Q1.

#include <bits/stdc++.h>

using namespace std;

void checkpallindrome(string *s*){

    stack<char> st;

    for(char c:*s*){

        tolower(c);

        if(c>='a' && c<='z'){

        st.push(c);

        }

    }

    for(char c:*s*){

        if(char top = st.top()){

            st.pop();

            if (tolower(c) != top) {

                cout << "Not Palindrome\n";

                return;}

        }

    }

    cout<<"pallindrome\n";

}

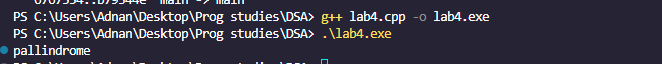
int main(){

checkpallindrome("nagan");

    return 0;

}

OUTPUT:



Q2.

string formstring(string *s*){

    stack <char> st;

     for(char c:*s*){

            if(c=='#'){

                st.pop();

            }

            else{

                st.push(c);

            }

        }

    string word = "";

    while(!st.empty()){

        word = word + st.top();

        st.pop();

    }

    return word;

}

bool check(string *s*,string *w*){

    string fs = formstring(*s*);

    string fw = formstring(*w*);

    if(fs==fw){return true;}

    return false;

}

Q3.

#include <iostream>

#include <vector>

using namespace std;

void backtrack(vector<int>& *nums*, vector<int>& *path*, vector<bool>& *used*, vector<vector<int>>& *result*) {

    if (*path*.size() == *nums*.size()) {

*result*.push\_back(*path*);

        return;

    }

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

        if (*used*[i]) continue;

*// choose*

*used*[i] = true;

*path*.push\_back(*nums*[i]);

*// explore*

        backtrack(*nums*, *path*, *used*, *result*);

*// un-choose (backtrack)*

*path*.pop\_back();

*used*[i] = false;

    }

}

vector<vector<int>> permute(vector<int>& *nums*) {

    vector<vector<int>> result;

    vector<int> path;

    vector<bool> used(*nums*.size(), false);

    backtrack(*nums*, path, used, result);

    return result;

}

int main() {

    vector<int> nums = {1,2,3};

    vector<vector<int>> ans = permute(nums);

    for (auto v : ans) {

        cout << "[ ";

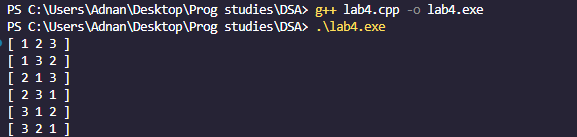
        for (int x : v) cout << x << " ";

        cout << "]\n";

    }

}

OUTPUT:



Q4.

bool dfs(vector<vector<char>>& *board*, string *word*, int *i*, int *j*, int *index*) {

    if (*index* == *word*.size()) return true; *// found full word*

    if (*i* < 0 || *j* < 0 || *i* >= *board*.size() || *j* >= *board*[0].size()) return false;

    if (*board*[*i*][*j*] != *word*[*index*]) return false;

    char temp = *board*[*i*][*j*];

*board*[*i*][*j*] = '#'; *// mark visited*

    bool found = dfs(*board*, *word*, *i*+1, *j*, *index*+1) ||

                 dfs(*board*, *word*, *i*-1, *j*, *index*+1) ||

                 dfs(*board*, *word*, *i*, *j*+1, *index*+1) ||

                 dfs(*board*, *word*, *i*, *j*-1, *index*+1);

*board*[*i*][*j*] = temp; *// backtrack*

    return found;

}

bool exist(vector<vector<char>>& *board*, string *word*) {

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

        for (int j = 0; j < *board*[0].size(); j++) {

            if (dfs(*board*, *word*, i, j, 0)) return true;

        }

    }

    return false;

}

int main(){

 vector<vector<char>> board = {

        {'A','B','C','E'},

        {'S','F','C','S'},

        {'A','D','E','E'}

    };

    string word = "ABCCED";

    cout << (exist(board, word) ? "true" : "false") << endl;