|  |
| --- |
| dsaa lab 02  Fatimaa Khan CT-23024  Rania Imran CT-23005 |
| Recursion, Backtracking, Stack |



## Question 01

## A palindrome is a word, phrase, number, or another sequence of characters that reads the same backward and forwards. Can you determine if a given string, s, is a palindrome? Write a Program using stack for checking whether a string is palindrome or not.

### **Source Code**

#include<iostream>

#include<string>

#include<stack>

using namespace std;

bool isPalindrome(const string& msg){

    stack<char> Stack;

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

        Stack.push(s[i]);

    }

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

        char top = Stack.top();

        Stack.pop();

        if(msg[i] != top){

            return false;

        }

        return true;

    }

}

int main(){

    string msg;

    cout<<"Enter string: ";

    getline(cin, msg);

    if(isPalindrome(msg)){

        cout<<"The given string is palindrome"<<endl;

    }

    else{

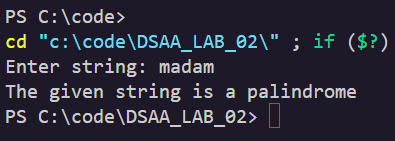
        cout<<"The given string is not palindrome"<<endl;

    }

    return 0;

}

### **Output**



**Question 2**

Given two strings s and t, return true if they are equal when both are typed into empty text editors. '#' means a backspace character. Note that after backspacing an empty text, the text will

continue empty.

Example 1: Input: s = "ab#c", t = "ad#c", Output: true, Explanation: Both s and t become "ac".

Example 2: Input: s = "a#c", t = "b", Output: false, Explanation: s becomes "c" while t becomes "b".

**Source Code**

#include <iostream>

#include <string>

#include<stack>

using namespace std;

string buildFinalString(const string & str) {

    string result;

    for (char ch : str) {

        if (ch == '#') {

            if (!result.empty()) {

                result.pop\_back();

            }

        } else {

            result.push\_back(ch);

        }

    }

    return result;

}

bool backspaceCompare(const string& s, const string& t) {

    return buildFinalString(s) == buildFinalString(t);

}

int main() {

    string s = "ab#c";

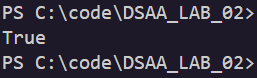
    string t = "ad#c";

    cout << (backspaceCompare(s, t) ? "True" : "False") << endl;

    return 0;

}

**Output**



**Question 3**

Given an array nums of distinct integers, return all the possible permutations. You can return the answer in any order.

Example 1: Input: nums = [1,2,3],

Output: [[1,2,3],[1,3,2],[2,1,3],[2,3,1],[3,1,2],[3,2,1]]

Example 2: Input: nums = [0,1],

Output: [[0,1],[1,0]]

**Source Code**

#include <iostream>

using namespace std;

void swap(int& a, int& b) {

    int temp = a;

    a = b;

    b = temp;

}

void printPermutation(int nums[], int size) {

    for (int i = 0; i < size; i++) {

        cout << nums[i] << " ";

    }

    cout << endl;

}

void permute(int nums[], int start, int size) {

    if (start == size) {

        printPermutation(nums, size);

        return;

    }

    for (int i = start; i < size; i++) {

        swap(nums[start], nums[i]); *// Manually swap the elements*

        permute(nums, start + 1, size); *// Recurse for the next element*

        swap(nums[start], nums[i]); *// Backtrack (undo the swap)*

    }

}

int main() {

    int nums[] = {1, 2, 3};

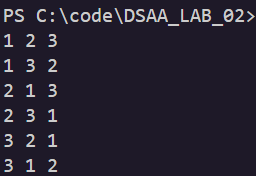
    int size = sizeof(nums) / sizeof(nums[0]);

    permute(nums, 0, size);

    return 0;

}

**Output**

****

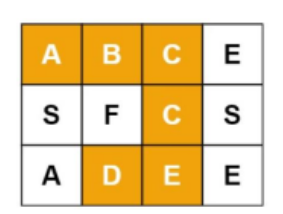
**Question 4**

Given an m x n grid of characters board and a string word, return true if word exists in the grid. The word can be constructed from letters of sequentially adjacent cells, where adjacent cells are

horizontally or vertically neighboring. The same letter cell may not be used more than once.

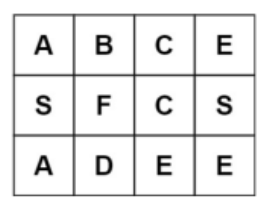
Example 01: Input: board = [["A","B","C","E"],["S","F","C","S"],["A","D","E","E"]],

word = "ABCCED", Output: true



Example 02: Input: board = [["A","B","C","E"],["S","F","C","S"],["A","D","E","E"]], word =

"ABCB", Output: false



**Source Code**

#include<iostream>

#include<vector>

#include<string>

using namespace std;

bool isSafe(vector<vector<char>>& board, string& word, int x, int y, int index){

    if(index == word.length()){

        return true;

    }

    if(x<0 || x>=board.size() || y<0 || y>=board[0].size() || board[x][y] != word[index]){return false;}

    char temp = board[x][y];

    board[x][y] = '\*';

    bool found = isSafe(board, word, x+1, y, index+1)||

                 isSafe(board, word, x-1, y, index+1)||

                 isSafe(board, word, x, y+1, index+1)||

                 isSafe(board, word, x, y-1, index+1);

    board[x][y] = temp;

    return found;

}

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

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

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

           if (isSafe(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 w1 = "ABCCED", w2 = "ABCB";

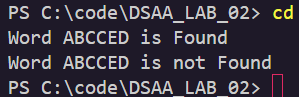
    cout<<"Word "<<w1<<" is "<<(ifExists(board, w1)? "Found" : "not Found")<<endl;

    cout<<"Word "<<w1<<" is "<<(ifExists(board, w2)? "Found" : "not Found")<<endl;

    return 0;

}

**Output**

****