STRINGS

1. INTRODUCTION

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

int main()
{
    string str="geeksforgeeks";
    int count[26]={0};
    for(int i=0;i<str.length();i++){
        count[str[i]-'a']++;
    }
    for(int i=0;i<26;i++){
        if(count[i]>0){
            cout<<char(i+'a')<<" "<<count[i]<<endl;
        }
    }
}
```

2. STRING IN C++

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

int main()
{
    string str = "geeksforgeeks";
    for (int i = 0; i < str.length(); i++)
        cout << str[i];
    cout << endl;
```

```
    for (char x: str)
        cout << x;
}

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

int main()
{
    string str;
    cout << "Enter your name";
    getline(cin, str, 'a');
    cout << "\nYour name is " << str;
    return 0;
}

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

int main()
{
    string str;
    cout << "Enter your name";
    getline(cin, str);
    cout << "\nYour name is " << str;
    return 0;
}

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

int main()
```

```
{
    string str;
    cout << "Enter your name";
    cin >> str;
    cout << "\nYour name is " << str;
    return 0;
}
```

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

int main()
{
    string str;
    cout << "Enter your name";
    cin >> str;
    cout << "\nYour name is " << str;
    return 0;
}
```

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

int main()
{
    char s1[] = "abc";
    char s2[] = "bcd";
    if (s1 == s2)
        cout << "Same";
    else if(s1 < s2)
        cout << "Smaller";
    else
        cout << "Greater";
```

```
}

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

int main()
{
    string str = "geeksforgeeks";
    cout << str.length() << " ";
    str = str + "xyz";
    cout << str << " ";
    cout << str.substr(1, 3) << " ";
    cout << str.find("eek") << " ";
    return 0;
}

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

int main()
{
    char str[5];
    strcpy(str, "gfg");
    cout << str;
    return 0;
}

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

int main()
{
```

```
    char s1[] = "bcd";
    char s2[] = "bcd";
    int res = strcmp(s1, s2);
    if (res > 0)
        cout << "Greater";
    else if(res == 0)
        cout << "Same";
    else
        cout << "Smaller";
}

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

int main()
{
    char s1[] = "gfg";
    char s2[] = "bcd";
    int res = strcmp(s1, s2);
    if (res > 0)
        cout << "Greater";
    else if(res == 0)
        cout << "Same";
    else
        cout << "Smaller";
}

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

int main()
{
    char s1[] = "abc";
```

```
    char s2[] = "bcd";
    int res = strcmp(s1, s2);
    if (res > 0)
        cout << "Greater";
    else if(res == 0)
        cout << "Same";
    else
        cout << "Smaller";
}
```

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

int main()
{
    char str[] = {'g', 'f', 'g', '\0'};
    cout << str;
    return 0;
}
```

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

int main()
{
    char str[] = {'g', 'f', 'g'};
    cout << str;
    return 0;
}
```

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

int main()
{
```

```
    char str[] = "gfg";
    cout << sizeof(str);
    return 0;
}

#include<iostream>
using namespace std;

int main()
{
    char str[] = "gfg";
    cout << str;
    return 0;
}
```

3. **PALINDROME CHECK**

4. **CHECK IF STRING IS SUBSEQUENCE OF OTHER**

   ITERATIVE:

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

bool isSubSeq(string s1, string s2, int n, int m){
    int j = 0;

    for(int i = 0; i < n && j < m; i++){
        if(s1[i] == s2[j])
        j++;
    }

    return j == m;
}
```

```
int main() {
        int n,m;
        string s1, s2;
        cin>>n>>m;
        cin>>s1>>s2;

        cout<<boolalpha<<isSubSeq(s1, s2, n, m);

        return 0;
}
```

RECURSIVE:

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

bool isSubSeq(string s1, string s2, int n, int m){
   if ( m == 0 )
      return true;

   if ( n == 0 )
      return false;

   if ( s1[n-1] == s2[m-1] )
      return isSubSeq(s1, s2, n-1, m-1);

   else
      return isSubSeq(s1, s2, n-1, m);
}

int main() {
        int n,m;
        string s1, s2;
        cin>>n>>m;
```

```
        cin>>s1>>s2;

        cout<<boolalpha<<isSubSeq(s1, s2, n, m);

        return 0;
}
```

5. CHECK FOR ANAGRAM

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

const int CHAR=256;
bool areAnagram(string &s1, string &s2)
{
   int n1 = s1.length();
   int n2 = s2.length();
   if (n1 != n2)
      return false;

   int count[CHAR]={0};
   for(int i=0;i<s1.length();i++){
      count[s1[i]]++;
      count[s2[i]]--;
   }

   for(int i=0;i<CHAR;i++){
      if(count[i]!=0)return false;
   }
   return true;
}

int main()
{
```

```
    string str1 = "abaac";
    string str2 = "aacba";
    if (areAnagram(str1, str2))
        cout << "The two strings are anagram of each other";
    else
        cout << "The two strings are not anagram of each other";

    return 0;
}
```

