KATHMANDU UNIVERSITY

DHULIKHEL KAVRE

SCHOOL OF ENGINEERING

Report on my Mini Project
SUDOKU SOLVER

SUBMITTED BY:

NAME: Ayush Regmi

Roll No.: 39

Group: CE (B)

1st Year / 1st Semester

SUBMITTED TO:

Dr. Rajani Chulyadyo

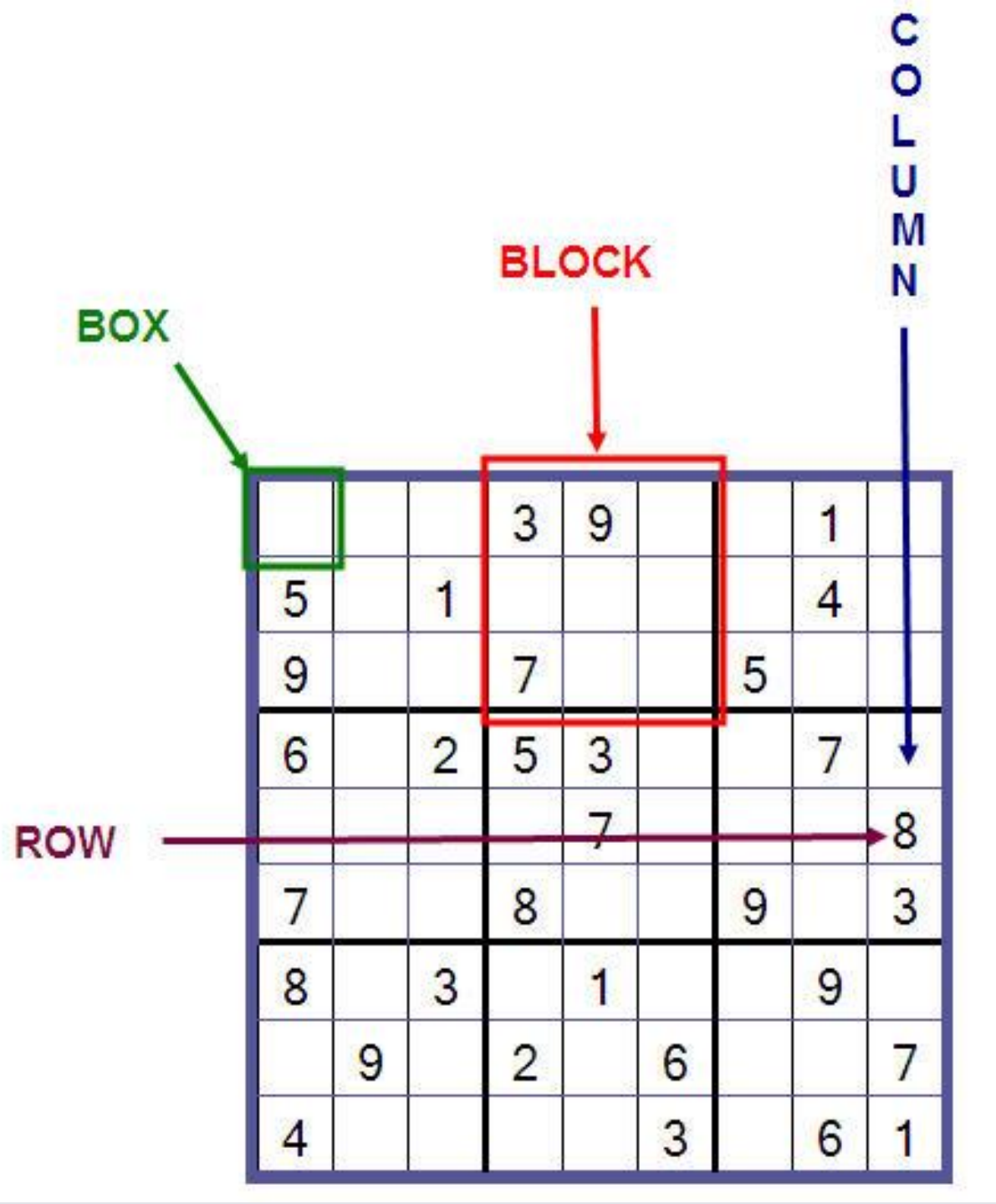
Department of Computer Science and Engineering

