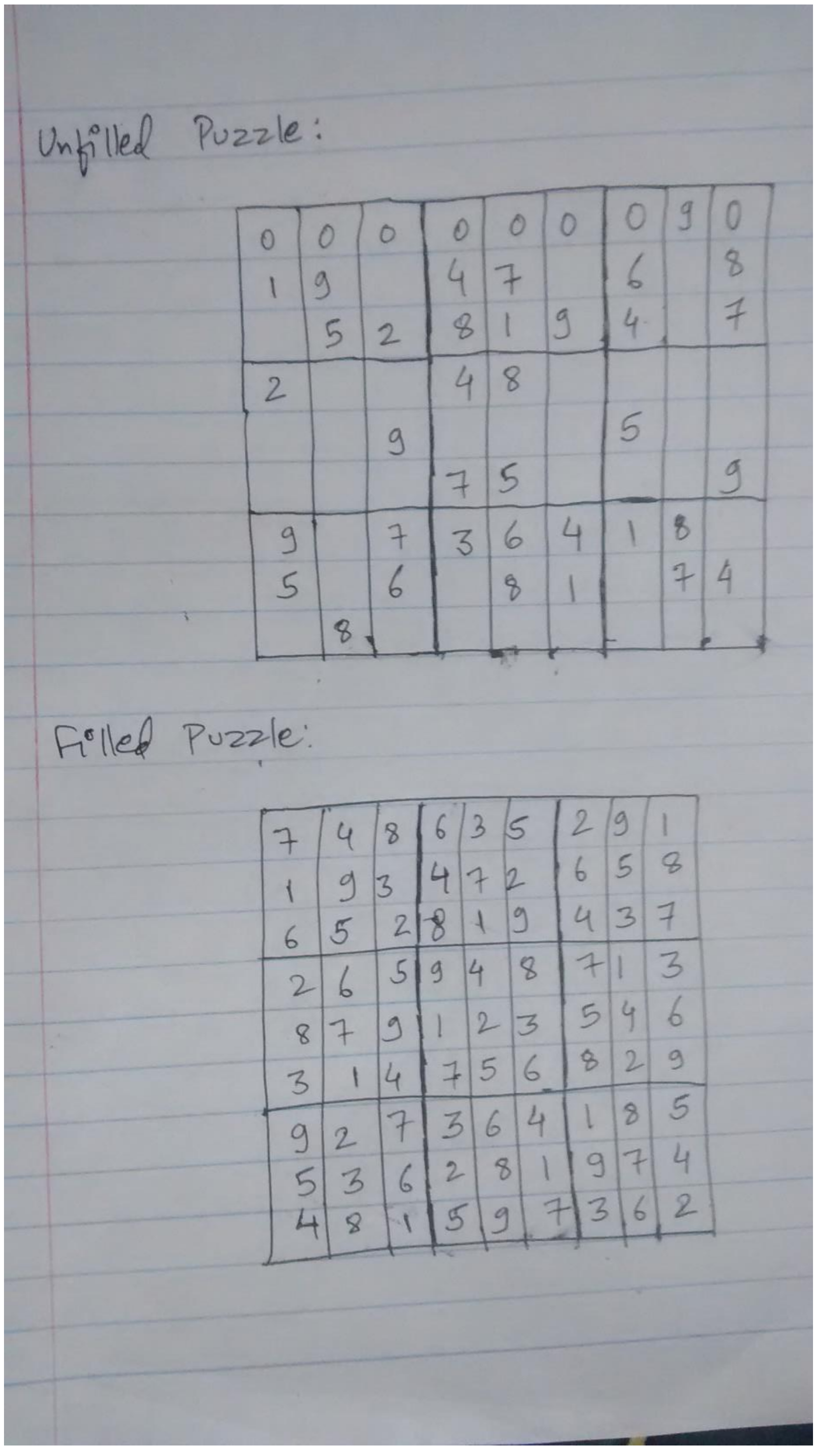
Pradip Lamsal

Data Structures Project

A Sudoku Solver program using C

1.

Manually constructed and filled Sudoku



Unfilled Puzzle:

0	0	0	0	0	0	0	9	0
1	9		4	7		6		8
	5	2	8	1	9	4		7
2			4	8				
		9				5		
			7	5				9
9		7	3	6	4	1	8	
5		6		8	1		7	4
	8							

Filled Puzzle:

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

2. Algorithm for solving the Sudoku puzzle:

1. Iterate through the manually constructed 9*9 Sudoku board to pick an empty cell.

2. Choose a number between 1 to 9(usually starting from 1) and check to see if the number has any duplicates along the rows, columns, and the 3x3 box the cell is in.

