/\* C++ program to solve N Queen Problem using backtracking to print all solutions \*/

#include <bits/stdc++.h>

using namespace std;

vector<vector<int> > result;

/\* A utility function to check if a queen can be placed on board[row][col]. Note that this function is called when "col" queens are already placed in columns from 0 to col -1. So we need to check only left side for attacking queens \*/

bool isSafe(vector<vector<int> > board,

            int row, int col)

{

    int i, j;

    int N = board.size();

    /\* Check this row on left side \*/

    for (i = 0; i < col; i++)

        if (board[row][i])

            return false;

    /\* Check upper diagonal on left side \*/

    for (i = row, j = col; i >= 0 && j >= 0; i--, j--)

        if (board[i][j])

            return false;

    /\* Check lower diagonal on left side \*/

    for (i = row, j = col; j >= 0 && i < N; i++, j--)

        if (board[i][j])

            return false;

    return true;

}

/\* A recursive utility function to solve N Queen problem \*/

bool solveNQUtil(vector<vector<int> >& board, int col)

{

    /\* base case: If all queens are placed then return true \*/

    int N = board.size();

    if (col == N) {

        vector<int> v;

        for (int i = 0; i < N; i++) {

            for (int j = 0; j < N; j++) {

                if (board[i][j] == 1)

                    v.push\_back(j + 1);

            }

        }

        result.push\_back(v);

        return true;

    }

    /\* Consider this column and try placing  this queen in all rows one by one \*/

 bool res = false;

    for (int i = 0; i < N; i++) {

        /\* Check if queen can be placed on board[i][col] \*/

        if (isSafe(board, i, col))

{

            /\* Place this queen in board[i][col] \*/

            board[i][col] = 1;

            // Make result true if any placement

            // is possible

            res = solveNQUtil(board, col + 1) || res;

            /\* If placing queen in board[i][col] doesn't lead to a solution, then

            remove queen from board[i][col] \*/

            board[i][col] = 0; // BACKTRACK

        }

    }

    /\* If queen can not be place in any row in

        this column col then return false \*/

    return res;

}

/\* This function solves the N Queen problem using Backtracking. It mainly uses solveNQUtil() to solve the problem. It returns false if queens cannot be placed, otherwise return true and prints placement of queens in the form of 1s. Please note that there may be more than one solutions, this function prints one of the feasible solutions.\*/

vector<vector<int> > nQueen(int n)

{

    result.clear();

    vector<vector<int> > board(n, vector<int>(n, 0));

    if (solveNQUtil(board, 0) == false) {

        return {};

    }

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

    return result;

}

int main()

{

    int n = 4;

    vector<vector<int> > v = nQueen(n);

    for (auto ar : v) {

        cout << "[";

        for (auto it : ar)

            cout << it << " ";

        cout << "]";

    }

    return 0;

}