SUDOKU GAME

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

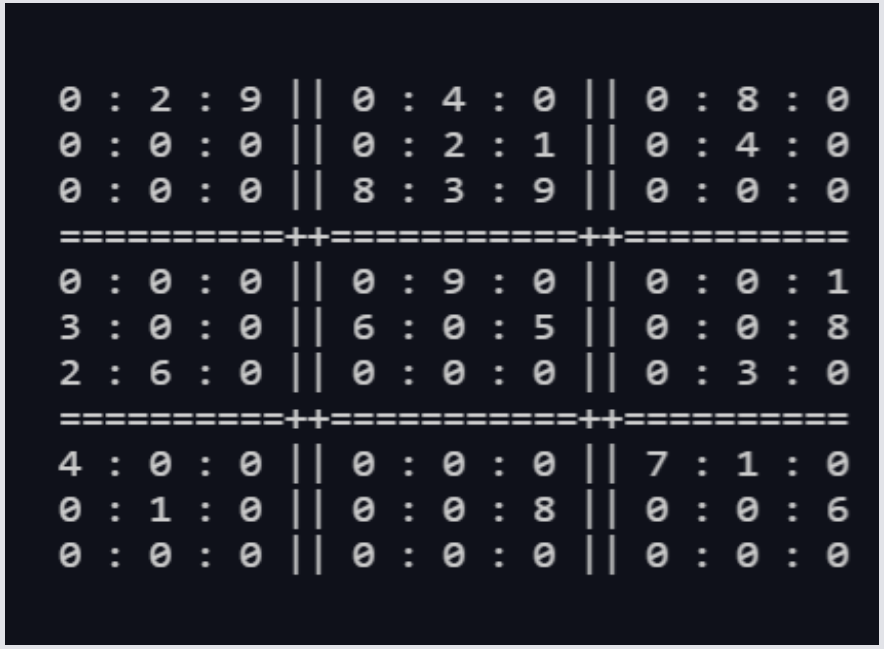
Rules of Sudoku

Given a partially filled 9×9 2D array ‘grid[9][9]’, the goal is to assign digits (from 1 to 9) to the empty cells so that every row, column, and sub-grid of size 3×3 contains exactly one instance of the digits from 1 to 9.



Input v/s Output

BEFORE SOLVING:



AFTER SOLVING:

Approach

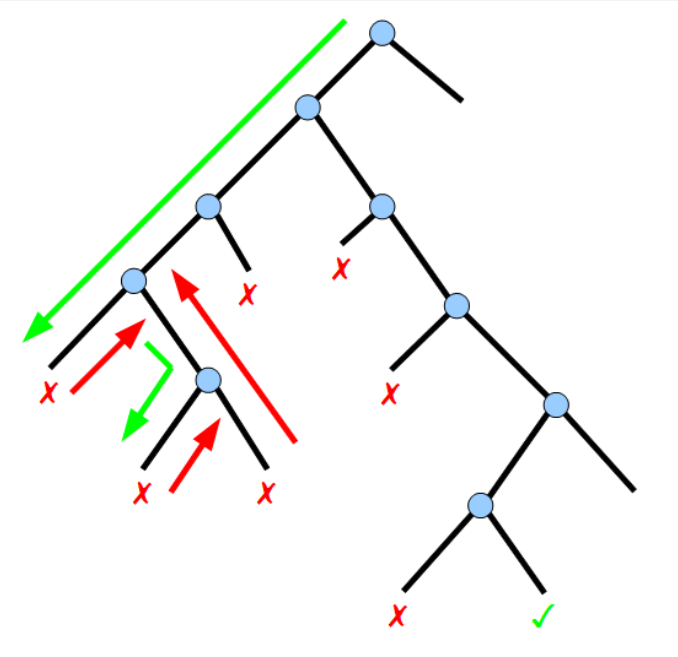
Sudoku can be solved by one by one assigning numbers to empty cells. Before assigning a number, check whether it is safe to assign. Check that the same number is not present in the current row, current column and current 3X3 sub-grid. After checking for safety, assign the number, and recursively check whether this assignment leads to a solution or not. If the assignment doesn’t lead to a solution, then try the next number for the current empty cell. And if none of the number (1 to 9) leads to a solution, return false and print no solution exists.