6. LEFTMOST REPEATING CHARACTER

METHOD 1:

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

const int CHAR=256;
int leftMost(string &str)
{
    int fIndex[CHAR]={0};
    fill(fIndex,fIndex+CHAR,-1);
    int res=INT_MAX;
    for(int i=0;i<str.length();i++){
        int fi=fIndex[str[i]];
        if(fi==-1)
        fIndex[str[i]]=i;
        else
        res=min(res,fi);
    }

    return (res==INT_MAX)?-1:res;
}
```

```
int main()
{
   string str = "geeksforgeeks";
   cout<<"Index of leftmost repeating character:"<<endl;
   cout<<leftMost(str)<<endl;

   return 0;
}
```

METHOD 2:

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

const int CHAR=256;
int leftMost(string &str)
{
   bool visited[CHAR];
   fill(visited,visited+CHAR,false);
   int res=-1;
   for(int i=str.length()-1;i>=0;i--){
      if(visited[str[i]])
      res=i;
      else
      visited[str[i]]=true;
   }

   return res;
}

int main()
{
   string str = "geeksforgeeks";
   cout<<"Index of leftmost repeating character:"<<endl;
   cout<<leftMost(str)<<endl;
```

```
    return 0;
}
```

7. LEFTMOST NON-REPEATING CHARACTER

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

const int CHAR=256;
int nonRep(string &str)
{
    int fl[CHAR];
    fill(fl,fl+CHAR,-1);

    for(int i=0;i<str.length();i++){
        if(fl[str[i]]==-1)
        fl[str[i]]=i;
        else
        fl[str[i]]=-2;
    }
    int res=INT_MAX;
    for(int i=0;i<CHAR;i++){
        if(fl[i]>=0)res=min(res,fl[i]);
    }
    return (res==INT_MAX)?-1:res;
}

int main()
{
    string str = "geeksforgeeks";
    cout<<"Index of leftmost non-repeating element:"<<endl;
    cout<<nonRep(str)<<endl;
```

```
    return 0;
}
```

8. REVERSE WORDS IN A STRING

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

void reverse(char str[],int low, int high){
    while(low<=high){
        swap(str[low],str[high]);
        low++;
        high--;
    }
}

void reverseWords(char str[],int n){
    int start=0;
    for(int end=0;end<n;end++){
        if(str[end]==' '){
            reverse(str,start,end-1);
            start=end+1;
        }
    }
    reverse(str,start,n-1);
    reverse(str,0,n-1);
}

int main()
{
    string s = "Welcome to Gfg";int n=s.length();char str[n];
    strcpy(str, s.c_str());
    cout<<"After reversing words in the string:"<<endl;
    reverseWords(str,n);
```

```
    for (int i = 0; i < n; i++)
        cout << str[i];

    return 0;
}
```

9. PATTERN SEARCHING

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

void patSearchinng(string &txt,string &pat){
    int m=pat.length();
    int n=txt.length();
    for(int i=0;i<=(n-m);i++){
        int j;
        for(j=0;j<m;j++)
        if(pat[j]!=txt[i+j])
        break;

        if(j==m)
        cout<<i<<" ";
    }
}

int main()
{
    string txt = "ABCABCD";string pat="ABCD";
    cout<<"All index numbers where pattern found:"<<" ";
    patSearchinng(txt,pat);

    return 0;
}
```

10. PATTERN SEARCHING FOR DISTINCT

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

void patSearchinng(string &txt,string &pat){
    int m=pat.length();
    int n=txt.length();
    for(int i=0;i<=(n-m);  ){
        int j;
        for(j=0;j<m;j++)
        if(pat[j]!=txt[i+j])
        break;

        if(j==m)
        cout<<i<<" ";
        if(j==0){
        i++;}
        else{
        i=(i+j);}
    }
}

int main()
{
    string txt = "ABCABCD";string pat="ABCD";
    cout<<"All index numbers where pattern found:"<<" ";
    patSearchinng(txt,pat);

    return 0;
}
```

11. RABIN KARP ALGORITHM

```
#include <bits/stdc++.h>
using namespace std;
#define d 256
const int q=101;
void RBSearch(string pat,string txt,int M, int N){

    //Compute (d^(M-1))%q
    int h=1;
    for(int i=1;i<=M-1;i++)
        h=(h*d)%q;

    //Compute p and to
    int p=0,t=0;
    for(int i=0;i<M;i++){
        p=(p*d+pat[i])%q;
        t=(t*d+txt[i])%q;
    }

    for(int i=0;i<=(N-M);i++){
      //Check for hit
      if(p==t){
          bool flag=true;
          for(int j=0;j<M;j++)
              if(txt[i+j]!=pat[j]){flag=false;break;}
           if(flag==true)cout<<i<<" ";
      }
      //Compute ti+1 using ti
      if(i<N-M){
          t=((d*(t-txt[i]*h))+txt[i+M])%q;
       if(t<0)t=t+q;
      }
    }

}
```

```
int main()
{
    string txt = "GEEKS FOR GEEKS";string pat="GEEK";
    cout<<"All index numbers where pattern found:"<<" ";
    RBSearch(pat,txt,4,15);

    return 0;
}
```

