniqChar(string s) {
        ios_base::sync_with_stdio(false);
        cin.tie(NULL);
        cout.tie(NULL);

        int n = s.length();
        vector<int> frequency(26, 0);
        for(int i = 0; s[i]!='\0'; ++i)
                frequency[s[i]-'a']+=1;
// This loop  is storing the frequency of the each element in the array frequency by using its ascii value.

        for(int i = 0; s[i]!='\0'; ++i)
                if(frequency[s[i]-'a']==1)
                        return i;
// traverse the array and try to find the character with  the frequency 1 if yes return i else -1.
        return -1;
    }
};
```