S. B. JAIN INSTITUTE OF TECHNOLOGY,

MANAGEMENT & RESEARCH, NAGPUR.

Practical No. 8

Aim: Create a program that solves the N-Queens problem using backtracking.

Name of Student: Shrutika Pradeep Bagdi

Roll No: CS22130

Semester/Year: V/III

Academic Session:2024-2025

Date of Performance:

Date of Submission:

AIM: Create a program that solves the N-Queens problem using backtracking.

OBJECTIVE/EXPECTED LEARNING OUTCOME:

The objectives and expected learning outcome of this practical are:

- To understand and implement the N Queens problem using backtracking.

THEORY:

The N-Queens problem is a classic algorithmic challenge in which the objective is to place N queens on an N×N chessboard such that no two queens threaten each other. This means that no two queens can be in the same row, column, or diagonal.

Problem Definition:

- **Input**: An integer N, representing the size of the chessboard and the number of queens to place.
- **Output**: All possible arrangements of the N queens on the board.

Constraints:

1. **Rows**: Each queen must occupy a different row.
2. **Columns**: Each queen must occupy a different column.
3. **Diagonals**: No two queens can be on the same diagonal, which can be represented by the difference and sum of their row and column indices.

Backtracking Approach:

A common method to solve the N-Queens problem is through backtracking. Here's how it works:

1. **Place a queen in a row**: Start from the first row and try to place a queen in each column.
2. **Check for validity**: After placing a queen, check if it is safe (not attacked by any previously placed queens).
3. **Recursion**: If safe, recursively attempt to place queens in the next row.
4. **Backtrack**: If placing a queen in any column of a row doesn't lead to a solution, remove the queen (backtrack) and try the next column.
5. **Base case**: If queens are placed in all rows, record the solution.

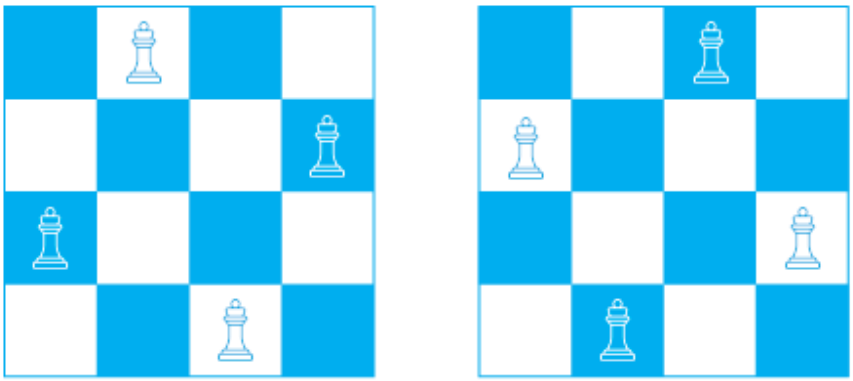
Complexity:

- **Time Complexity**: The time complexity is approximately O(N!) in the worst case, due to the permutations of queen placements.
- **Space Complexity**: O(N) for storing the positions of the queens.

Solutions:

For small values of N (like 1 through 15), the problem can be solved relatively quickly. The number of solutions varies with N:

- N=1: 1 solution
- N=2: 0 solutions
- N=3: 0 solutions
- N=4: 2 solutions
- N=5: 10 solutions
- N=8: 92 solutions



Applications:

The N-Queens problem has applications in various fields including:

- **Artificial Intelligence**: Used to teach backtracking and constraint satisfaction problems.
- **Computer Science**: Helps in understanding algorithms, recursion, and optimization.
- **Game Theory**: Insights into strategy and game development.

Overall, the N-Queens problem serves as an excellent introduction to algorithm design and problem-solving techniques.