3. If there exists a duplicate, follow the same process as in step 2 for the number after it until the first one with no duplicates is found.

4. When the number is found, assign it to the particular cell.

5. If all the numbers from 1 to 9 are tried and still no solution is found, print that no solution was found.

6. If the number is found, follow steps 1 to 4 recursively until all of the empty cells in the board are filled.

7. Print the completed Sudoku board.

3. Pseudocode:

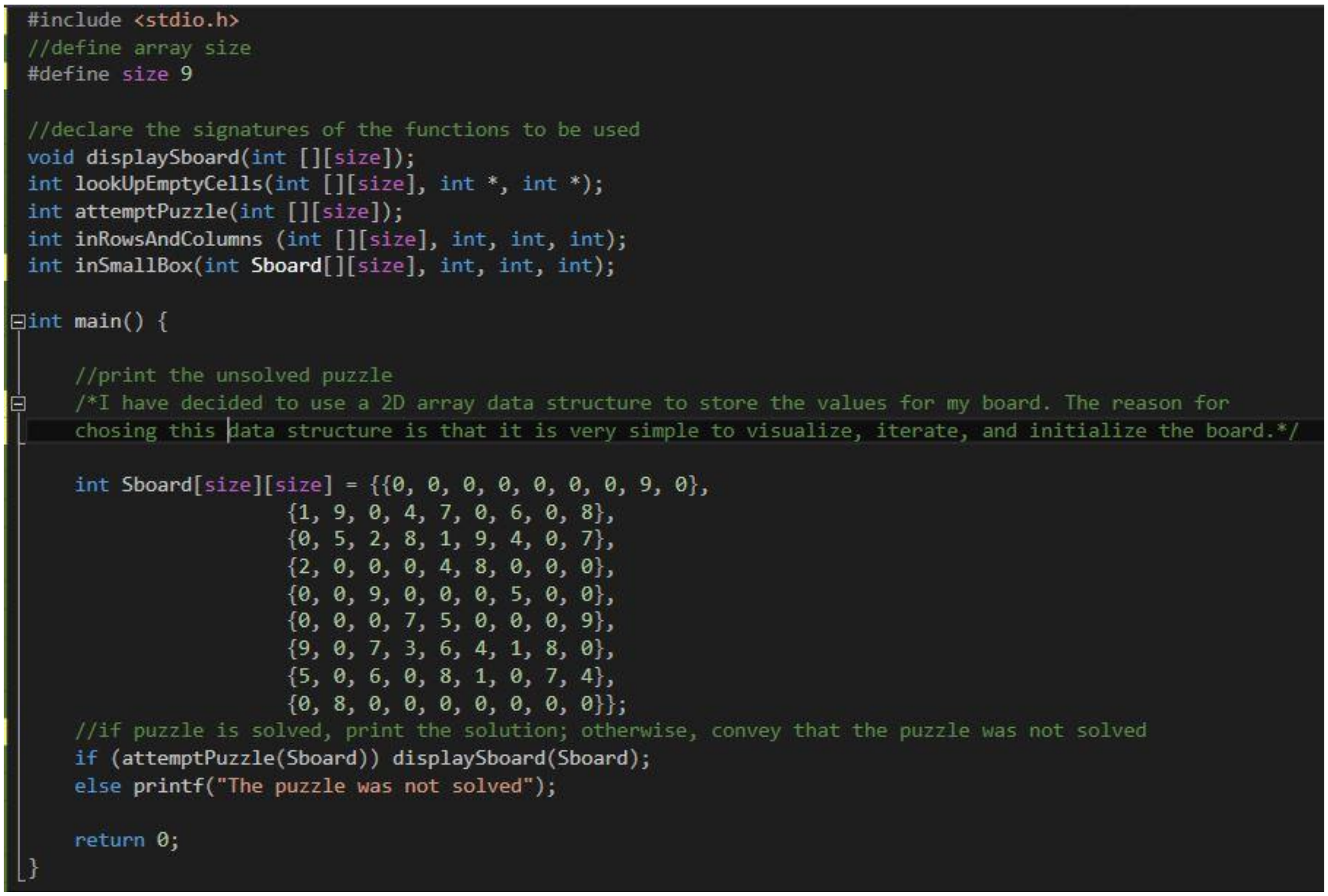
```
define size 9


//function that prints the board when the puzzle is solved
displaySboard(Sboard[size][size]) :
    for (row = 0 to size):
        for (coln = 0 to size):
            Print(Sboard[row][coln])




//function to find in there is any empty cell left in the board.
lookUpEmptyCells(Sboard[size][size], *row, *coln):
    for (*row = 0 to size) :
        for (*coln = 0 to size):
            if (Sboard[*row][*coln] == 0)    return 1




//function that actually inserts the appropriate value into the puzzle one by one
attemptPuzzle(Sboard[size][size]):
    row = 0; coln = 0
    if (!lookUpEmptyCells()) , then  return 1

    for (value = 1 to size) :
        if (!inRowsAndColumns() && !inSmallBox()) are true, then do
            Sboard[row][coln] = value
            if (attemptPuzzle(Sboard)) returns 1, then return 1
            else Sboard[row][coln] = 0
```

```
//function to check if the value is along the rows or the columns along the cell
inRowsAndColumns (Sboard, row, coln, value):
        for (i = 0 to 9):
                if ((Sboard[row][i] == value) or (Sboard[i][coln] == value)) is true, then return 1




//function to check if the value is present in any cells inside the 3x3 box the cell is in
inSmallBox(Sboard, row, coln, value):
          if (row < 3) row = 0
          else if (row < 6) row = 3
          else row = 6

          if (coln < 3) coln = 0
          else if (coln < 6) coln = 3
          else coln = 6

          for (int i = row to row+3):
            for (int j = coln to coln+3):
              if (Sboard[i][j] == value), then return 1




//main function
main():
        Sboard[size][size] = {{....}, {....}, ....}
        if (attemptPuzzle(Sboard)) returns 1, then displaySboard(Sboard)
        else Print "The puzzle was not solved"
```

