**Code-1:**

class Solution {

public:

    string removeOuterParentheses(string s)

    {

        string ans;

        int cnt = 0;

        for(char ch:s)

        {

            if(ch == '(')

            {

                if(cnt) ans += ch;

                cnt++;

            }

            else

            {

                cnt--;

                if(cnt) ans += ch;

            }

        }

        return ans;

    }

};

**Code-2:**

class Solution {

public:

    string reverseWords(string s) {

       reverse(s.begin(), s.end());

        int n = s.size();

        int left = 0;

        int right = 0;

        int i = 0;

        while (i < n) {

            while (i < n && s[i] == ' ')

                i++;

            if (i == n)

                break;

            while (i < n && s[i] != ' ') {

                s[right++] = s[i++];

            }

            reverse(s.begin() + left, s.begin() + right);

            s[right++] = ' ';

            left = right;

            i++;

        }

        s.resize(right - 1);

        return s;

    }

};

**Code-3:**

class Solution {

public:

    string largestOddNumber(string num) {

     for(int i=num.size()-1;i>=0;i--){

            if((num[i]-'0')%2!=0){//Odd

                return num.substr(0,i+1);

            }

        }

        return "";

    }

};

**Code-4:**

class Solution {

public:

    string longestCommonPrefix(vector<string>& strs) {

        string ans="";

        sort(strs.begin(),strs.end());

        int n = strs.size();

         string first=strs[0],last=strs[n-1];

        for(int i=0;i<min(first.size(),last.size());i++){

            if(first[i]!=last[i]){

                return ans;

            }

            ans+=first[i];

        }

        return ans;

    }

};

**Code-5:**

class Solution {

public:

    bool isIsomorphic(string s, string t) {

     unordered\_map<char, char> mp;

        string s1;

        char i = 'a';

        for (auto x: s)

        {

            if (mp.find(x) == mp.end())

            {

                s1 += i;

                mp[x] = i;

                i++;

            }

            else

            {

                s1 += mp[x];

            }

        }

        mp.clear();

        string s2;

        i = 'a';

        for (auto x: t)

        {

            if (mp.find(x) == mp.end())

            {

                s2 += i;

                mp[x] = i;

                i++;

            }

            else

            {

                s2 += mp[x];

            }

        }

        return s1 == s2;

    }

};

**Code-6:**

class Solution {

public:

    bool rotateString(string s, string goal) {

       if (s.length() == 1) {

            return true ;

        }

        if (s.length() != goal.length()) {

            return false ;

        }

        int count = 0 ;

        while (count < s.length()) {

            s = s.substr(1) + s[0] ;

            if (s == goal) {

                return true ;

            }

            count++ ;

        }

        return false ;

    }

};

**Code-7:**

class Solution {

public:

    bool isAnagram(string s, string t) {

        sort(s.begin(), s.end());

        sort(t.begin(), t.end());

        return s == t;

    }

};

**Code-8:**

class Solution {

public:

    string frequencySort(string s) {

        unordered\_map<char, int> freq;

        for (char c : s) {

            freq[c]++;

        }

        vector<pair<int, char>> freq\_vec;

        for (auto& kv : freq) {

            freq\_vec.push\_back({kv.second, kv.first});

        }

        sort(freq\_vec.rbegin(), freq\_vec.rend());

        string sorted\_str;

        for (auto& p : freq\_vec) {

            sorted\_str += string(p.first, p.second);

        }

        return sorted\_str;

    }

};

**Code-9:**

class Solution {

public:

    int maxDepth(string s) {

        int cnt=0;

        int ans=0;

        for(auto x:s){

            if(x=='(') cnt++;

            else if(x==')') cnt--;

            ans=max(ans,cnt);

        }

        return ans;

    }

};

**Code-10:**

class Solution {

public:

    int romanToInt(string s) {

      unordered\_map<char, int> m;

        m['I'] = 1;

        m['V'] = 5;

        m['X'] = 10;

        m['L'] = 50;

        m['C'] = 100;

        m['D'] = 500;

        m['M'] = 1000;

        int ans = 0;

        for(int i = 0; i < s.length(); i++){

            if(m[s[i]] < m[s[i+1]]){

                ans -= m[s[i]];

            }

            else{

                ans += m[s[i]];

            }

        }

        return ans;

    }

};

**Code-11:**

class Solution {

public:

    int myAtoi(string s) {

        long ans = 0;

        int sign = 1;

        int i = 0;

        while (s[i] == ' ') i++;

        if (s[i] == '-' || s[i] == '+') {

            sign = (s[i] == '-') ? -1 : 1;

            i++;

        }

        while (i < s.length() && isdigit(s[i])) {

            ans = ans \* 10 + (s[i] - '0');

            if (ans\*sign <= INT\_MIN) return INT\_MIN;

            if (ans\*sign >= INT\_MAX) return INT\_MAX;

            i++;

        }

        return ans \* sign;

    }

};

**Code-12:**

private:  
    long long int helper(string s, int k){  
        vector<int> mpp(26, 0);  
        long long int count = 0;  
        int left = 0;  
        int distinct = 0;  
          
        for(int right = 0; right < s.size(); right++){  
            mpp[s[right]-'a']++;  
            if(mpp[s[right]-'a'] == 1) distinct++;  
              
            while(distinct > k){  
                mpp[s[left]-'a']--;  
                if(mpp[s[left]-'a'] == 0) distinct--;  
                left++;  
            }  
              
            count += (right - left + 1);  
        }  
        return count;  
    }  
public:  
    long long int substrCount (string s, int k) {  
        //code here.  
        return helper(s, k) - helper(s, k-1);  
    }

**Code-13:**

class Solution {

public:

    string longestPalindrome(string s) {

        int start = 0, end = 0;

        for(int i =0; i<s.length(); i++){

            int len1 = expandCenter(s, i, i);

            int len2 = expandCenter(s, i, i+1);

            int len = max(len1, len2);

            if(len>end-start){

                start = i - (len-1)/2;

                end = i+len/2;

            }

        }

        return s.substr(start, end-start+1);

    }

    int expandCenter(string s, int left, int right){

        while(left>=0 && right<s.length() && s[left]==s[right]){

            left--;

            right++;

        }

        return right-left -1;

    }

};

**Code-14:**

class Solution {

public:

    int beautySum(string s) {

      int sum = 0; //  to store ans;

        int n = s.length();

        for (int i = 0; i < n; i++) {

            unordered\_map<char, int> mp;

            for (int j = i; j < n; j++) {

                mp[s[j]]++; // storing every possible substring

                int maxFreq = 0, minFreq = INT\_MAX;

                //<!--finding beauty of substing -->

                for (auto it : mp) {

                    maxFreq = max(maxFreq, it.second);

                    minFreq = min(minFreq, it.second);

                }

                sum += (maxFreq - minFreq);

            }

        }

        return sum;

    }

};

**Code-15:**

class Solution {

public:

    int beautySum(string s) {

      int sum = 0; //  to store ans;

        int n = s.length();

        for (int i = 0; i < n; i++) {

            unordered\_map<char, int> mp;

            for (int j = i; j < n; j++) {

                mp[s[j]]++; // storing every possible substring

                int maxFreq = 0, minFreq = INT\_MAX;

                //<!--finding beauty of substing -->

                for (auto it : mp) {

                    maxFreq = max(maxFreq, it.second);

                    minFreq = min(minFreq, it.second);

                }

                sum += (maxFreq - minFreq);

            }

        }

        return sum;

    }

};