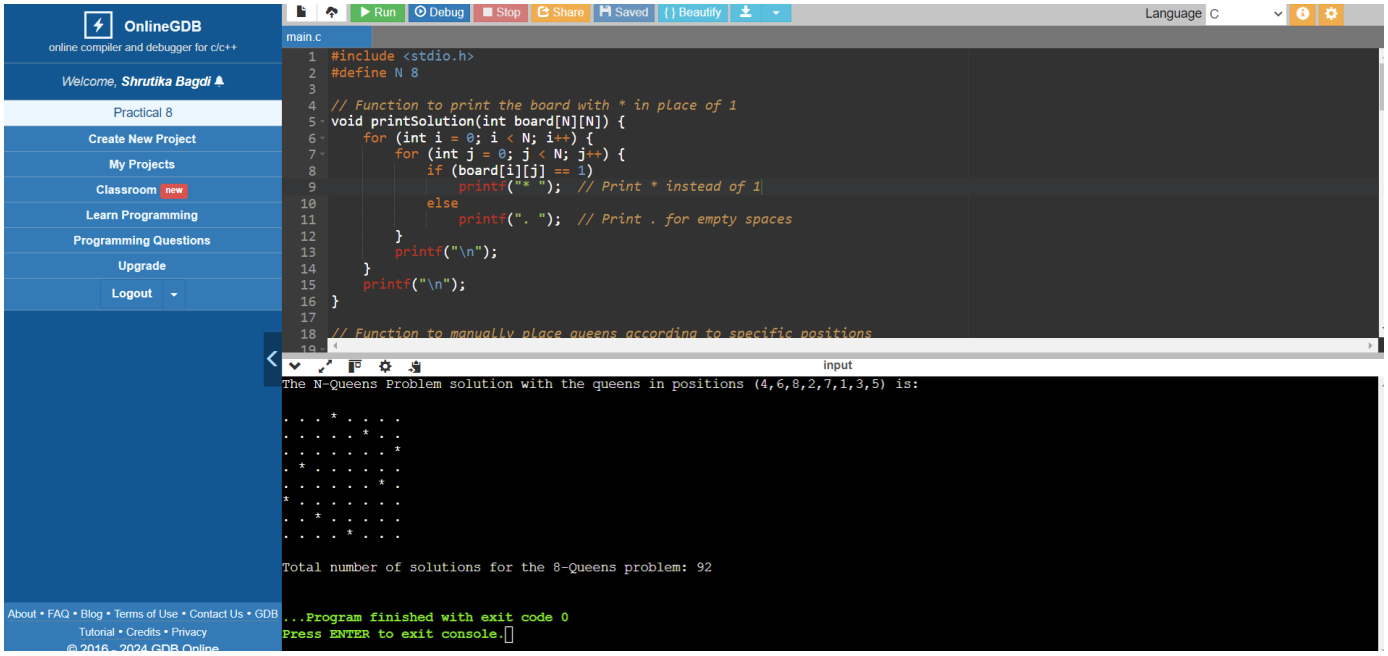
CODE:

```
#include <stdio.h>
#define N 8
void printSolution(int board[N][N]) {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            if (board[i][j] == 1)
                printf("* ");
            else
                printf(". ");
        }
        printf("\n");
    }
    printf("\n");
}
void placeQueens(int board[N][N], int positions[N]) {
    for (int i = 0; i < N; i++) {
        board[i][positions[i] - 1] = 1;  // Place the queen at the specified column in each row
    }
}
int Safe(int board[N][N], int row, int col) {
    int i, j;
    for (i = 0; i < col; i++)
        if (board[row][i])
            return 0;
    for (i = row, j = col; i >= 0 && j >= 0; i--, j--)
        if (board[i][j])
            return 0;
    for (i = row, j = col; j >= 0 && i < N; i++, j--)
        if (board[i][j])
            return 0;
```

```
    return 1;
}
int solveNQUtil(int board[N][N], int col) {
    // Base case: If all queens are placed, a solution is found
    if (col >= N) {
        return 1;
    }
    int solutions = 0;  // To count the number of valid solutions
    for (int i = 0; i < N; i++) {
        if (Safe(board, i, col)) {
            board[i][col] = 1;
            solutions += solveNQUtil(board, col + 1);
            board[i][col] = 0;
        }
    }
    return solutions;
}
int solveNQ() {
    int board[N][N] = {0};
    return solveNQUtil(board, 0);
}
int main() {
    int board[N][N] = {0};
    int positions[N] = {4, 6, 8, 2, 7, 1, 3, 5};
    placeQueens(board, positions);
    printf("The N-Queens Problem solution with the queens in positions (4,6,8,2,7,1,3,5) is:\n\n");
    printSolution(board);
    int totalSolutions = solveNQ();
    printf("Total number of solutions for the 8-Queens problem: %d\n", totalSolutions);
```

```
    return 0;
}
```

INPUT & OUTPUT WITH DIFFERENT TEST CASES:



CONCLUSION:

DISCUSSION AND VIVA VOCE:

- Explain the N Queens problem.
- Discuss the complexity of N Queens problem.
- Discuss the applications of backtracking approach.

REFERENCES:

- https://www.geeksforgeeks.org/n-queen-problem-backtracking-3/
- https://www.geeksforgeeks.org/printing-solutions-n-queen-problem/
- https://www.javatpoint.com/n-queens-problems
- https://www.prepbytes.com/blog/backtracking/how-do-you-solve-the-n-queen-problem/