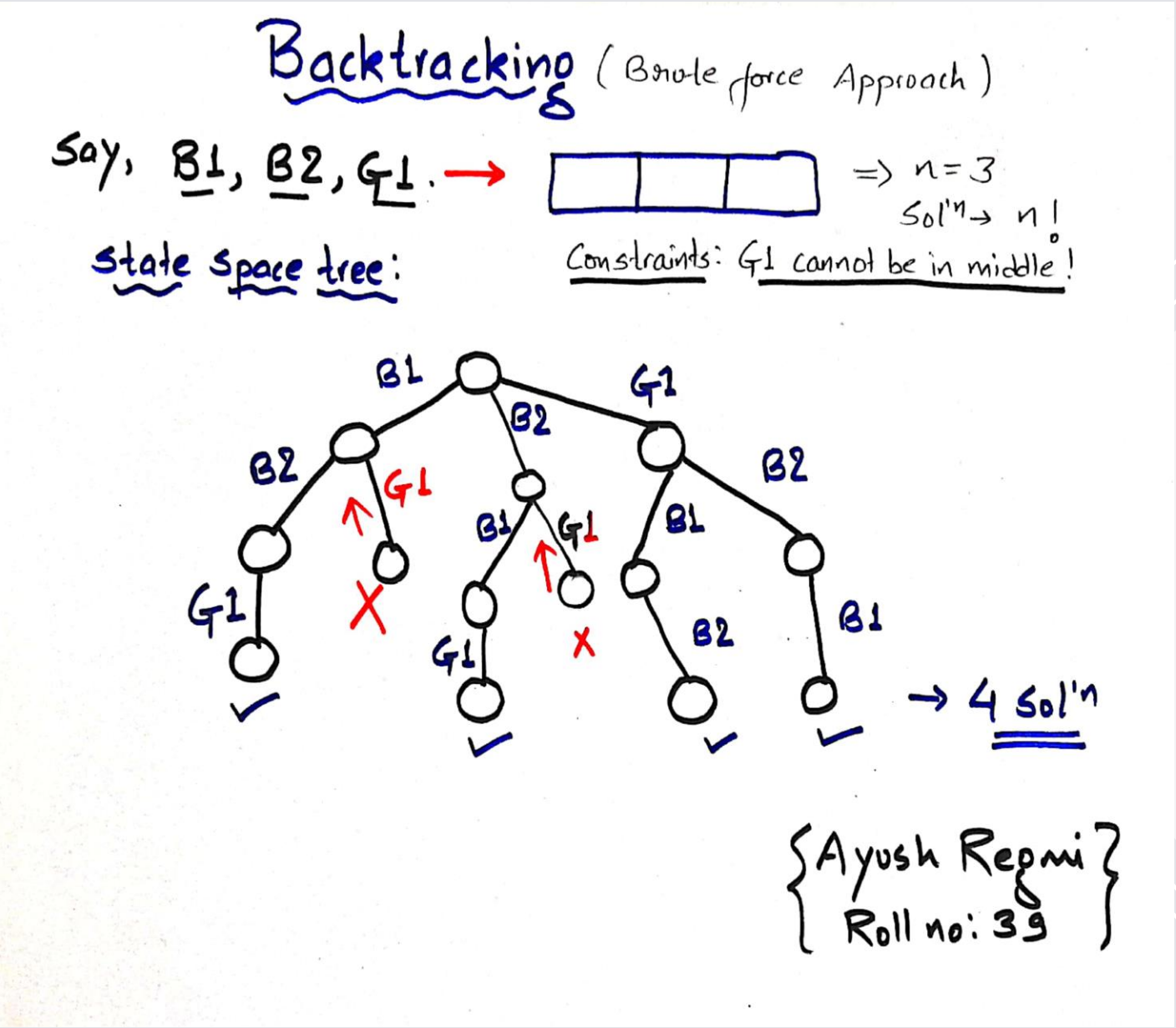
Backtracking Algorithm

According to the wiki definition,

Backtracking *can be defined as a general algorithmic technique that considers searching every possible combination in order to solve a computational problem.*

Backtracking is an algorithmic-technique for solving problems recursively by trying to build a solution incrementally, one piece at a time, removing those solutions that fail to satisfy the constraints of the problem at any point of time





Permutation

Permutation of 2 boys and 1 girl in the grid,

Where girl cannot seat in the middle of 2 boys.

