N – QUEEN Solution Using the Las-Vegas Algorithm

```
1.  Algorithm: n_queen(empty_slots_vector , row)
2.  // Empty_slots_vector contains the safe indexes where the queen can be placed
3.  //Row defines the current row where we need to place the queen
4.          If(row := n-1 and sizeof(empty_slots_vector) = 1) then
5.          {       Col := empty_slots_vector[0];
6.                  Write( Board[row][col] = 1; )
7.                  return true;
8.         } else if(row <= n-1 and  sizeof(empty_slots_vector) = 0 )
9.                   {
10.                    // No Empty Slots left
11.                    return false;
12.                  }
13.       else{
14.          // Randomly select a col from empty_slot_vector
15.          Col:= rand() % sizeof(empty_slot_vector);
16.          //Assume placing the queen is safe and place it
17.          Write ( Board[row][col] = 1; )
18.          //Mark all unsafe positions on board due to the present queen
19.          // Compute new_empty_slots_vector in next row of board
20.          empty_slot_vector :=  empty_slot_vector ( row + 1 );
21.          return:  n_queen(empty_slots_vector , row + 1);
22.          }
23. }
```

Implementation of N-Queen Problem in C++

Code:

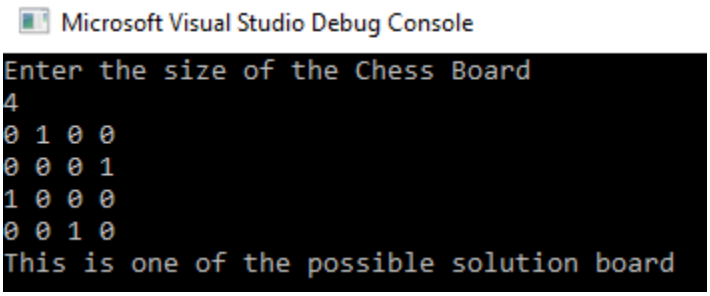
```
#include<iostream>
#include<vector>
#include<cstdlib>
using namespace std;
/// <summary>
/// 8-Queen Randomised Algorithm ( Las Vegas Algorithm )
	/// board[i][j] == 1 -----> Queen is Placed There
	/// board[i][j] == 0 -----> All other queens in the board doesnt attack the position
	/// board[i][j] == -1 -----> The position is attacked by alteast 1 queen in the board
	/// </summary>
	/// <returns></returns>
//Globally Set the board
vector<vector<int>> board{};

void spot_attack_vertical(int row, int col) {
	for (int i = 0; i < 8; i++)
		board[i][col] = -1;
}
void spot_attack_left_diagonal(int row, int col) {
	for (int i = row + 1, j = col - 1; i < 8 && col >= 0; i++, j--) {
		board[i][j] = -1;
	}
}
void spot_attack_right_diagonal(int row, int col) {
	for (int i = row + 1, j = col + 1; i < 8 && col < 8; i++, j++) {
		board[i][j] = -1;
	}
}
vector<int> compute_empty_slots(int row) {
	vector<int> empty{};
	for (int i = 0; i < 8; i++) {
		if (board[row][i] == 0)
			empty.push_back(i);
	}
	return empty;
}
//Send the Empty_slot_indexes vector and the row to place the queen as parameters
bool compute_board(vector<int> empty_slots, int row) {
	//Its last row on the board while computing and only 1 empty_slot is left
	if (row == 7 && empty_slots.size() == 1) {
		board[row][empty_slots.at(0)] = 1;
		return true;
	}
	//Its computation of any row and No of empty slots in that row is 0 --> no safe_slot to
place the queen
	else if (row <= 7 && empty_slots.size() == 0) {
		return false;
	}
	else {
		//Randomly Select any element_index in the empty slot vector
		int rand_index = rand() % (empty_slots.size());
		//select the column at the randmo index
		int col = empty_slots.at(rand_index);
		//Assume queen is safe there and place the queen
		board[row][col] = 1;
		//Marks all unsafe position in all directions and mark them as -1
```

