University of Engineering and Technology, Lahore (New Campus)

Department of Electrical Engineering and Technology

**Course:** Data Structures and Algorithms

Open Ended Lab Project: **N Queen Problem**

**Code:**

**#include <iostream>**

**#include <iomanip>**

**using namespace std;**

**// Function to initialize the chessboard with all cells as empty**

**void initializeBoard(char board[][8], int N) {**

**for (int i = 0; i < N; ++i) {**

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

**board[i][j] = '.';**

**}**

**}**

**}**

**// Function to display the chessboard**

**void displayBoard(char board[][8], int N) {**

**cout << "Current Board State:\n";**

**cout << " ";**

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

**cout << setw(2) << j << " ";**

**}**

**cout << endl;**

**for (int i = 0; i < N; ++i) {**

**cout << setw(2) << i << " ";**

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

**cout << board[i][j] << " ";**

**}**

**cout << endl;**

**}**

**cout << endl;**

**}**

**// Function to check if placing a queen at position (row, col) is safe**

**bool isSafe(char board[][8], int row, int col, int N) {**

**// Check row**

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

**if (board[row][i] == 'Q') {**

**return false;**

**}**

**}**

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

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

**if (board[i][j] == 'Q') {**

**return false;**

**}**

**}**

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

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

**if (board[i][j] == 'Q') {**

**return false;**

**}**

**}**

**return true;**

**}**

**// Function to solve N-Queen problem recursively**

**bool solveNQueensUtil(char board[][8], int col, int N, bool &backtracked) {**

**// Base case: If all queens are placed, return true**

**if (col == N) {**

**return true;**

**}**

**// Try placing a queen in each row of the current column**

**for (int i = 0; i < N; ++i) {**

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

**if (isSafe(board, i, col, N)) {**

**// Place the queen**

**board[i][col] = 'Q';**

**// Display the board after placing the queen**

**displayBoard(board, N);**

**// Recur to place rest of the queens**

**if (solveNQueensUtil(board, col + 1, N, backtracked)) {**

**return true;**

**}**

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

**// then backtrack and remove the queen from board[i][col]**

**board[i][col] = '.';**

**// Set the backtracking flag to true to indicate a backtrack**

**backtracked = true;**

**}**

**}**

**// If no queen can be placed in any row of the current column, return false**

**if (backtracked) {**

**cout << "Backtracking...\n" << endl;**

**backtracked = false; // Reset the flag**

**}**

**return false;**

**}**

**// Function to solve the N-Queen problem and display the solution**

**void solveNQueens() {**

**int N;**

**cout << "Welcome to the N-Queens Solver!\n";**

**cout << "Please enter the number of queens (greater than 4 for meaningful solutions): ";**

**cin >> N;**

**if (N <= 4) {**

**cout << "The number of queens should be greater than or equals to 4. Please restart the program and try again." << endl;**