ALGORITHM for Sudoku

Create a function that checks after assigning the current index the grid becomes unsafe or not. Keep Hashmap for a row, column and boxes. If any number has a frequency greater than 1 in the hashMap return false else return true; hashMap can be avoided by using loops.

Create a recursive function that takes a grid.

Check for any unassigned location. If present then assign a number from 1 to 9, check if assigning the number to current index makes the grid unsafe or not, if safe then recursively call the function for all safe cases from 0 to 9. if any recursive call returns true, end the loop and return true. If no recursive call returns true then return false.

If there is no unassigned location then return true.

About My Project:

This mini project 'Sudoku Solver' contains five .C files and four .txt files.

The main.c file is (literally) a main file where user can enter the data for Sudoku Problem.

PrintBoard.c file prints the Sudoku Board.

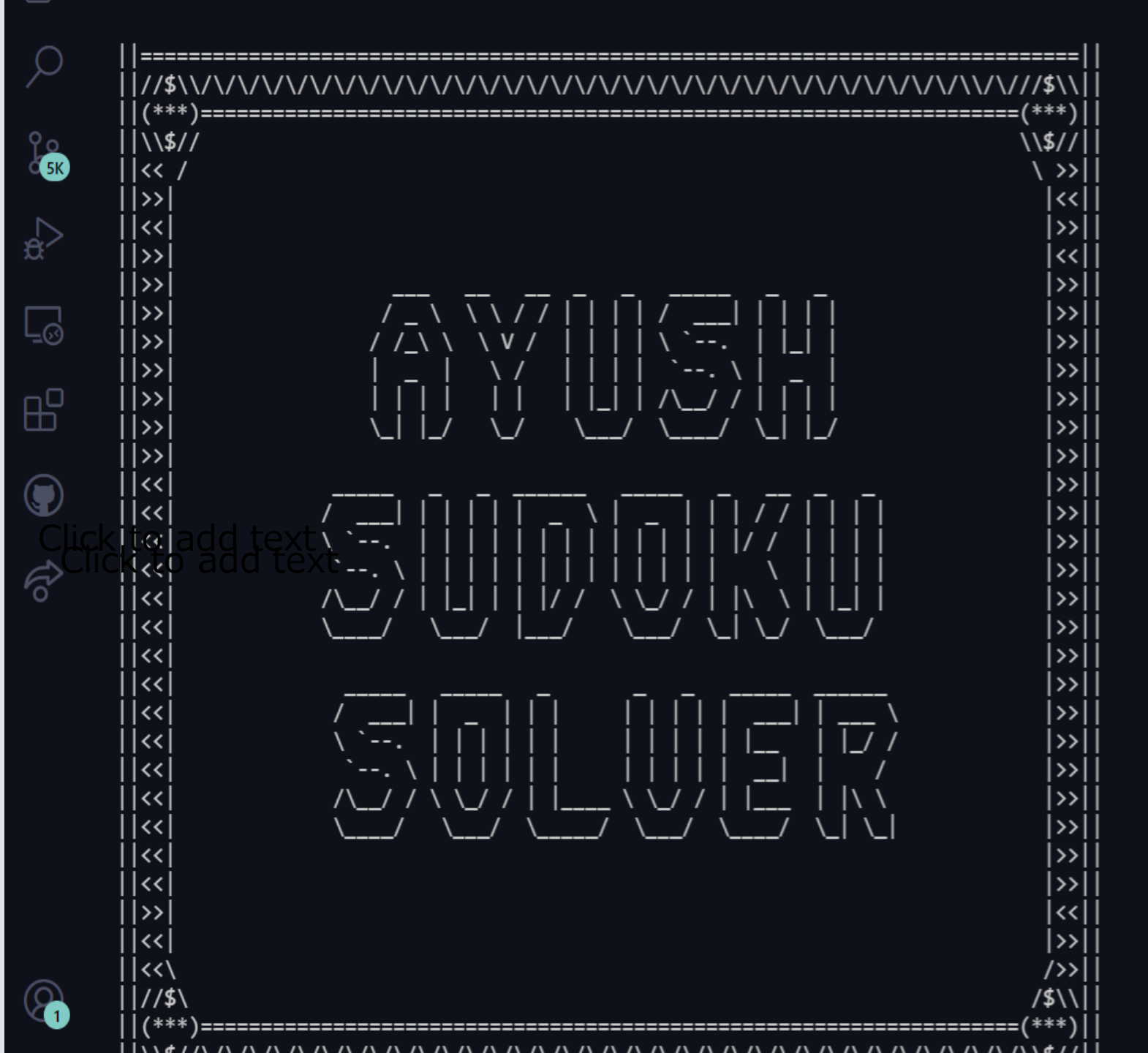
SolvingFunctions.c file contains all the functions and algorithms to solve Sudoku

