PROBLEM SOLVING USING C
FINAL PROJECT

Submitted by:

Amal Jyothi S (2447206)

Team Members:

Kuruvilla George (2447232)

Rohan Cherian Jacob (2447243)

Submitted on:

17th SEPTEMBER 2024

Submitted to:

Dr. Nisha Varghese

CHRIST UNIVERSITY

BENGALURU, INDIA

Declared as Deemed to be University under Section 3 of UGC Act 1956

“SUDOKU SOLVER USING C PROGRAMMING”

CODE:

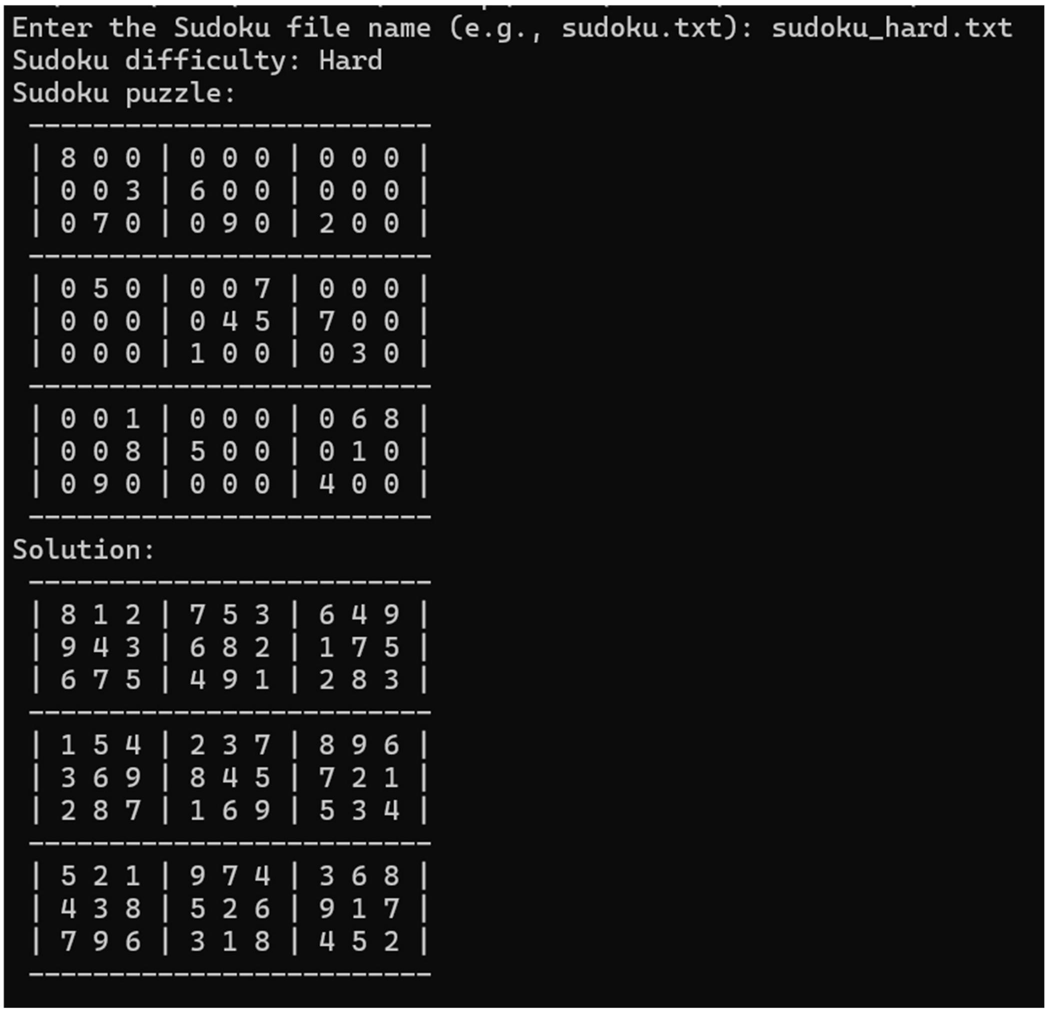
```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>


#define N 9  // Standard 9x9 Sudoku size


// Function to print the Sudoku grid

void print(int arr[N][N]) {

   printf(" -------------------------\n");

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

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

         if (j == 0)

            printf(" | ");

         printf("%d ", arr[i][j]);

         if ((j - 2) % 3 == 0)

            printf("| ");

      }

      printf("\n");

      if ((i - 2) % 3 == 0)

         printf(" -------------------------\n");

   }

}


// Function to check if it's safe to place a number in the grid

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

   for (int x = 0; x < N; x++)

      if (grid[row][x] == num || grid[x][col] == num)

         return 0;
```

```
    int startRow = row - row % 3, startCol = col - col % 3;
    for (int i = 0; i < 3; i++)
        for (int j = 0; j < 3; j++)
            if (grid[i + startRow][j + startCol] == num)
                return 0;

    return 1;
}

// Recursive function to solve the Sudoku puzzle
int solveSudoku(int grid[N][N], int row, int col) {
    if (row == N - 1 && col == N)
        return 1;

    if (col == N) {
        row++;
        col = 0;
    }

    if (grid[row][col] > 0)
        return solveSudoku(grid, row, col + 1);

    for (int num = 1; num <= 9; num++) {
        if (isSafe(grid, row, col, num)) {
            grid[row][col] = num;

            if (solveSudoku(grid, row, col + 1))
                return 1;
        }
        grid[row][col] = 0;
    }
    return 0;
}

// Function to read the Sudoku grid from a file and return the number of empty cells (zeros)