4. The C program with functions, codes, test cases, and results:

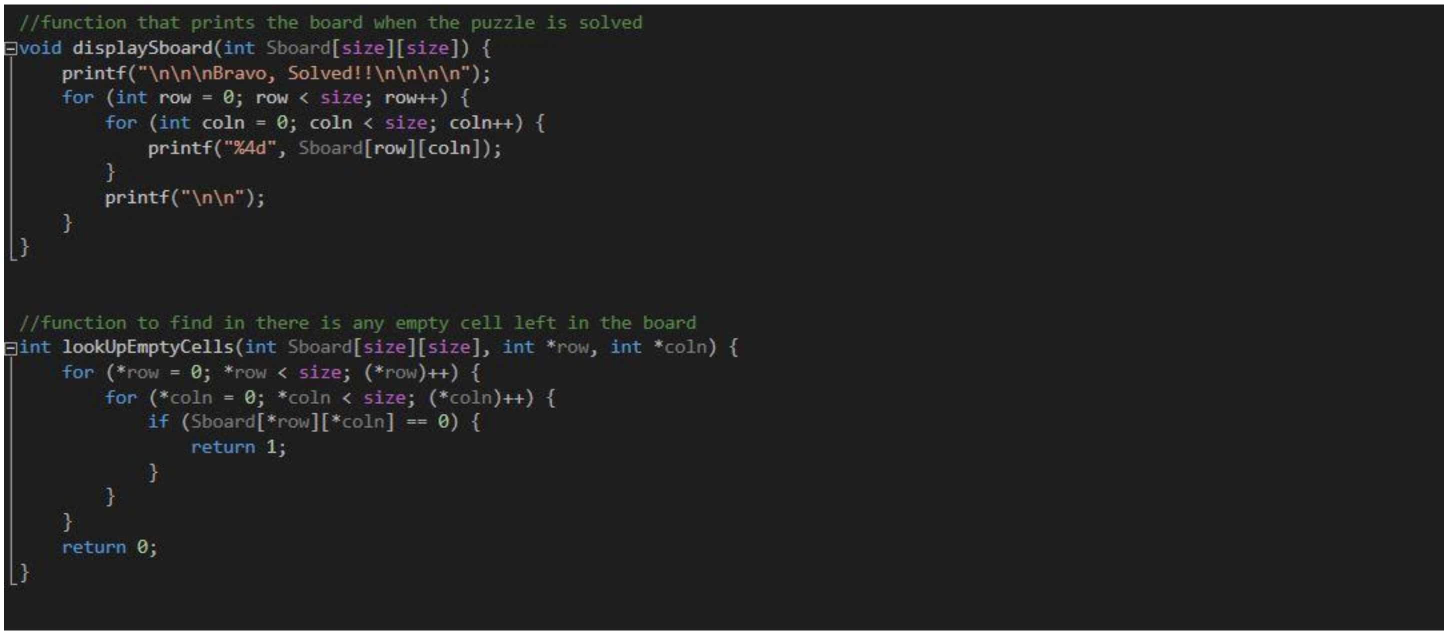
In my program, I have implemented a linear data structure called array which are built using primitive data types like integers. The Sudoku board that I have used is a two dimensional array built upon integers(Sboard[size][size]). I specifically chose a two dimensional array for my Sudoku board because it is very simple to visualize and iterate over. Moreover, it is possible to hard-code the initial unfilled Sudoku board using an array and the process is quite straightforward. Other data structures like Trees, Graphs, Heaps may have been inefficient to implement. I have also made use of pointers in my program so that I would not have to pass the values as arguments in every function as pointer updates the new value for other functions without having to be returned.

