FREEMIND

Date

Page

Lecture 48

(Strings)

Q1 What are strings?

```
char ch = 'a';
          abcd
```

→ objects of a class std: string in C++.

→ Used to represent sequence of characters.

```
string string_name = "abcd"
```

⇒ **Initialising**

```
string str = "college";
string str1("Wallah");
```

⇒ **Taking Input**

- `string str;`
  `cin>> str;` // Single Word Input
- `getline(cin, str);` // Line Input

(for both) → `cout << str << endl`

⇒ Indexing of characters in a string

⇒ similar to arrays

str = C O L L E G E \0 → null character

0 1 2 3 4 5 6 7

str[3] = L

⇒ **ASCII Values**:- Character has a numeric value

A - Z a - z , *, !, #, $

A → 65 a → 97

B → 66 b → 98

⋮

Z → 90 z → 122

FREEMIND

Date

Page

(A) String V/s Character Array

String	Character Array
→ string is a class string variables → objects of this class.	→ array of char data type
→ string str_name	→ char arr_name [size]
→ dynamic memory Allocation	→ static memory allocation
→ no pre-allocated memory	→ unused allocated memory is wasted
→ have inbuilt functions	→ faster

→ Commonly used inbuilt functions :-

(1) reverse() → reverse a str. from starting ptr to end ptr.
→ string str = "abcd";
reverse (str.began(), str.end())
• Time complexity = O (length of string)

(2) substr() → to find substring of a given string
"Hello" substring "Hel"
→ S.substr (position, length)
str.substr (0, 3);
→ If length not provided then whole string printed from that position
• Time complexity → O(length)

(3) The "+" operator
↳ concatenate string
"College" + "Wallah"
= CollegeWallah
→ S1 = S1 + S2
• Copy string created for S1
• extra space for creating copy

→ S1 += S2
S2 is getting appended after S1.

FREEMIND

Date

Page

④ strcat() → Used to Concatenate Character arrays.

```
char S1[20] = "College";
Char S2[20] = "Wallah";
→ strcat(S1, S2);
→ Cout << S1 << endl;  // CollegeWallah
```

⑤ push_back() → insert character at the end of string

```
push_back(Char);
```

⑥ size()

```
str.size()
str.length()
```
→ for string. O(1)

```
→ char ch[20];
→ strlen(ch);
```
→ for Char array. O(n)

⑦ to_string() → to convert numeric value to string

```
int num = 4;
to_string(num);  // → "4"
```

Q1) Given a Strings str, sort the given string.

Constraints:- The string will contain only alphabatical characters from a-z.

input :- "CodingWallah"

output :- "aacdghillnow"

Time complexity → O(n)

Space complexity → O(1)

Ans → Count Sort

FREEMIND

Date

Page

Sol:

```
string countSort(string str) {

    Vector<int> freq(26,0)
    // storing freq. of every char. in a string.
    for (i = 0; i < str.length(); i++) {
        int index = str[i] - 'a';
        freq[index]++;
    }
    // Create our sorted string.
    int j = 0;
    for (int i = 0; i < 26; i++)
        while (freq[i]--) {
            str[j++] = i + 'a';
        }
    }
    return str;
}
int main() {
    string str;
    cin>> str;
    cout << countSort(str) << endl;
    return
}
```

Q] Given two strings s and t, return true if t is an anagram of s and false otherwise.

Constraints :- String s and t will only contain lowercase alphabetical characters.

input :- s = "anagram", t = ~~"nagaram"~~ "nagaram"

output :- yes

① Same letters
② Same count of letters.

FREEMIND

Date

Page

Method 1 ⇒ ① Sort Both Strings
② If equal → Yes, anagram.

Method 2 ⇒ ① For string ①
Do freq.[ ] ++
② For string ②
Do freq.[ ] --
if all element of freq.[] are 0 then → Yes, anagram.

```
#include <iostream>
#include <string>
#include <vector>
Using namespace std;

bool isAnagram(String S1, String S2) {
    //Creat freq. array.
    Vector<int> freq(26,0)
    //if lengths are diff. return false
    if (S1.length() != S2.length()) {
        return false;
    }
    //Store freq. of Char in S1, S2
    for (int i=0; i<S1.length(); i++) {
        freq [S1[i] - 'a'] ++; // For S1, incrementing freq of ch.
        freq [S2[i] - 'a'] --; // For S2, decrementing freq of character.
    }
    for (int i=0; i<26; i++) { //Checking if freq of every char. is 0;
        if (freq[i] != 0) {
            return false; //Not anagram
        }
    }
    return true;
}
```

(ASCII Values)

```
int main() {
    S1, S2 -> enter
    if (isAnagram(S1, S2)) {
        cout << "Strings are Anagram" << endl;
    } else {
        cout << "Not Anagram" << endl;
    }
}
```

Time complexity → O(length)
Space complexity → O(26) - O(1)

Q3) Given two strings s and t, determine if they are isomorphic.