12. CONSTRUCTING LPS ARRAY

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


void fillLPS(string str, int *lps){
    int n=str.length(),len=0;
    lps[0]=0;
    int i=1;
    while(i<n){
        if(str[i]==str[len])
        {len++;lps[i]=len;i++;}
        else
        {if(len==0){lps[i]=0;i++;}
            else{len=lps[len-1];}
        }
    }
}

int main()
{
    string txt = "abacabad";int lps[txt.length()];
    fillLPS(txt,lps);
    for(int i=0;i<txt.length();i++){
```

```
        cout<<lps[i]<<" ";
    }

    return 0;
}
```

13. KMP STRING MATCHING

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


void fillLPS(string str, int *lps){
    int n=str.length(),len=0;
    lps[0]=0;
    int i=1;
    while(i<n){
        if(str[i]==str[len])
        {len++;lps[i]=len;i++;}
        else
        {if(len==0){lps[i]=0;i++;}
            else{len=lps[len-1];}
        }
    }
}
void KMP(string pat,string txt){
    int N=txt.length();
    int M=pat.length();
    int lps[M];
    fillLPS(pat,lps);
    int i=0,j=0;
    while(i<N){
        if(pat[j]==txt[i]){i++;j++;}
```

```
        if (j == M) {
            printf("Found pattern at index %d ", i - j);
            j = lps[j - 1];
        }
        else if (i < N && pat[j] != txt[i]) {
            if (j == 0)
                i++;
            else
                j = lps[j - 1];
        }
    }
}

int main()
{
    string txt = "ababcababaad",pat="ababa";
    KMP(pat,txt);
    return 0;
}
```

14. CHECK IF STRINGS ARE ROTATIONS

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

bool areRotations(string s1,string s2){
    if(s1.length()!=s2.length())return false;
    return ((s1+s1).find(s2)!=string::npos);
}

int main()
{
    string s1 = "ABCD";string s2="CDAB";
```

```
    if(areRotations(s1,s2)){cout<<"Strings are rotations of each
other"<<endl;}
    else{cout<<"Strings are not rotations of each other"<<endl;}

    return 0;
}
```

15. ANAGRAM SEARCH

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

const int CHAR=256;

bool areSame(int CT[],int CP[]){
    for(int i=0;i<CHAR;i++){
        if(CT[i]!=CP[i])return false;
    }
    return true;
}

bool isPresent(string &txt,string &pat){
    int CT[CHAR]={0},CP[CHAR]={0};
    for(int i=0;i<pat.length();i++){
        CT[txt[i]]++;
        CP[pat[i]]++;
    }
    for(int i=pat.length();i<txt.length();i++){
        if(areSame(CT,CP))return true;
        CT[txt[i]]++;
        CT[txt[i-pat.length()]]--;
    }
    return false;
}
```

```
int main()
{
    string txt = "geeksforgeeks";
    string pat = "frog";
    if (isPresent(txt,pat))
        cout << "Anagram search found";
    else
        cout << "Anagram search not found";

    return 0;
}
```

16. LEXICOGRAPHIC RANK OF A STRING

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

const int CHAR=256;
int fact(int n)
{
    return (n <= 1) ? 1 : n * fact(n - 1);
}

int lexRank(string &str)
{
    int res = 1;
    int n=str.length();
    int mul= fact(n);
    int count[CHAR]={0};
    for(int i=0;i<n;i++)
        count[str[i]]++;
    for(int i=1;i<CHAR;i++)
        count[i]+=count[i-1];
```

```
    for(int i=0;i<n-1;i++){
        mul=mul/(n-i);
        res=res+count[str[i]-1]*mul;
        for(int j=str[i];j<CHAR;j++)
            count[j]--;
    }
    return res;
}


int main()
{
    string str = "STRING";
    cout << lexRank(str);
    return 0;
}
```

17. LONGEST SUBSTRING WITH DISTINCT CHARACTERS

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

int longestDistinct(string str)
{
        int n = str.length();
        int res = 0;
        vector<int> prev(256,-1);
        int i=0;
        for (int j = 0; j < n; j++){
            i=max(i,prev[str[j]]+1);
            int maxEnd=j-i+1;
            res=max(res,maxEnd);
            prev[str[j]]=j;
        }
```

```
        return res;
}

int main()
{
        string str = "geeksforgeeks";
        int len = longestDistinct(str);
        cout << "The length of the longest distinct characters substring
is "<< len;
        return 0;
}
```