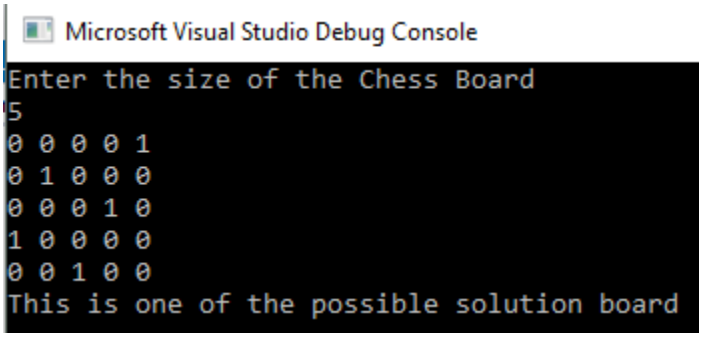
```
                spot_attack_vertical(row, col);
                spot_attack_left_diagonal(row, col);
                spot_attack_right_diagonal(row, col);
                //Compute the New Empty Slots Indexes for the next row
                vector<int> new_empty_slots = compute_empty_slots(row + 1);
                return compute_board(new_empty_slots, row + 1);
        }
}
void initialise_board() {
        for (int i = 0; i < 8; i++) {
                for (int j = 0; j < 8; j++) {
                        board[i][j] = 0;
                }
        }
}
void display_board() {
        for (int i = 0; i < 8; i++) {
                for (int j = 0; j < 8; j++) {
                        cout << board[i][j] << " ";
                }
                cout << endl;
        }
}
int main() {
        // Indexes of the empty_slots in the board
        vector<int> empty_slots{ 0,1,2,3,4,5,6,7 };
        //Set initial result to false
        bool result = false;
        //Start from row = 0 to row = 7
        int row = 0;
        while (!result){
                //set all positions in board to zeros
                initialise_board();
                result = compute_board(empty_slots, row);
        }
        display_board();
}
```

Output :

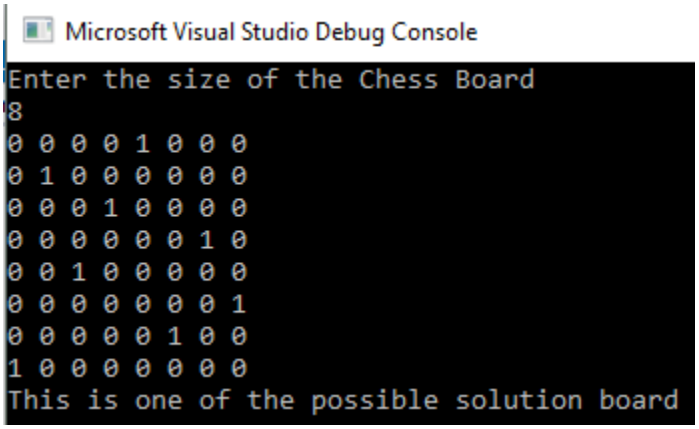
Solution for Board Size 4



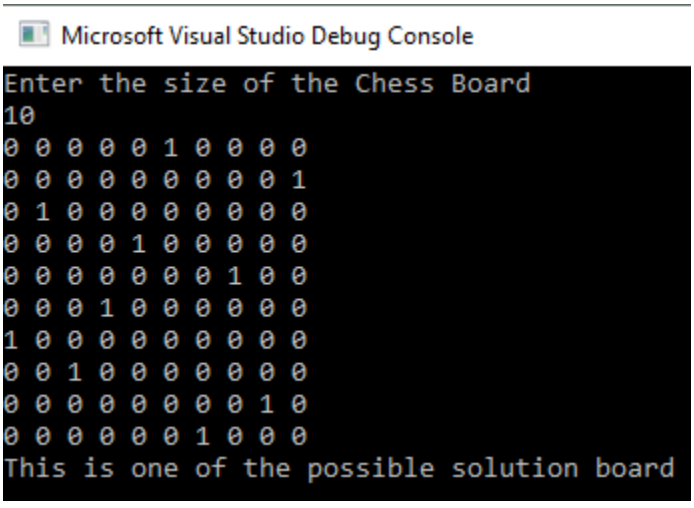
Solution for Board Size 5



Solution for Board Size 8



Solution for Board Size 10



Solution for Board Size 16

```
Enter the size of the Chess Board
16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
This is one of the possible solution board
```

Solution for Board Size 25

```
Enter the size of the Chess Board
25
0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
This is one of the possible solution board
```

========== THANK YOU ==========