Project 2: Sudoku Solver

A C Program that takes a matrix of 9x9 as an input off sudoku with empty spaces as 0 and then displays the solution in the form of matrix as output.

What is SUDOKU?

Sudoku is a puzzle game designed for a single player, much like a crossword puzzle. The puzzle itself is nothing more than a grid of little boxes called "cells". They are stacked nine high and nine wide, making 81 cells total. The puzzle comes with some of the cells (usually less than half of them) already filled in, like this:

		6		5	4	9		
1				6			4	2
7				8	9			
	7				5		8	1
	5		3	4		6		
4		2						
	3	4				1		
9			8				5	
			4			3		7

Each little square is called a "cell." Most often, Sudoku cells are filled with numbers (1-2-3-4-5-6-7-8-9), but sometimes pictures or Japanese symbols are used instead.

Technology Used :

1)Programming Language :- C

2) Software :- Visual Studio Code (1.52.1)

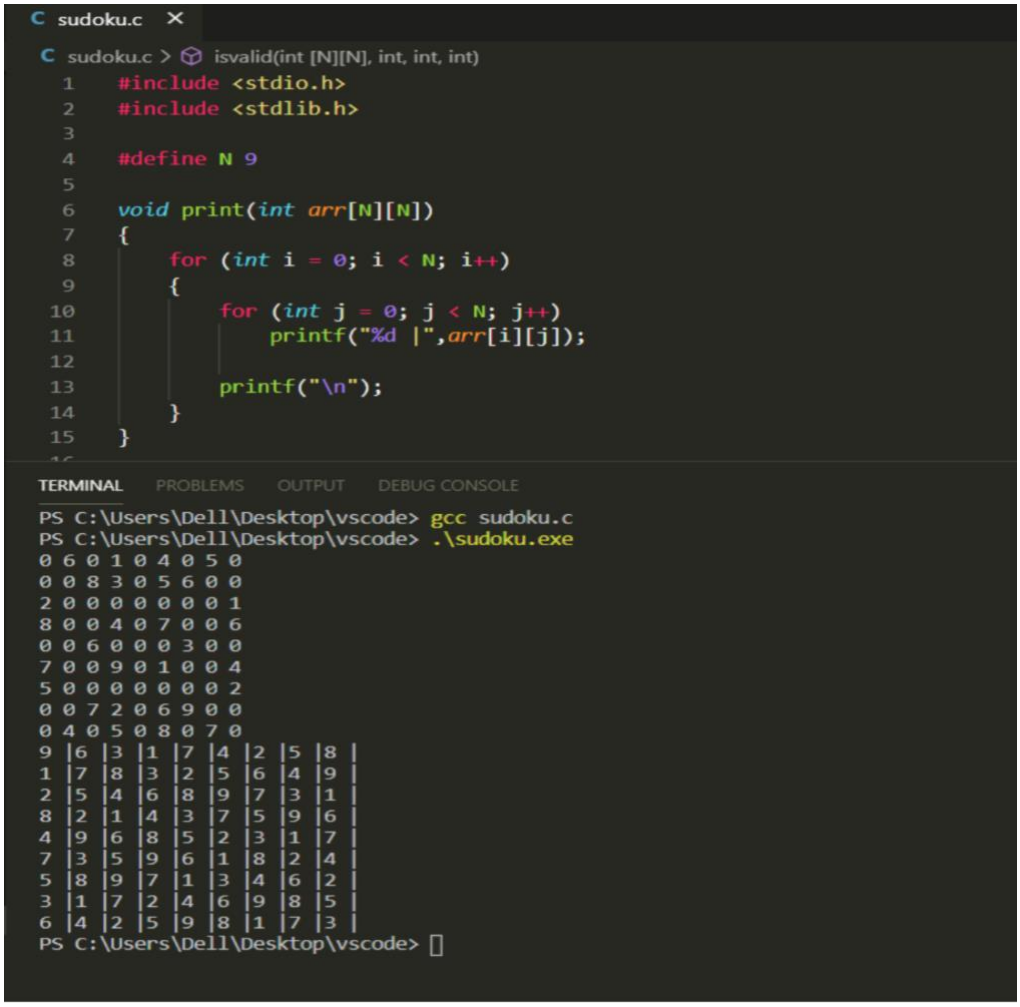
3)Platforms :- Geeks for Geeks:

https://www.geeksforgeeks.org/sudoku-backtracking-7/ (For Logic understanding)

Steps to run project :-

1) Download ZIP and extract the file on your local system or clone repository using below command in command prompt :

- 2) Open cloned file in Visual Studio Code
- 3) Open Terminal >> Run Build Task.. (or Ctrl + F7)
- 4) In Terminal below , after successful build.
- Run following commands :
  • gcc sudoku.c (program_name.c) • .\sudoku.exe



Method / Approach Used :

Question: Implement a Sudoku solver. (a classic Sudoku board is 9x9)

The Rules of Sudoku:

Approach : **The backtracking approach**.

We are working with numbers 1 thorugh 9. Each number can only

appear once in a: Row Column And one of the 3 x 3 sub-boxes.

We can traverse and place an entry in empty cells one by one. We

then check the whole board to see if it is still valid. If it is we continue

our recursion. If it is not we do not even follow that path. When all

entries have been filled, we are finished. The 3 Keys To Backtracking:

Our Choice What we place in an empty cell that we see. Our

Constraints Standard Sudoku constraints. We cannot make a

placement that will break the board. Our Goal Place all empty spaces

on the board. We will know we have done this when we have placed a

valid value in the last cell in our search which programmatically will be

the bottom right cell in the 2D matrix. Validating Placements Before

We Place Them INSIGHT: We could traverse every row, column, and

3x3 subgrid to perform validation but at every decision point we know

that we are adding onto a grid that is already valid. So we just check

the row, column, and subgrid of the item that has been added.

Whenever backtracking we know that every decision that we made

stayed within the constraint that we started with being that the board

was valid. Complexities Since Sudoku is traditionally 9x9 we can't

really discuss complexities because nothing can scale. Solving

Sudoku generalized to n x n boards is a NP-Complete problem so we

know for sure that our time complexity is exponential at the very least.