#include<iostream>

using namespace std;

const int MAX\_N = 10; // Assuming a maximum size for simplicity

void printsolution(int board[][MAX\_N], int n) {

for(int i = 0; i < n ; i++) {

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

if(board[i][j]) {

cout << "Q ";

} else {

cout << "\_ ";

}

}

cout << "\n";

}

cout << "\n";

}

bool issafe(int board[][MAX\_N], int row, int col, int n) {

int i, j;

// check in row

for(i = 0 ; i < col ; i++) {

if(board[row][i]) {

return false;

}

}

// check for upper diagonal

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

if(board[i][j]) {

return false;

}

}

// Check lower diagonal

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

if(board[i][j]) {

return false;

}

}

return true;

}

bool nqprob(int board[][MAX\_N], int col, int n) {

if (col >= n) {

// All queens are placed successfully

printsolution(board, n);

return true;

}

bool solutionFound = false; // Flag to track if a solution is found

for (int i = 0; i < n; i++) {

if (issafe(board, i, col, n)) {

board[i][col] = 1;

// Display the board for this iteration

cout << "Current Board (Column " << col + 1 << "):\n";

printsolution(board, n);

// Recursively check for the next column

solutionFound = nqprob(board, col + 1, n);

if (solutionFound) {

// No need to continue if a solution is found

return true;

}

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

cout << "\nBacktracking at Column " << col + 1 << ", Row " << i + 1 << "\n";

printsolution(board, n);

}

}

return solutionFound; // Return whether a solution was found in this branch

}

int main() {

int n;

cout << "Enter the number of queens (N): ";

cin >> n;

if (n < 4 || n > MAX\_N) {

cout << "Enter a number between 4 and " << MAX\_N << ".";

return 1;

}

int board[MAX\_N][MAX\_N] = {0};

if (!nqprob(board, 0, n)) {

cout << "Solution does not exist";

}

return 0;

}