**Assignment 6**

**The N-Queens Problem**

The N-Queens Problem is a classic puzzle and a well-known problem in computer science and mathematics. The challenge is to place N chess queens on an chessboard in such a way that no two queens threaten each other. According to the rules of chess, this means that no two queens can be on the same row, column, or diagonal.

The problem is a great example of a constraint satisfaction problem that can be solved efficiently using an algorithmic paradigm known as backtracking.

**Backtracking**

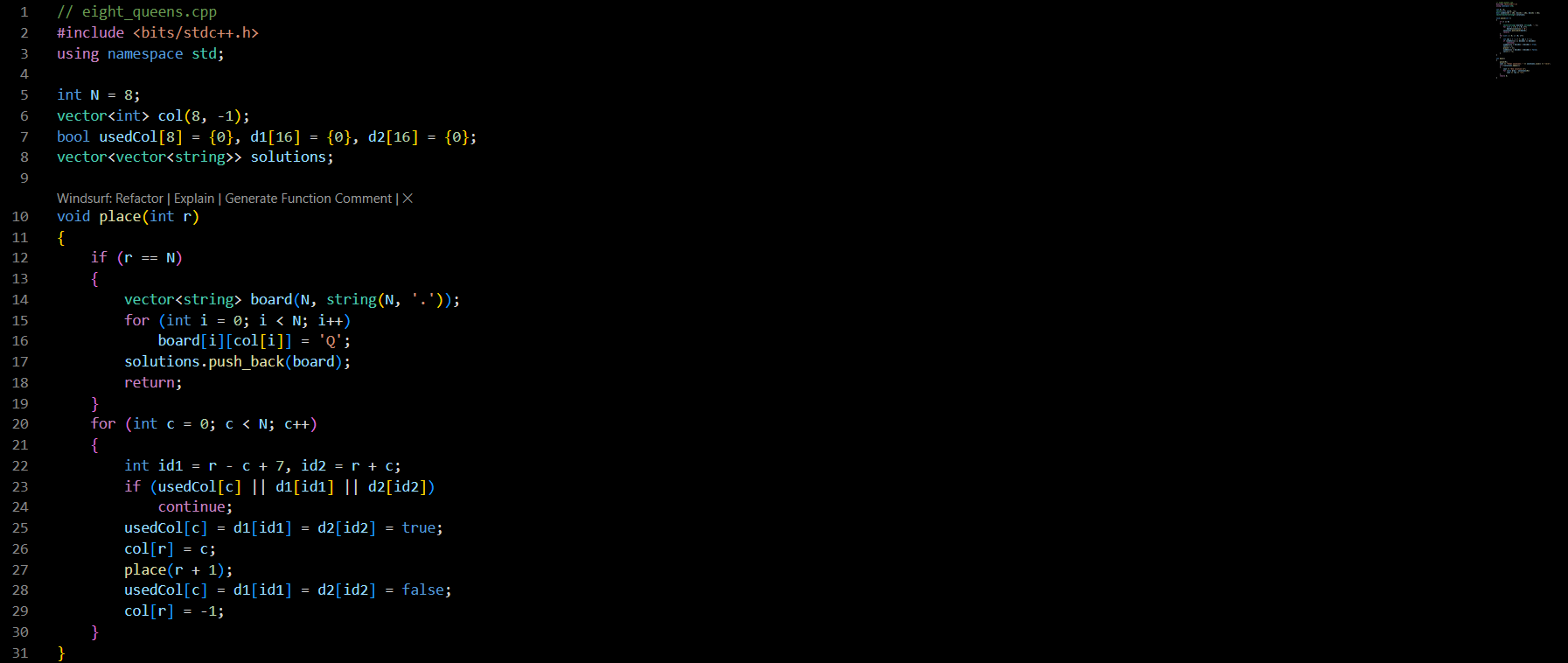
Backtracking is a recursive, depth-first search algorithm for finding solutions to computational problems. It works by incrementally building candidates for a solution and abandoning a candidate ("backtracking") as soon as it determines that the candidate cannot possibly be completed to a valid solution.

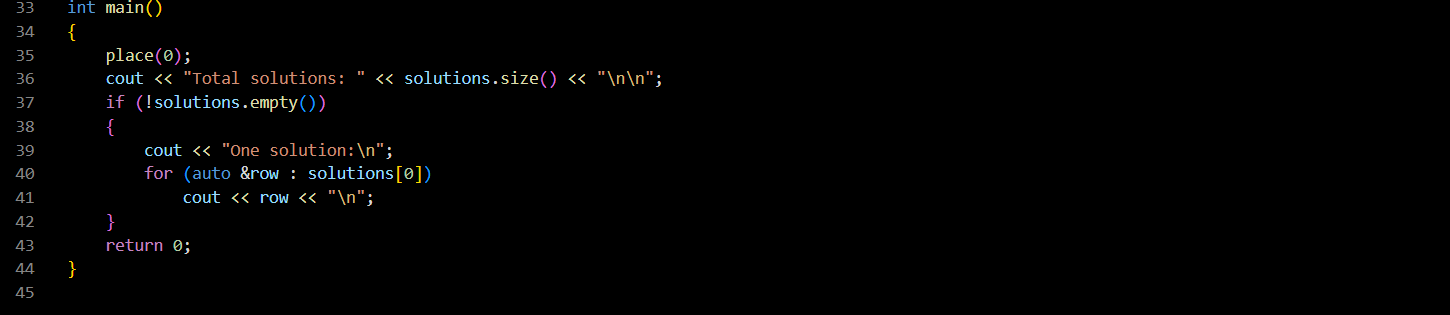
The general approach for the N-Queens problem is as follows:

1. Start: Begin by placing a queen in the first column of the first row (row 0).
2. Recurse: Move to the next row and try to place a queen in a "safe" column. A column is safe if it's not under attack from any previously placed queens.
3. Check for Safety: To place a queen at position , we must check three conditions:
   * The column c is not already occupied.
   * The main diagonal is not occupied. All cells on this diagonal share the same value for the expression .
   * The anti-diagonal is not occupied. All cells on this diagonal share the same value for the expression .
4. Solution Found: If we successfully place a queen in the final row (row N-1), we have found a valid solution.
5. Backtrack: If we get to a row where no column is safe, it means our previous placement was incorrect. We must backtrack to the previous row, remove the queen we placed there, and try placing it in the next available safe column in that row.

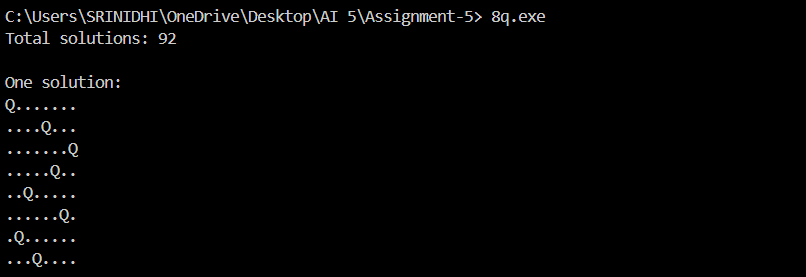
This process continues until all possible configurations have been explored, finding every possible solution.

**Code –**

****

****

**Output –**

****

**Global Variables -**

The program begins by setting up several key variables to keep track of the board state. A variable N is set to 8, defining the size of the chessboard.

A vector named col is used to store the queen's position for each row; for example, if a queen is at row 2, column 5, this vector would store the value 5 at its second index.

To quickly check if a new queen placement is valid, three boolean arrays are used:

* usedCol: Tracks if a column is already occupied by another queen.
* d1: Tracks the main diagonals (top-left to bottom-right). All squares on the same main diagonal share a constant value for the formula row - column. An offset is added to this value to ensure the array index is always positive.
* d2: Tracks the anti-diagonals (top-right to bottom-left). All squares on an anti-diagonal share a constant value for row + column.

Finally, a vector of vectors of strings, named solutions, is created to store all the valid board configurations that are found.

**The place Function -**

This is the core recursive function that implements the backtracking algorithm. It's designed to place a queen in a specific row, which is passed as an argument.

* Base Case: The function first checks if it has successfully placed a queen in every row (i.e., if the current row number is equal to N). If it has, a complete and valid solution has been found. The program then constructs a string representation of the board based on the positions stored in the col vector and adds this board to the solutions list.
* Recursive Step: If it's not the base case, the function loops through every column in the current row. For each column, it calculates the corresponding diagonal indices and checks the usedCol, d1, and d2 arrays to see if placing a queen there would be safe.
* Place and Recurse: If the spot is safe, the function "places" the queen. It does this by updating the tracking arrays to mark the current column and diagonals as occupied. It then records the queen's column position in the col vector and makes a recursive call to itself to solve for the *next* row.
* Backtrack: This is the critical step. After the recursive call for the next row returns, the function undoes the move. It resets the tracking arrays and the col vector for the current position. This "backtracking" frees up the spot, allowing the loop to continue and try placing the queen in the next column of the current row to explore other possibilities.

**The main Function -**

The main function is the entry point of the program. It kicks off the entire process by making the initial call to the place function, starting the search from row 0. Once that initial call completes, the backtracking algorithm has explored all possible configurations. The program then prints the total number of solutions found and displays the first solution from the solutions list as an example.

**Conclusion -** This program effectively demonstrates the power and elegance of the **backtracking** algorithm. By systematically exploring possibilities and pruning branches of the search space that violate the problem's constraints, it can find all solutions to a complex combinatorial problem like N-Queens. The use of boolean arrays for checking columns and diagonals is a key optimization that makes the safety check very fast (), contributing to the overall efficiency of the solution. The N-Queens problem remains a fundamental exercise for understanding recursion and backtracking in computer science.