At last, ASCII_ART.c file displays the ASCII Arts from 4 Text files

Title Of the Project

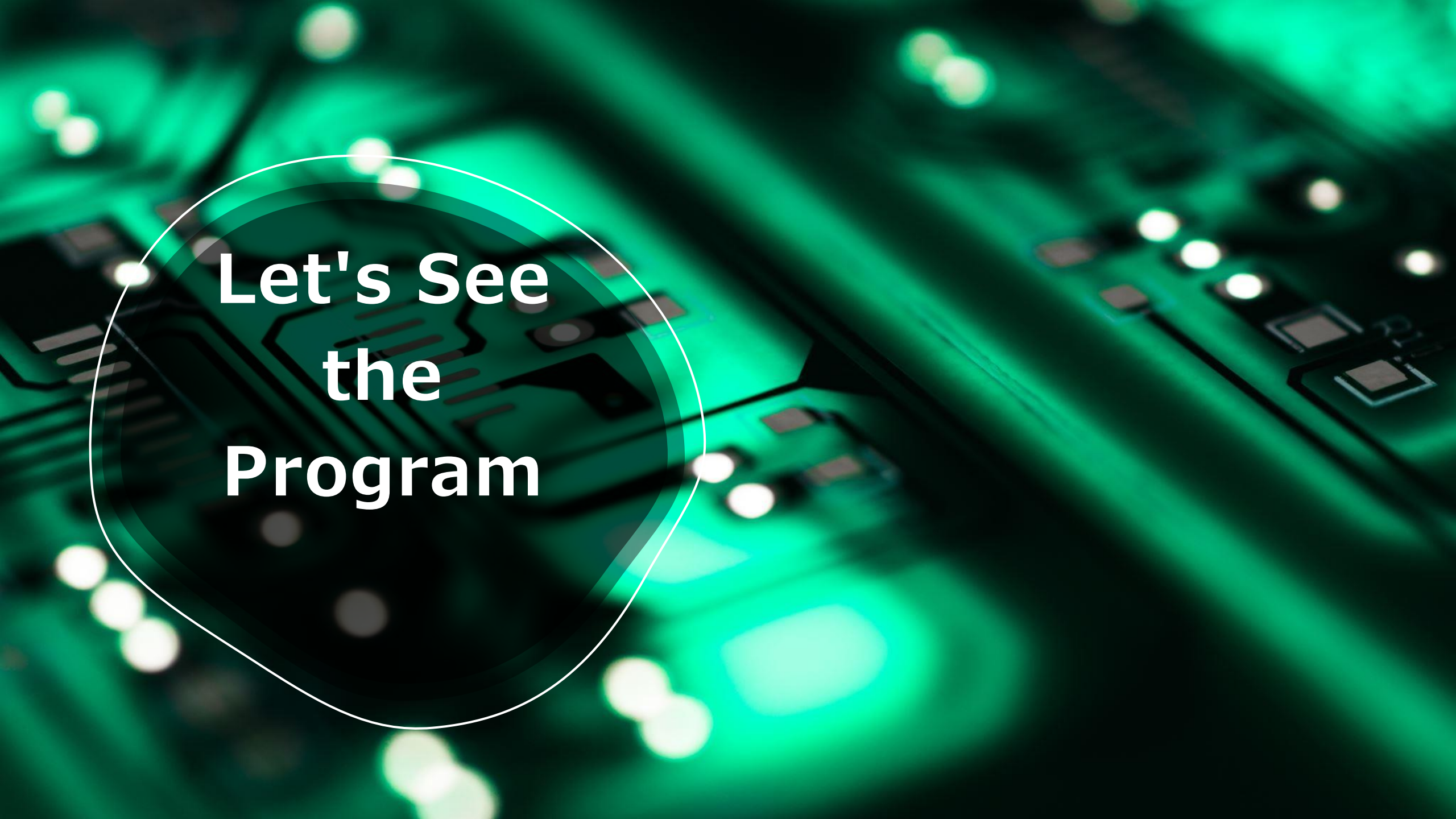
Click to add text

Click to add text



When 'S' key is Pressed, The program solve the given Sudoku problem and gives the best probable solution