main function:



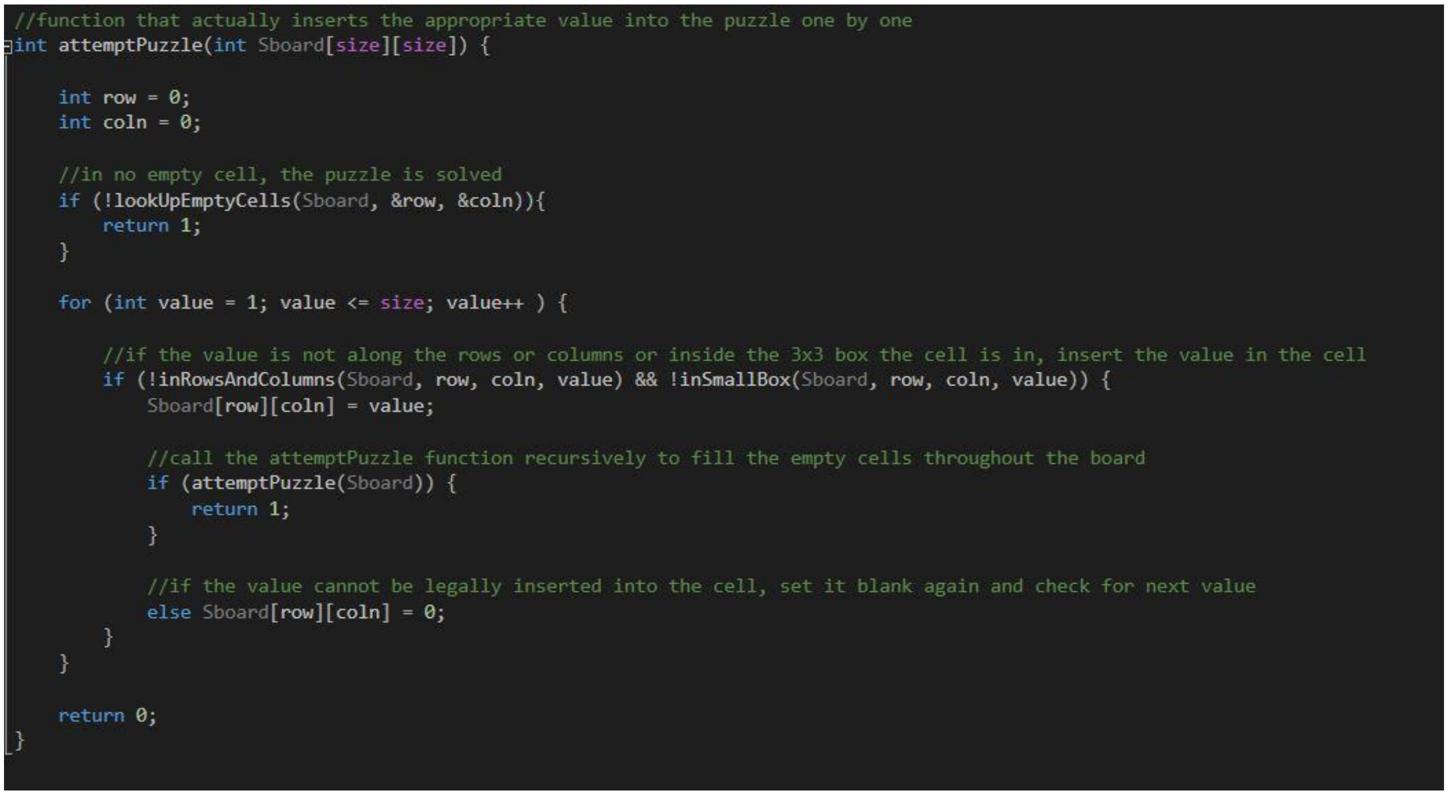
```c
#include <stdio.h>
//define array size
#define size 9

//declare the signatures of the functions to be used
void displaySboard(int [][size]);
int lookUpEmptyCells(int [][size], int *, int *);
int attemptPuzzle(int [][size]);
int inRowsAndColumns (int [][size], int, int, int);
int inSmallBox(int Sboard[][size], int, int, int);

int main() {

    //print the unsolved puzzle
    /*I have decided to use a 2D array data structure to store the values for my board. The reason for
    chosing this data structure is that it is very simple to visualize, iterate, and initialize the board.*/

    int Sboard[size][size] = {{0, 0, 0, 0, 0, 0, 0, 9, 0},
                              {1, 9, 0, 4, 7, 0, 6, 0, 8},
                              {0, 5, 2, 8, 1, 9, 4, 0, 7},
                              {2, 0, 0, 0, 4, 8, 0, 0, 0},
                              {0, 0, 9, 0, 0, 0, 5, 0, 0},
                              {0, 0, 0, 7, 5, 0, 0, 0, 9},
                              {9, 0, 7, 3, 6, 4, 1, 8, 0},
                              {5, 0, 6, 0, 8, 1, 0, 7, 4},
                              {0, 8, 0, 0, 0, 0, 0, 0, 0}};
    //if puzzle is solved, print the solution; otherwise, convey that the puzzle was not solved
    if (attemptPuzzle(Sboard)) displaySboard(Sboard);
    else printf("The puzzle was not solved");

    return 0;
}
```