(1.20) input :- s = "egg", t = "add"
output :- True

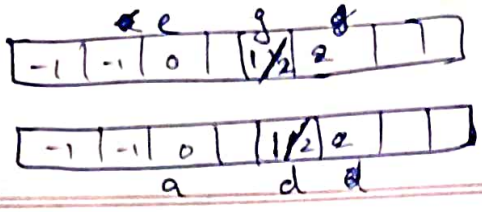
Sol⇒

```
bool isIsomorphic(string s1, string s2) {
    Vector<int> V1(128, -1);
    Vector<int> V2(128, -1);

    if (S1.size() != S2.size()) {
        return false;
    }
    for (int i=0; i<S1.size(); i++) {
        if (V1[S1[i]] != V2[S2[i]]) {
            return false;
        }
        V1[S1[i]] = V2[S2[i]] = i
    }
    return true;
}
```

Space complexity O(256)

Time complexity O(n)



FREEMIND

Date

Page

```
int main () {
    S1, S2 → enter
    if (isIsomorphic (S1, S2)) {
        cout << "Yess Isomorphic" << endl;
    } else {
        cout << "No" << endl;
```

Ques Given an array of strings. Write a program to find the longest common prefix string amongst an array of strings.

input :- arr = ["flower", "flight", "flask"]

output :- "fl"

Ans →

```
string LongestCommonPrefix (vector<string> &str) {
    sort (str.begin(), str.end());
    string S1 = str[0];
    int i = 0;
    string S2 = str[str.size()-1];
    int j = 0;
    string ans = "";
    while (i < S1.size() && j < S2.size()) {
        if (S1[i] == S1[j])
            ans += S1[i];
            i++; j++;
        } else {
            break;
        }
    }
    return ans;
}
```

FREEMIND

Date

Page

```
int main () {
    int n;
    cout << "Enter no. of strings";
    cin >> n;
    Vector < string > str;
    for( i = 0 ; i < n ; i++ ) {
        cin >> str[i]
    }
    cout << longest common prefix: << longestCommonPrefix (str)
                                                << endl;
}
```

- Time complexity => $O(n \log n * m) + O(\min(S[0].size(), S[n-1].size()))$

Method 2: Keeping first string as const. and comparing it with all other strings

```
string longestCommonPrefix ( Vector < string > & str) {

    string S1 = str[0];
    int ans_length = S1.size();

    for(int i = 0; i < str.size(); i++)
    int j = 0
    while ( j < S1.size() && j < str[1].size() && S1[j] == str[i][j]
        j++;
        ans_length = min(ans_length, j);
    }
    string ans = S1.substr (0, ans_length)
    return ans;
}
int main () {
    ~~~ Same Previous
}
```

Time Complexity → O(n-1 * m)
O(n * m)

Q5 → An encoded string (s) is given and the task is to decode it. The encoding pattern is that the occurance of the string is given at the starting of the string and such string is enclose by square brackets.

Note :- The occurance of a single string is less than 1000.

input :- S = 3[b2[ca]]

output :- bcacabcacabcacabcacabcaca

Concept

```
for (i=0 to i2 s.length())
    if (S[i] != ']') {
        insert into result;
    } else {
        ① Extract str from result
        ② Reverse str
        ③ Remove last char
        ④ Extract digit/num from result
            till S[i] >= '0' && S[i] <= '9'
        ⑤ Reverse num
        ⑥ Convert num string to int
        ⑦ Insert str in result, int num times;
```

S = 3[b2[ca]]
result = "3[bcaca"
str = "ca" } "bcaca"
num = "2" } "3"
int num = stoi(num)

```
int main () {
    string str;
    cin>> str;
    cout << DecodedString(str) << endl;
    return 0;
}
```

FREEMIND

Date

Page

Leet 22

```
string DecodedString(string s) {
    string result = "";
    // traversing the encoded string.
    for (int i=0 ; i < s.size(); i++)
        if (s[i] != ']") {
            result.push_back(s[i])
        }
        else {
            //extract str from result.
            string str="";
            while (!result.empty() && result.back()!= '[')
                str.push_back(result.back());
                result.pop_back();
            }
            //reversing the str.
            reverse(str.began(), str.end());
            //remove last char. from result which is [
            result.pop_back();
            // Extract num from result.
            string num = "";
            while (!result.empty() && result.back() > '0'
                                   && result.back() < '9') {
                num.pushback(result.back());
                result.pop_back;
            }
            //reversing the num string
            reverse(num.began(), num.end())
            //convert string to integer
            int int_num = stoi(num);
            // inserting str in result int_num times
            while(int_num) {
                result += str;
                int_num--;
            }
        }
    }
```

Space complexity O(n)

Time complexity O(n)

n is length of Decoded string

Imp.

Date
Page

return result
}

Q 6 → Given a binary string and an integer k, return the maximum no. of consecutive 1's in the string if you can flip at most k 0's.
input :- "0001101011", k=2 / To find longest substring with max k 0's
output :- 7

Sol → Method 1 (Brute Force) →

```
for (i = 0 to i < s.length())
    for (j = i to j < s.length())
        $ [i ... j] ⟶ 0's <= k
            length i ... j
```

Sub-String i to j

Method 2 → Sliding Window technique
Used to find longest/shortest sequence with some given condition.

start
↓
0 0 0 1 1 0 1 0 1 1
↑ end

Zero count = 2
Max_length = 7

```
while (zero count > k) {
    if (str[s] == '0') zero count --
    s++
}
```

```
int main () {
    Cout << Enter binary string
    Enter → str.
    Cout << Enter max flips
    Enter → k
    Cout << longestOnes(str, k)
    << endl;
    return 0;
}
```

~~int main () {~~

Time complexity → O(n) } n is length of binary string
Space → O(1)

FREEMIND

Date

Page

```
#include <iostream>
using namespace std;

int longestOnes(string str, int k) {
    int start = 0;
    int end = 0;
    int zero_count = 0;
    int max_length = 0;

    for (; end < str.length(); end++) {
        if (str[end] == '0') {
            zero_count++
        }
        while (zero_count > k)
            if (str[start] == '0') zero_count--;
            start++; //contracting our window
        }
        max_length = max(max_length, end - start + 1);
    }
}
//zero_count <= k
```