The Final Solution


Let's See
the
Program

The Main File

The user/programmer is supposed to enter the valid Sudoku problem in th is Board

** The Dimension of the board is 9 x9:*

** 9 rows and 9 columns*

** 9 Sub-grids with 3 rows and 3 columns each*

```
/**
 * @brief The Sudoku Board!!
 *
 * The user/programmer is supposed to enter the valid Sudoku problem in this Board
 * The Dimension of the board is 9x9:
 * 9 rows and 9 columns
 * 9 Sub-grids with 3 rows and 3 columns each
 *
 */
int board[N][N] =   {{2, 2, 9,  0, 4, 0,  0, 8, 0},
                     {0, 0, 0,  0, 2, 1,  0, 4, 0},
                     {0, 0, 0,  8, 3, 9,  0, 0, 0},

                     {0, 0, 0,  0, 9, 0,  0, 0, 1},
                     {3, 0, 0,  6, 0, 5,  0, 0, 8},
                     {2, 6, 0,  0, 0, 0,  0, 3, 0},

                     {4, 0, 0,  0, 0, 0,  7, 1, 0},
                     {0, 1, 0,  0, 0, 8,  0, 0, 6},
                     {0, 0, 0,  0, 0, 0,  0, 0, 0}};


printf("\n\n");
```

The PrintBoard File

PrintBoard function
iss a Utility Function to Display The Sudoku Bo
ard in the Terminal

** This function takes a 2-*
Dimesional array of Sudoku Grid with number o
f

rows and number of columns,

and displays the Grid in the terminal.

** A number of for loops are used to make 2-dimensional Grid and make the display visually attractive.*

PrintBoard.c > printBoard(int [][N], int, int)

```c
 */
void printBoard(int grid[][N], int row, int col)
{
    for (row = 0; row < N; row++)
    {
        /**
         * @brief If statement to print the Boarder after every 3 rows in the Grid.
         *
         */
        if (row % 3 == 0 && row != 0)
        {
            printf("==========++==========++==========\n");
        }

        for (col = 0; col < N; col++)
        {
            /**
             * @brief If statement to print the Boarder after every 3 columns in the Grid.
             *
             */
            if (col % 3 == 0 && col != 0)
            {
                printf("|| ");
            }

            printf("%d ", grid[row][col]);

            /**
             * @brief If statement to print the dotted boundary between every columns in the Grid.
             *
             */
            if (col != 8 && col != 2 && col != 5)
            {
                printf(": ");
            }
        }
        printf("\n");
    }
}
```

The Solving Function files

- *The START function is a Utility Function which displays the whole Solution of this problem.*
- *When the user presses 'S' button then the START function displays the Un solved Grid to the Users.*
- *After that it displays the Solution of that Sudoku Grid.*

main.c | PrintBoard.c | SolvingFunctions.c | ASCII_ARTS.c | image.txt

SolvingFunctions.c > isValidBlock(int [][N], int, int, int)

```c
/* The START function displays the whole Solution of this problem */
void START(int board[][N], int row, int col)
{
    /* This function is also from the ASCII_ARTS Header file
    wich displays the word 'Before Solving' in the ASCII Art format */
    BeforeSolving();
    printf("\n\n");

    /* This fucntion is from the PrintBoard Header file
    which prints the Sudoku Grid provided by the user in unsolved format */
    printBoard(board, row, col);


    if (SolveBoard(board, row, col))
    {
        printf("\n\n");

        /* This function is also from the ASCII_ARTS Header file
        wich displays the word 'After Solving' in the ASCII Art format */
        AfterSolving();
        printf("\n\n");

        /* This fucntion is from the PrintBoard Header file
        which prints the Solution of the Sudoku Grid provided by the user*/
        printBoard(board, row, col);
    }

    else
    {
        printf("\n\n");

        /* This function if from ASCII_ARTS Header file
        which Displays the error message  in the ASCII Art format
        if the provided sudoku board is invalid*/
        Error();
        printf("\n\n");
    }
}
```