displaySboard() function and lookUpEmptyCells() functions:



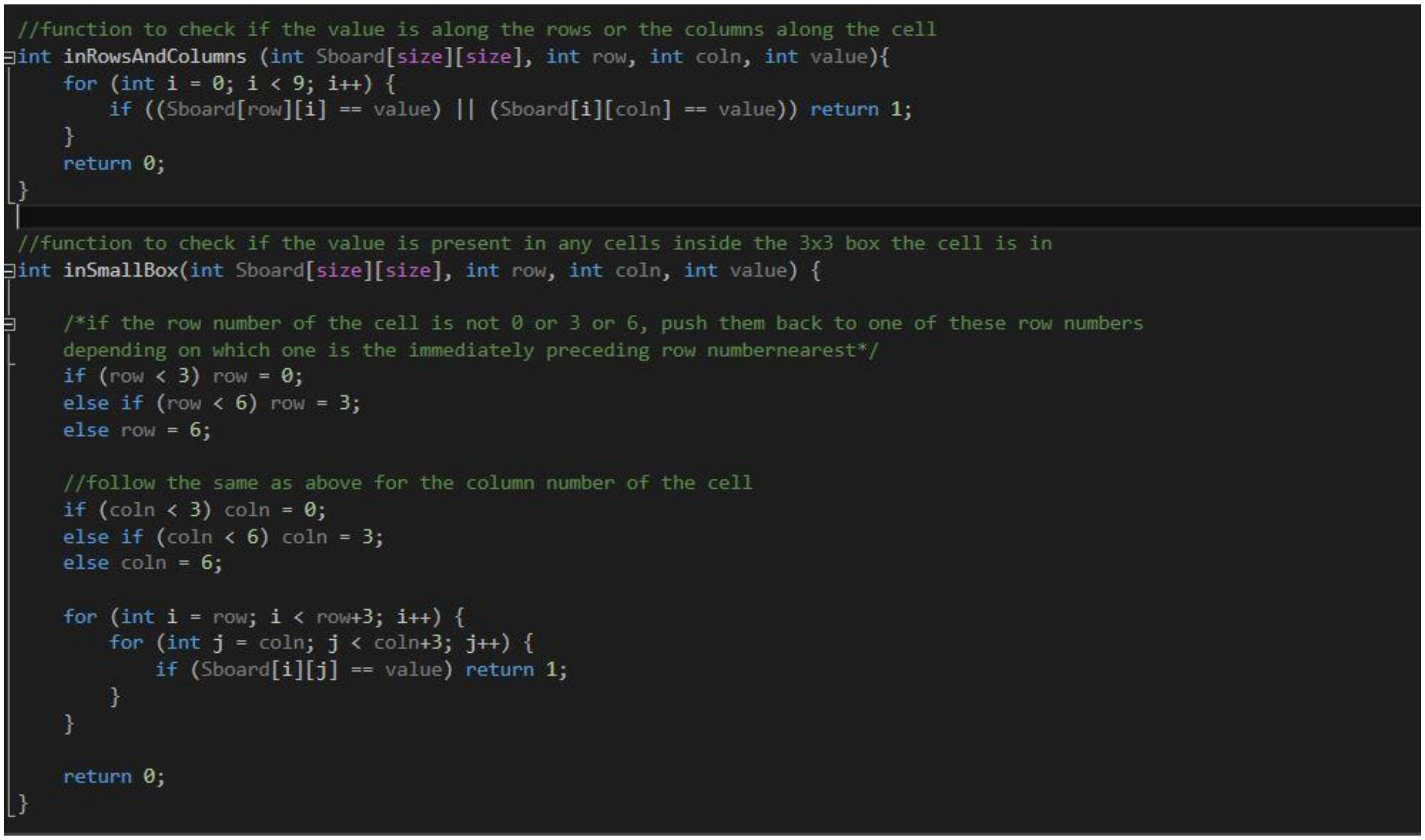
```c
//function that prints the board when the puzzle is solved
void displaySboard(int Sboard[size][size]) {
    printf("\n\n\nBravo, Solved!!\n\n\n\n");
    for (int row = 0; row < size; row++) {
        for (int coln = 0; coln < size; coln++) {
            printf("%4d", Sboard[row][coln]);
        }
        printf("\n\n");
    }
}


//function to find in there is any empty cell left in the board
int lookUpEmptyCells(int Sboard[size][size], int *row, int *coln) {
    for (*row = 0; *row < size; (*row)++) {
        for (*coln = 0; *coln < size; (*coln)++) {
            if (Sboard[*row][*coln] == 0) {
                return 1;
            }
        }
    }
    return 0;
}
```