int readGridFromFile(int grid[N][N], const char *filename) {
    FILE *file = fopen(filename, "r");
    if (file == NULL) {
        printf("Error: Could not open file %s\n", filename);
        exit(1);
    }

    int zeroCount = 0;
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            if (fscanf(file, "%d", &grid[i][j]) != 1) {
                printf("Error reading file\n");
                fclose(file);
                exit(1);
            }
            if (grid[i][j] == 0)
                zeroCount++;  // Count number of empty cells (0s)
        }
    }

    fclose(file);
    return zeroCount;
```

```
}

// Function to determine the difficulty based on the number of empty cells
int determineDifficulty(int zeroCount) {
   if (zeroCount <= 20)
      return 1;  // Easy
   else if (zeroCount <= 40)
      return 2;  // Medium
   else
      return 3;  // Hard
}

int main() {
   int grid[N][N];
   char filename[100];

   // Step 1: Allow the user to enter the file name
   printf("Enter the Sudoku file name (e.g., sudoku.txt): ");
   scanf("%s", filename);

   // Step 2: Read the grid from the file and get the number of empty cells (zeros)
   int zeroCount = readGridFromFile(grid, filename);

   // Step 3: Determine difficulty based on the number of empty cells
   int difficulty = determineDifficulty(zeroCount);

   // Step 4: Switch case to handle difficulty
   switch (difficulty) {
      case 1:
            printf("Sudoku difficulty: Easy\n");
            break;
        case 2:
            printf("Sudoku difficulty: Medium\n");
            break;
        case 3:
            printf("Sudoku difficulty: Hard\n");
            break;
        default:
            printf("Unknown difficulty\n");
            exit(1);
    }

    // Step 5: Print the Sudoku puzzle before solving
    printf("Sudoku puzzle:\n");
    print(grid);

    // Step 6: Solve the Sudoku puzzle
    if (solveSudoku(grid, 0, 0)) {
        printf("Solution:\n");
        print(grid);
    } else {
        printf("No solution exists\n");
    }
    return 0;
}
```