The Solve Function

The SolveBoard function assign valid values to all Blank locations provided by find_empty function

* *This SolveBoard function is the Heart of the entre solution.*

* *This is the function wich triggers backtracking.*

main.c | PrintBoard.c | SolvingFunctions.c | ASCII_ARTS.c | image.txt | BeforeSolving.txt

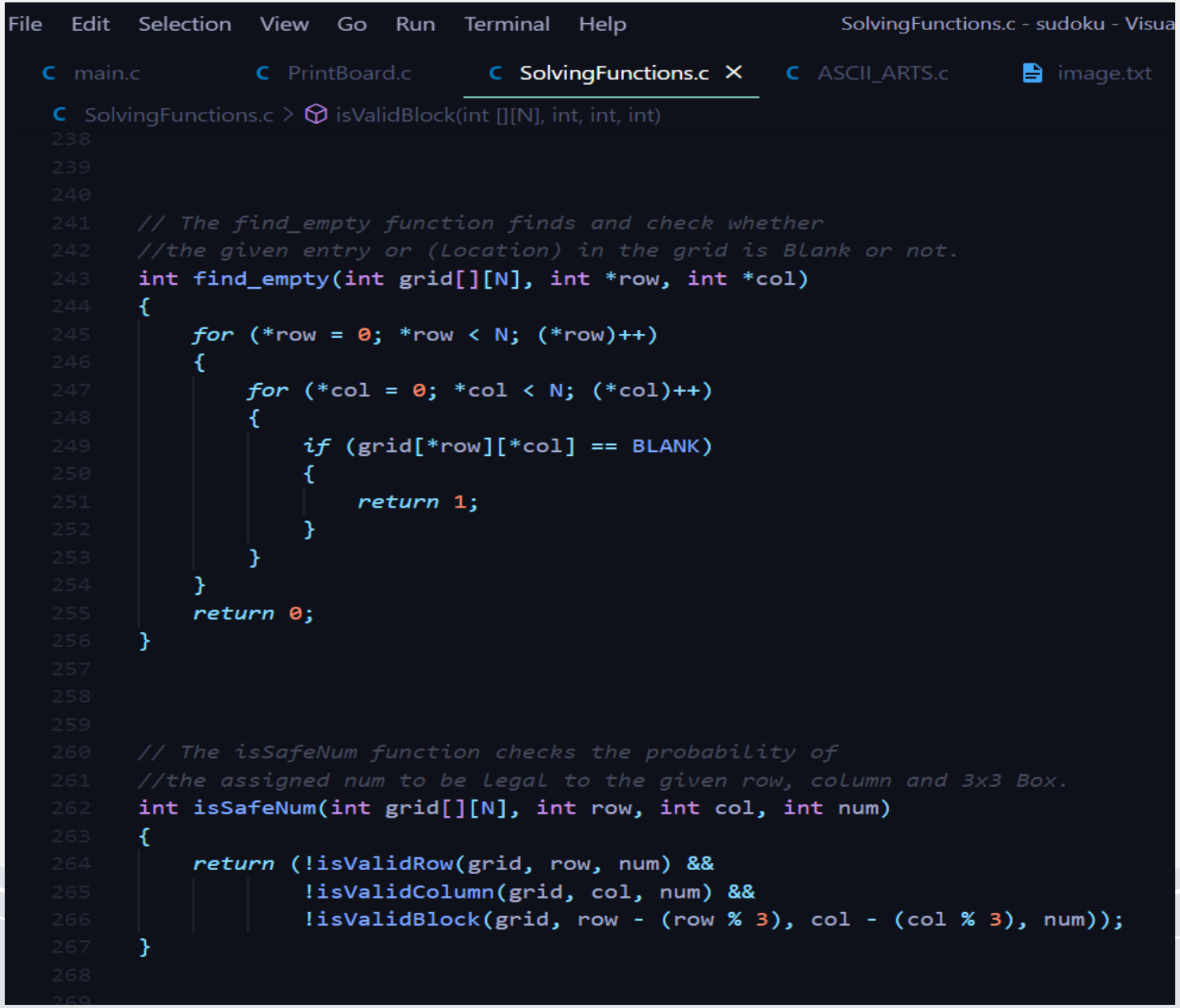
SolvingFunctions.c > isValidBlock(int [][N], int, int, int)

```c

/* The SolveBoard function assign valid values to all Blank locations provided by find_empty function
 *
 * This SolveBoard function is the Heart of the entire solution.
 *
 * This is the function wich triggers backtracking. */
int SolveBoard(int grid[][N], int row, int col) {

    // check if there are no Blank location,
    // If true, We are done!!!
    if (!find_empty(grid, &row, &col))
    {
        return 1; // success!
    }

    // consider num => 1, 2, 3, 4, 5, 6, 7, 8, 9
    for (int num =1; num <= N; num++)
    {
        // The isSafeNum function checks the probability of
        // the assigned num to be legal to the given row, column and 3x3 Box.
        if (isSafeNum(grid, row, col, num))
        {
            grid[row][col] = num;

            // Return, if success !!
            if (SolveBoard(grid, row, col))
            {
                return 1; // Success!!
            }

            //Failure, unmake & try again i.e. BackTrack!!!
            grid[row][col] = BLANK;
        }

    }

    // This return triggers the backtracking
    return 0;
}
```