attemptPuzzle() function



```c
//function that actually inserts the appropriate value into the puzzle one by one
int attemptPuzzle(int Sboard[size][size]) {

    int row = 0;
    int coln = 0;

    //in no empty cell, the puzzle is solved
    if (!lookUpEmptyCells(Sboard, &row, &coln)){
        return 1;
    }

    for (int value = 1; value <= size; value++ ) {

        //if the value is not along the rows or columns or inside the 3x3 box the cell is in, insert the value in the cell
        if (!inRowsAndColumns(Sboard, row, coln, value) && !inSmallBox(Sboard, row, coln, value)) {
            Sboard[row][coln] = value;

            //call the attemptPuzzle function recursively to fill the empty cells throughout the board
            if (attemptPuzzle(Sboard)) {
                return 1;
            }

            //if the value cannot be legally inserted into the cell, set it blank again and check for next value
            else Sboard[row][coln] = 0;
        }
    }

    return 0;
}
```

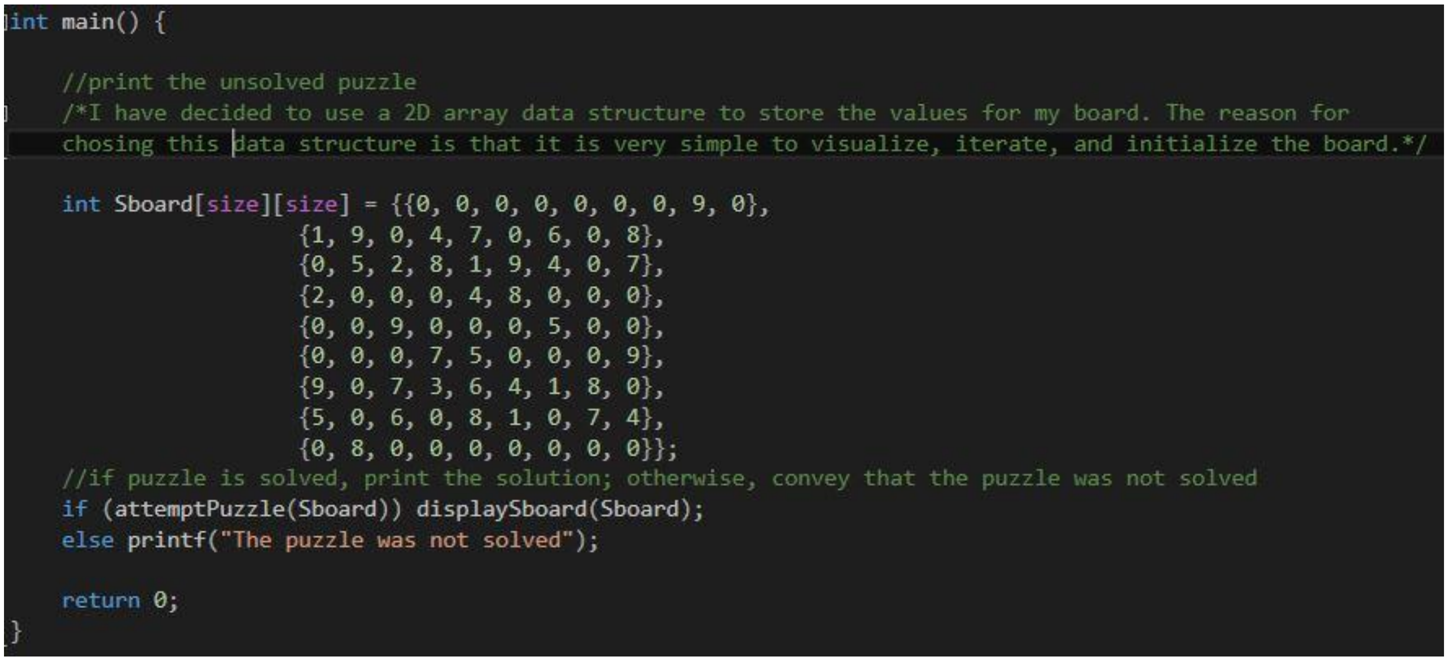
inRowsAndColumns() function and inSmallBox() function



