Given a string ss, return the longest palindromic substring in ss  
  
#include <bits/stdc++.h>

using namespace std;

string longestpalind(string ans)

{   int maxlen=1;

     int start=0;

   int n=ans.size();

    for(int i=0;i<n;i++)

    {

         int left=i;

         int right=i;

         while(right+1<n && ans[right]==ans[right+1]) right++;

         i=right+1;

         while(left-1>=0 && right+1<n && ans[left-1]==ans[right+1])

         {

             left--;

             right++;

         }

    if(right-left+1>maxlen)

    {

        start=left;

        maxlen=right-left+1;

    }

    }

    return ans.substr(start,maxlen);

}

int main()

{

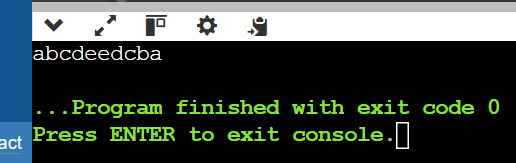
    string s="abcdeedcba";

    string fans=longestpalind(s);

     cout<<fans;

    return 0;

}

  
  
  
Given a string containing just the characters "(",")","{","}","[",and "]", determine if the input

string is valid.

#include <bits/stdc++.h>

using namespace std;

bool isValid(string str) {

stack<char> st;

for (char ch : str) {

  if (ch == '(' || ch == '{' || ch == '[') {

  st.push(ch);

   }

else {

  if (st.empty()) return false;

  char top = st.top();

  if ((ch == ')' && top != '(') ||

      (ch == '}' && top != '{') ||

      (ch == ']' && top != '[')) {

  return false;

  }

   st.pop();

   }

}

return st.empty();

}

int main()

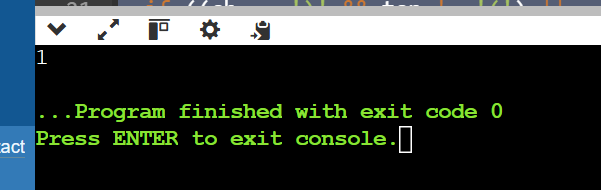
{

    string s="{}[][]()";

    cout<<isValid(s);

    return 0;

}

  
  
  
Given two strings ss and tt, determine if it is an anagram of a substring of ss. In other words, check if there exists a substring in ss that is an anagram of tt.  
  
  
#include <bits/stdc++.h>

using namespace std;

bool isanagram(string t, string s) {

   if (s.size() < t.size()) return false;

   vector<int> pre(26, 0), main(26, 0);

   for (char c : t) pre[c - 'a']++;

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

       main[s[i] - 'a']++;

       if (i >= t.size()) main[s[i - t.size()] - 'a']--;

       if (pre == main) return true;

   }

   return false;

}

int main()

{

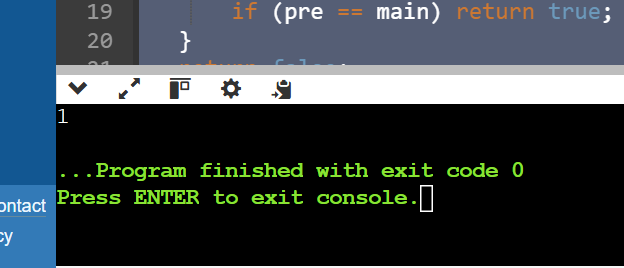
    string s="eat";

    string f="ate";

    cout<<isanagram(s,f);

    return 0;

}

  
  
  
  
  
  
Given two strings ss and rt, return the smallest substring in ss that contains all the characters in tt. If no such substring exists, return an empty string  
  
  
  
  
#include <bits/stdc++.h>

using namespace std;

string minWindow(string ss, string tt) {

   if (ss.length() < tt.length()) return "";

   unordered\_map<char, int> need, window;

   for (char ch : tt) need[ch]++;

   int left = 0, right = 0, valid = 0, start = 0, len = INT\_MAX;

   while (right < ss.size()) {

       char ch = ss[right++];

       if (need.count(ch)) {

           window[ch]++;

           if (window[ch] == need[ch]) valid++;

       }

       while (valid == need.size()) {

           if (right - left < len) {

               start = left;

               len = right - left;

           }

           char d = ss[left++];

           if (need.count(d)) {

               if (window[d] == need[d]) valid--;

               window[d]--;

           }

       }

   }

   return len == INT\_MAX ? "" : ss.substr(start, len);

}

int main() {

   string ss, tt;

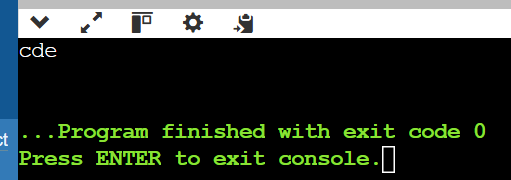
    ss="ancdefghijkl";

    tt="cde";

   cout <<minWindow(ss, tt) << endl;

   return 0;

}



Two strings are considered close if you can swap letters or change the frequency of any letter to match the other string. Determine if two given strings are close.  
  
#include <bits/stdc++.h>

using namespace std;

bool close(string str1, string str2) {

     if (str1.size() != str2.size()) return false;

    unordered\_set<char> s1, s2;

    for (char c : str1)  s1.insert(c);

    for (char c : str2)  s2.insert(c);

    return s1 == s2;

}

int main()

{

   string str1="abbzzca";

   string str2="babzzcz";

   cout<<close(str1,str2);

    return 0;

}