The Solving Function file

The find_empty function finds an d check whether the given entry or (Location) in the grid is Blank or not

The isSafeNum function check t he probability of the assigned n um to be legal to the given row, column and 3x3 Box.


File Edit Selection View Go Run Terminal Help
SolvingFunctions.c - sudoku - Visua
main.c
PrintBoard.c
SolvingFunctions.c
ASCII_ARTS.c
image.txt
SolvingFunctions.c > isValidBlock(int [][N], int, int, int)
238
239
240
241 // The find_empty function finds and check whether
242 //the given entry or (Location) in the grid is Blank or not.
243 int find_empty(int grid[][N], int *row, int *col)
244 {
245 for (*row = 0; *row < N; (*row)++)
246 {
247 for (*col = 0; *col < N; (*col)++)
248 {
249 if (grid[*row][*col] == BLANK)
250 {
251 return 1;
252 }
253 }
254 }
255 return 0;
256 }
257
258
259
260 // The isSafeNum function checks the probability of
261 //the assigned num to be legal to the given row, column and 3x3 Box.
262 int isSafeNum(int grid[][N], int row, int col, int num)
263 {
264 return (!isValidRow(grid, row, num) &&
265 !isValidColumn(grid, col, num) &&
266 !isValidBlock(grid, row - (row % 3), col - (col % 3), num));
267 }
268
269

Is Valid Functions

isValid functions:

- *checks the validity of an assigned e ntry in the specified row*
- *Checks validity of an assigned entry in the specified column*
- *checks the validity of an assigned e ntry in the specified row*

main.c | PrintBoard.c | SolvingFunctions.c | ASCII_ARTS.c | image.txt | BeforeSolvi

SolvingFunctions.c > isValidBlock(int [][N], int, int, int)

```c
// isValidColumn function checks the validity of an assigned entry in the specified column
int isValidColumn(int grid[][N], int col, int num)
{

    for (int row = 0; row < N; row++)
    {
        if (grid[row][col] == num)
        {
            return 1;
        }
    }
    return 0;
}


// isValidRow function checks the validity of an assigned entry in the specified row
int isValidRow(int grid[][N], int row, int num)
{
    for (int col = 0; col < N; col++)
    {
        if (grid[row][col] == num)
        {
            return 1;
        }
    }
    return 0;
}
```

File Edit Selection View Go Run Terminal Help SolvingFunctions.c - sudoku - Visual

main.c PrintBoard.c SolvingFunctions.c ASCII_ARTS.c image.txt

SolvingFunctions.c > isValidBlock(int [][N], int, int, int)

```c
299
300
301
302    // isValidBlock function checks the validity of
303    //an assigned entry in the specified 3x3 sub-grid (Box)
304    int isValidBlock(int grid[][N], int boxStartRow, int boxStartCol, int num)
305    {
306        for (int row = 0; row < 3; row++)
307        {
308            for (int col = 0; col < 3; col++)
309            {
310                if (grid[row + boxStartRow][col + boxStartCol] == num)
311                {
312                    return 1;
313                }
314
315            }
316
317        }
318        return 0;
319    }
```

Thank You So Much !!!