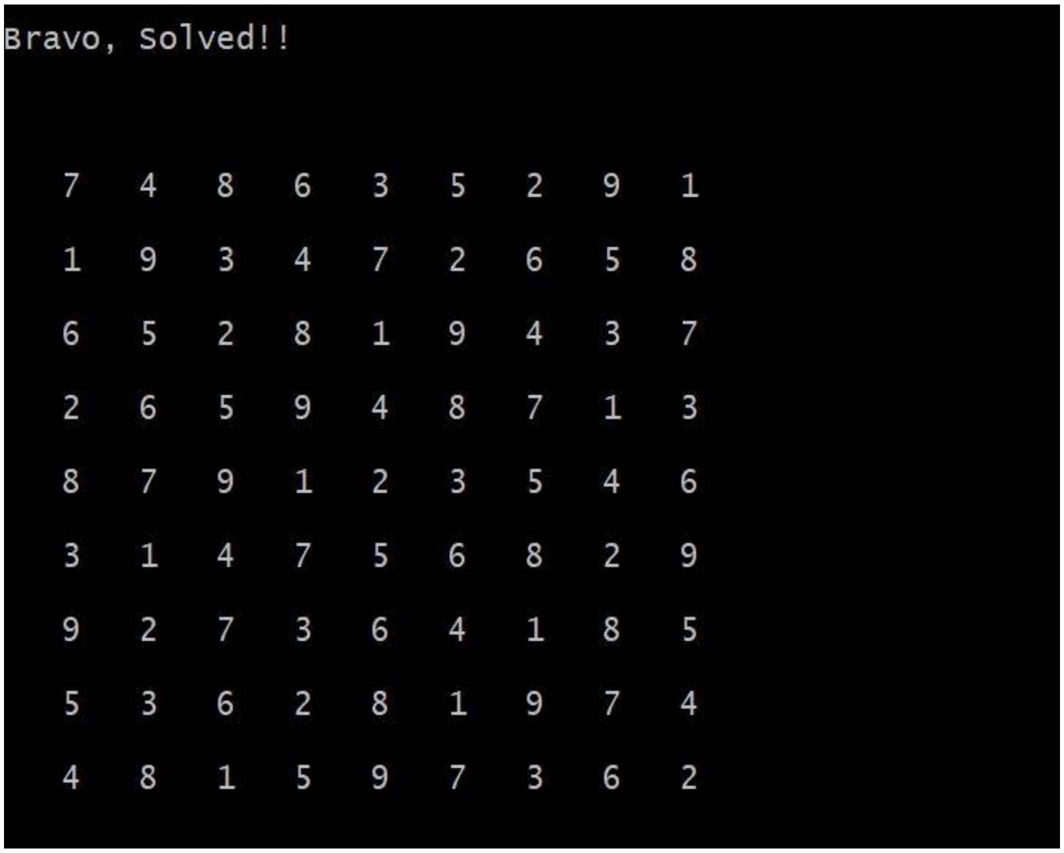
```
//function to check if the value is along the rows or the columns along the cell
int inRowsAndColumns (int Sboard[size][size], int row, int coln, int value){
    for (int i = 0; i < 9; i++) {
        if ((Sboard[row][i] == value) || (Sboard[i][coln] == value)) return 1;
    }
    return 0;
}

//function to check if the value is present in any cells inside the 3x3 box the cell is in
int inSmallBox(int Sboard[size][size], int row, int coln, int value) {

    /*if the row number of the cell is not 0 or 3 or 6, push them back to one of these row numbers
    depending on which one is the immediately preceding row numbernearest*/
    if (row < 3) row = 0;
    else if (row < 6) row = 3;
    else row = 6;

    //follow the same as above for the column number of the cell
    if (coln < 3) coln = 0;
    else if (coln < 6) coln = 3;
    else coln = 6;

    for (int i = row; i < row+3; i++) {
        for (int j = coln; j < coln+3; j++) {
            if (Sboard[i][j] == value) return 1;
        }
    }

    return 0;
}
```