**return;**

**}**

**char board[8][8];**

**initializeBoard(board, N);**

**// Display initial empty board**

**cout << "\nInitial empty board:" << endl;**

**displayBoard(board, N);**

**// Flag to track backtracking**

**bool backtracked = false;**

**// Solve N-Queen problem starting from the first column**

**if (solveNQueensUtil(board, 0, N, backtracked)) {**

**cout << "Solution exists:" << endl;**

**displayBoard(board, N);**

**} else {**

**cout << "Solution does not exist for " << N << "-Queens problem." << endl;**

**}**

**}**

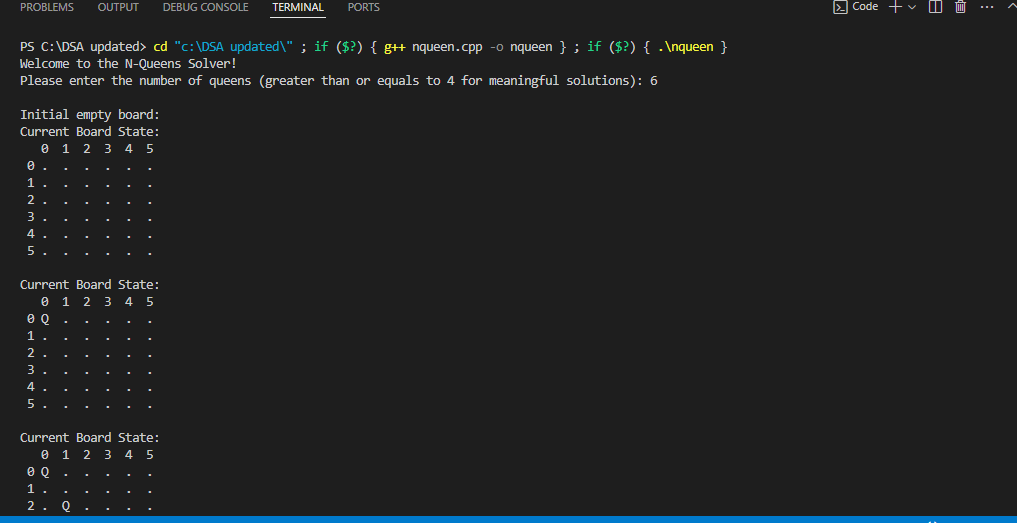
**int main() {**

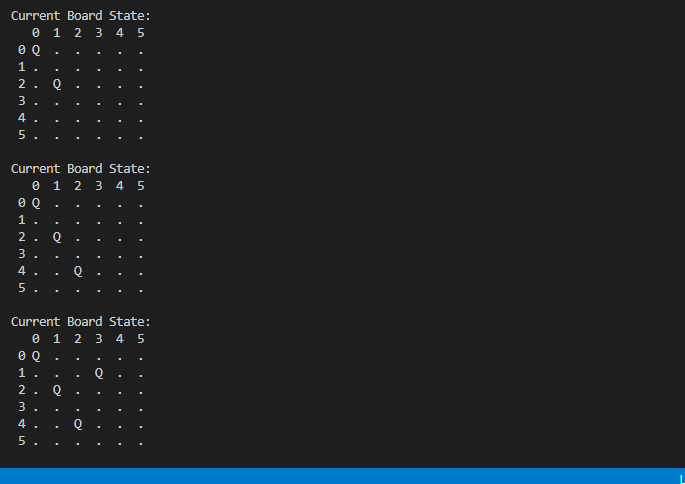
**solveNQueens();**

**return 0;**

**}**

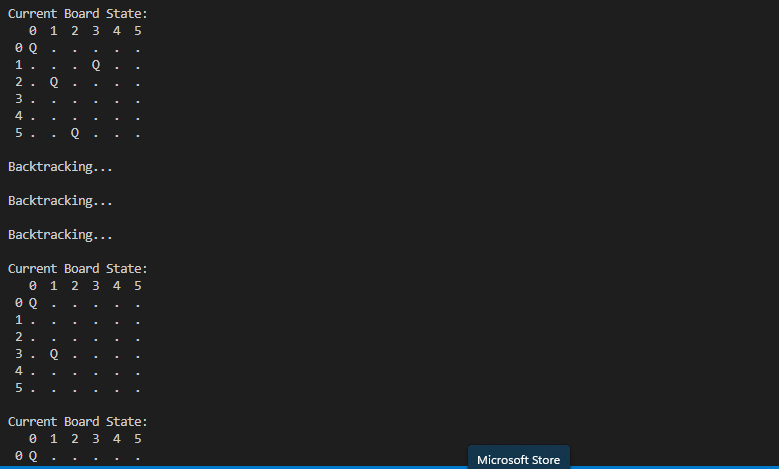
**Output:**

****

****

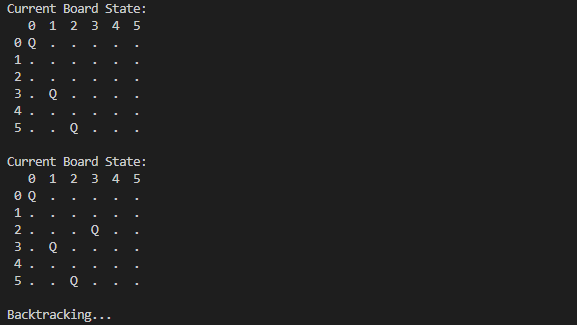
**A screenshot of a computer

Description automatically generated**

****

**A screenshot of a computer

Description automatically generated**

****

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**A screen shot of a computer

Description automatically generated**

**A black rectangle with white text

Description automatically generated**

**A screen shot of a computer

Description automatically generated**

**A black rectangle with white text

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**Final Output**

**A black rectangle with white text

Description automatically generated**

**Methodology:**1. User Interaction and Input Validation: The program begins with a welcome message and prompts the user to enter the number of queens (`N`). It checks if `N` is greater than 4, as smaller values do not produce meaningful solutions for the N-Queens problem.2. Chessboard Initialization: A function initializeBoard() initializes a chessboard of size `N x N` with all cells set to ‘ . ’ indicating they are empty.3. Chessboard Display: A function displayBoard() displays the current state of the chessboard. It includes row and column headers for better readability.4. Safety Check for Queen Placement: The function isSafe() checks if placing a queen at a given position (row, col) is safe. It ensures no other queens can attack this position by checking: - The row to the left of the column. - The upper diagonal on the left side. - The lower diagonal on the left side.5. Recursive Backtracking Algorithm: - The core recursive function solveNQueensUtil() tries to place queens column by column. - For each column, it attempts to place a queen in each row (from 0 to `N-1`). - After placing a queen, it recursively attempts to place queens in the next column. - If placing a queen in a column leads to a solution, the function returns `true`. - If placing a queen in any row of the current column does not lead to a solution, it backtracks by removing the queen and sets a flag `backtracked` to indicate that backtracking is occurring. - The "Backtracking..." message is displayed whenever the algorithm backtracks.6. Final Solution: - If a solution exists, the final board configuration is displayed. - If no solution exists, an appropriate message is displayed.

**Detailed Code Execution:**1. main() Function: Calls the solveNQueen() function.2. solveNQueens() Function: - Prompts the user for the number of queens and validates the input. - Initializes the chessboard. - Displays the initial empty board. - Calls the recursive solveNQueensUtil() function to attempt to solve the problem. - Displays the solution if one is found or indicates that no solution exists.3. initializeBoard() Function: - Sets all positions on the board to '.'.4. displayBoard() Function: - Prints the current state of the board, with each row and column value nicely formatted.5. isSafe() Function: - Checks the row, upper diagonal, and lower diagonal for conflicts.6. solveNQueensUtil() Function: - Recursively attempts to place queens in each column. - Calls isSafe() to check if a position is valid for placing a queen. - Places the queen and calls itself to try to place queens in subsequent columns. - If placing a queen leads to a solution, it returns `true`. - If no valid position is found, it removes the last placed queen and sets the `backtracked` flag. - Displays the "Backtracking..." message when backtracking.This methodology ensures that the N-Queens problem is solved using a recursive backtracking approach, with user-friendly prompts and clear output at each step.