```c
#include <stdio.h>
#include <stdlib.h>

int ld[30] = { 0 };
int rd[30] = { 0 };
int cl[30] = { 0 };

// A utility function to print solution
void printSolution(int** board, int n) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++)
            printf(" %s ", board[i][j] == 1 ? "Q" : ".");
        printf("\n");
    }
}

// A recursive utility function to solve N-Queen problem
int solveNQUtil(int** board, int col, int n) {
    // Base case: If all queens are placed then return true
    if (col >= n)
        return 1;

    // Consider this column and try placing this queen in all rows one by one
    for (int i = 0; i < n; i++) {
        // Check if the queen can be placed on board[i][col]
        if ((ld[i - col + n - 1] != 1 && rd[i + col] != 1) && cl[i] != 1) {
            // Place this queen in board[i][col]
            board[i][col] = 1;
            ld[i - col + n - 1] = rd[i + col] = cl[i] = 1;

            // Recur to place the rest of the queens
            if (solveNQUtil(board, col + 1, n))
                return 1;

            // BACKTRACK
            board[i][col] = 0;
            ld[i - col + n - 1] = rd[i + col] = cl[i] = 0;
        }
    }

    // If the queen cannot be placed in any row in this column, return 0
    return 0;
```

```c

// Main function to solve N-Queen problem
int solveNQ(int n) {
    // Allocate memory for the board
    int** board = (int**)malloc(n * sizeof(int*));
    for (int i = 0; i < n; i++)
        board[i] = (int*)malloc(n * sizeof(int));

    // Initialize the board with 0s
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            board[i][j] = 0;

    // Call the recursive function to solve the problem
    if (solveNQUtil(board, 0, n) == 0) {
        printf("Solution does not exist\n");
        return 0;
    }

    // Print the solution
    printSolution(board, n);
    return 1;
}

// Driver program to test the function
int main() {
    int n;
    printf("Enter the size of the board (n): ");
    scanf("%d", &n);

    // Ensure that the board size is valid
    if (n <= 0) {
        printf("Board size should be a positive integer.\n");
        return 1;
    }

    // Solve the N-Queen problem for the given size
    solveNQ(n);

    return 0;
}

```

Output

```
Enter the size of the board (n): 4
 .  .  Q  .
 Q  .  .  .
 .  .  .  Q
 .  Q  .  .


=== Code Execution Successful ===
```

```c
#include <stdio.h>
#include <string.h>

void search(char* pat, char* txt) {
    int M = strlen(pat);
    int N = strlen(txt);

    // A loop to slide pat[] one by one
    for (int i = 0; i <= N - M; i++) {
        int j;
        // For current index i, check for pattern match
        for (j = 0; j < M; j++) {
            if (txt[i + j] != pat[j]) {
                break;
            }
        }
        // If pattern matches at index i
        if (j == M) {
            printf("Pattern found at index %d\n", i);
        }
    }
}

int main() {
    // Declare character arrays for the text and the pattern
    char txt[100], pat[100];

    // Prompt the user for input
    printf("Enter the text: ");
    scanf("%s", txt);  // Reading the text string

    printf("Enter the pattern: ");
    scanf("%s", pat);  // Reading the pattern string

    // Call the search function to find pattern in the text
    printf("\nSearching for pattern in the text...\n");
    search(pat, txt);

    return 0;
}
```

Output

```
Enter the text: jaaayshriram
Enter the pattern: ram

Searching for pattern in the text...
Pattern found at index 9


=== Code Execution Successful ===
```