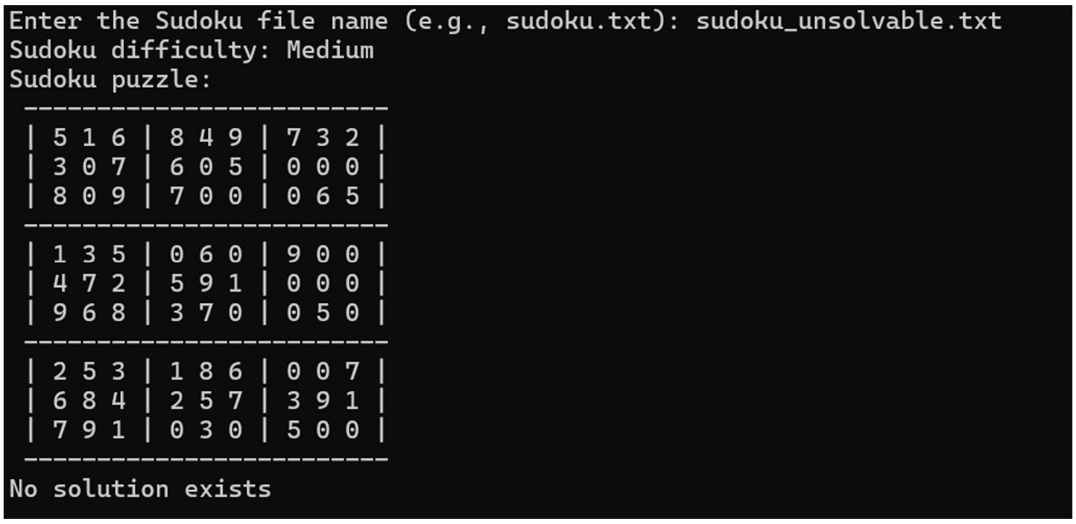
OUTPUT:

1) SOLVABLE SUDOKU



```
Enter the Sudoku file name (e.g., sudoku.txt): sudoku_hard.txt
Sudoku difficulty: Hard
Sudoku puzzle:
 -------------------------
 | 8 0 0 | 0 0 0 | 0 0 0 |
 | 0 0 3 | 6 0 0 | 0 0 0 |
 | 0 7 0 | 0 9 0 | 2 0 0 |
 -------------------------
 | 0 5 0 | 0 0 7 | 0 0 0 |
 | 0 0 0 | 0 4 5 | 7 0 0 |
 | 0 0 0 | 1 0 0 | 0 3 0 |
 -------------------------
 | 0 0 1 | 0 0 0 | 0 6 8 |
 | 0 0 8 | 5 0 0 | 0 1 0 |
 | 0 9 0 | 0 0 0 | 4 0 0 |
 -------------------------
Solution:
 -------------------------
 | 8 1 2 | 7 5 3 | 6 4 9 |
 | 9 4 3 | 6 8 2 | 1 7 5 |
 | 6 7 5 | 4 9 1 | 2 8 3 |
 -------------------------
 | 1 5 4 | 2 3 7 | 8 9 6 |
 | 3 6 9 | 8 4 5 | 7 2 1 |
 | 2 8 7 | 1 6 9 | 5 3 4 |
 -------------------------
 | 5 2 1 | 9 7 4 | 3 6 8 |
 | 4 3 8 | 5 2 6 | 9 1 7 |
 | 7 9 6 | 3 1 8 | 4 5 2 |
 -------------------------
```

2) UNSOLVABLE SUDOKU



```
Enter the Sudoku file name (e.g., sudoku.txt): sudoku_unsolvable.txt
Sudoku difficulty: Medium
Sudoku puzzle:
 -------------------------
 | 5 1 6 | 8 4 9 | 7 3 2 |
 | 3 0 7 | 6 0 5 | 0 0 0 |
 | 8 0 9 | 7 0 0 | 0 6 5 |
 -------------------------
 | 1 3 5 | 0 6 0 | 9 0 0 |
 | 4 7 2 | 5 9 1 | 0 0 0 |
 | 9 6 8 | 3 7 0 | 0 5 0 |
 -------------------------
 | 2 5 3 | 1 8 6 | 0 0 7 |
 | 6 8 4 | 2 5 7 | 3 9 1 |
 | 7 9 1 | 0 3 0 | 5 0 0 |
 -------------------------
No solution exists
```