NAME:- Sourav Paul
GROUP:- D2
REG. NO:- 20214056
BRANCH:- CSE DEPT.

ASSIGNMENT - 8

Q1 Write a C program to implement N Queen Problem using back tracking.

Ans:-
```
#include <bits/stdc++.h>
#define N 4
using namespace std;


void printSolution(int board[N][N])
{
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++)
           if(board[i][j])
            cout << "Q ";
           else cout<<". ";
        printf("\n");
    }
}


bool isSafe(int board[N][N], int row, int col)
{
    int i, j;
    /* Check this row on left side */
    for (i = 0; i < col; i++)
        if (board[row][i])
            return false;

    /* Check upper diagonal on left side */
    for (i = row, j = col; i >= 0 && j >= 0; i--, j--)
        if (board[i][j])
            return false;

    /* Check lower diagonal on left side */
    for (i = row, j = col; j >= 0 && i < N; i++, j--)
        if (board[i][j])
            return false;

    return true;
}


bool solveNQUtil(int board[N][N], int col)
{

    if (col >= N)
        return true;

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

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

            board[i][col] = 1;


            if (solveNQUtil(board, col + 1))
                return true;


            board[i][col] = 0; // BACKTRACK
        }
    }


    return false;
}


bool solveNQ()
{
    int board[N][N] = { { 0, 0, 0, 0 },
                        { 0, 0, 0, 0 },
                        { 0, 0, 0, 0 },
                        { 0, 0, 0, 0 } };

    if (solveNQUtil(board, 0) == false) {
        cout << "Solution does not exist";
        return false;
    }

    printSolution(board);
    return true;
}


int main()
{
    solveNQ();
    return 0;
}
```

Q2) Write a C program to implement Rat in a Maze.

Ans:-

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

```
#define N 4


bool solveMazeUtil(int maze[N][N], int x, int y, int sol[N][N]);


void printSolution(int sol[N][N])
{
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++)
            printf(" %d ", sol[i][j]);
        printf("\n");
    }
}


bool isSafe(int maze[N][N], int x, int y)
{
    // if (x, y outside maze) return false
    if (x >= 0 && x < N && y >= 0 && y < N && maze[x][y] == 1)
        return true;

    return false;
}
```

```
bool solveMaze(int maze[N][N])
{
    int sol[N][N] = { { 0, 0, 0, 0 },
                    { 0, 0, 0, 0 },
                    { 0, 0, 0, 0 },
                    { 0, 0, 0, 0 } };

    if (solveMazeUtil(maze, 0, 0, sol) == false) {
        printf("Solution doesn't exist");
        return false;
    }

    printSolution(sol);
    return true;
}


bool solveMazeUtil(int maze[N][N], int x, int y, int sol[N][N])
{
    // if (x, y is goal) return true
    if (x == N - 1 && y == N - 1) {
        sol[x][y] = 1;
        return true;
    }

    // Check if maze[x][y] is valid
    if (isSafe(maze, x, y) == true) {
        // mark x, y as part of solution path
        sol[x][y] = 1;

        /* Move forward in x direction */
        if (solveMazeUtil(maze, x + 1, y, sol) == true)
            return true;

        /* If moving in x direction doesn't give solution then
        Move down in y direction */
        if (solveMazeUtil(maze, x, y + 1, sol) == true)
            return true;

        /* If none of the above movements work then BACKTRACK:
            unmark x, y as part of solution path */
        sol[x][y] = 0;
        return false;
    }

    return false;
```

```
}



int main()

{

    int maze[N][N] = { { 1, 0, 0, 0 },

                       { 1, 1, 0, 1 },

                       { 0, 1, 0, 0 },

                       { 1, 1, 1, 1 } };


    solveMaze(maze);

    return 0;

}
```

Q3) Write a C program to implement Sudoku Problem.

Ans:-

```
#include <bits/stdc++.h>
using namespace std;

#define UNASSIGNED 0
#define N 9


bool FindUnassignedLocation(int grid[N][N],
                            int& row, int& col);
```

```
bool isSafe(int grid[N][N], int row,
            int col, int num);

/
bool SolveSudoku(int grid[N][N])
{
    int row, col;

    // If there is no unassigned location, we are done
    if (!FindUnassignedLocation(grid, row, col))
            return true;

    // Consider digits 1 to 9
    for (int num = 1; num <= 9; num++)
    {
        if (isSafe(grid, row, col, num))
        {
            grid[row][col] = num;
            if (SolveSudoku(grid))
                return true;

            grid[row][col] = UNASSIGNED;
        }
    }

    return false;
}

bool FindUnassignedLocation(int grid[N][N],
                            int& row, int& col)
{
    for (row = 0; row < N; row++)
        for (col = 0; col < N; col++)
            if (grid[row][col] == UNASSIGNED)
                return true;
    return false;
}


bool UsedInRow(int grid[N][N], int row, int num)
```

```
{
   for (int col = 0; col < N; col++)
      if (grid[row][col] == num)
         return true;
   return false;
}

bool UsedInCol(int grid[N][N], int col, int num)
{
   for (int row = 0; row < N; row++)
      if (grid[row][col] == num)
         return true;
   return false;
}


bool UsedInBox(int grid[N][N], int boxStartRow,
         int boxStartCol, int num)
{
   for (int row = 0; row < 3; row++)
      for (int col = 0; col < 3; col++)
         if (grid[row + boxStartRow]
               [col + boxStartCol] ==
                           num)
            return true;
   return false;
}

bool isSafe(int grid[N][N], int row,
         int col, int num)
{

   return !UsedInRow(grid, row, num)
      && !UsedInCol(grid, col, num)
      && !UsedInBox(grid, row - row % 3,
               col - col % 3, num)
      && grid[row][col] == UNASSIGNED;
}

void printGrid(int grid[N][N])
```

```
{
   for (int row = 0; row < N; row++)
   {
      for (int col = 0; col < N; col++)
         cout << grid[row][col] << " ";
      cout << endl;
   }
}


int main()
{
   // 0 means unassigned cells
   int grid[N][N] = { { 3, 0, 6, 5, 0, 8, 4, 0, 0 },
               { 5, 2, 0, 0, 0, 0, 0, 0, 0 },
               { 0, 8, 7, 0, 0, 0, 0, 3, 1 },
               { 0, 0, 3, 0, 1, 0, 0, 8, 0 },
               { 9, 0, 0, 8, 6, 3, 0, 0, 5 },
               { 0, 5, 0, 0, 9, 0, 6, 0, 0 },
               { 1, 3, 0, 0, 0, 0, 2, 5, 0 },
               { 0, 0, 0, 0, 0, 0, 0, 7, 4 },
               { 0, 0, 5, 2, 0, 6, 3, 0, 0 } };
   if (SolveSudoku(grid) == true)
      printGrid(grid);
   else
      cout << "No solution exists";

   return 0;
}
```