5. Results:

First test case and its result:

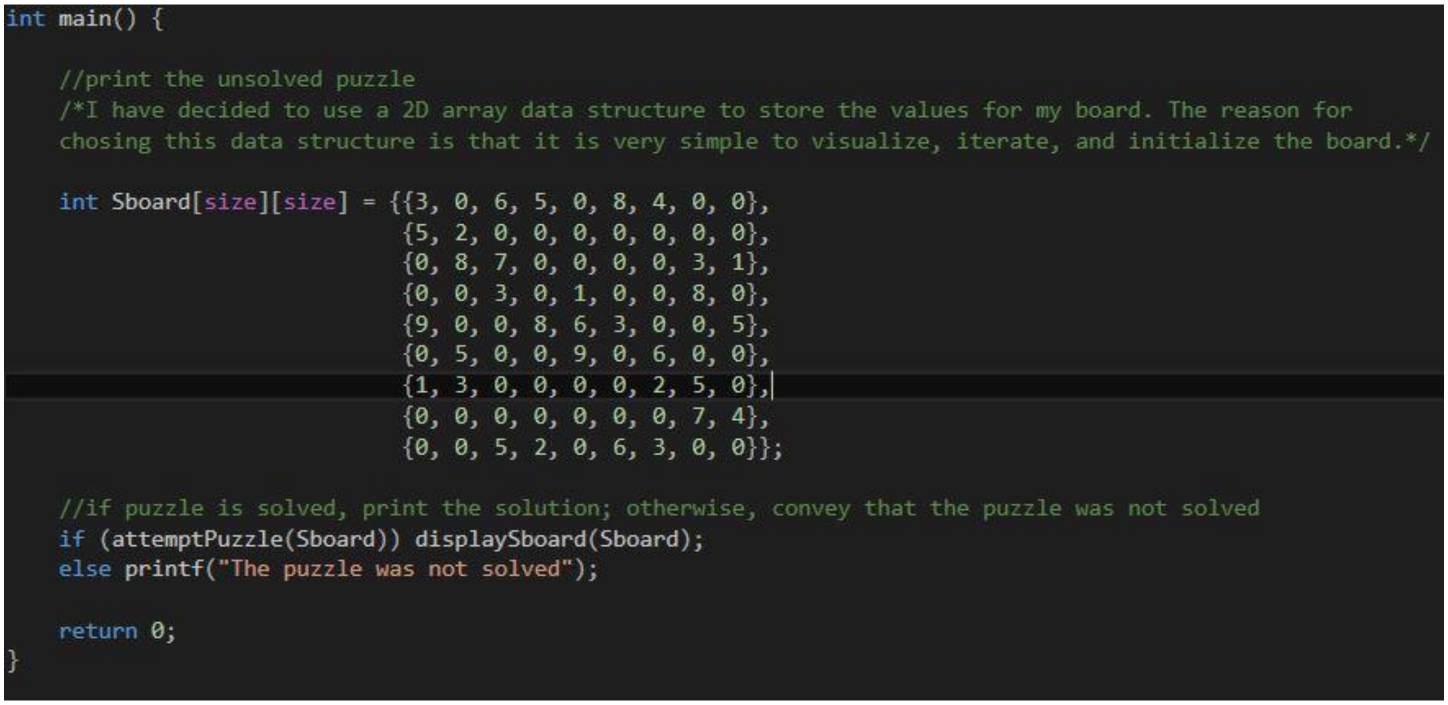


```
int main() {

    //print the unsolved puzzle
    /*I have decided to use a 2D array data structure to store the values for my board. The reason for
    chosing this data structure is that it is very simple to visualize, iterate, and initialize the board.*/

    int Sboard[size][size] = {{0, 0, 0, 0, 0, 0, 0, 9, 0},
                              {1, 9, 0, 4, 7, 0, 6, 0, 8},
                              {0, 5, 2, 8, 1, 9, 4, 0, 7},
                              {2, 0, 0, 0, 4, 8, 0, 0, 0},
                              {0, 0, 9, 0, 0, 0, 5, 0, 0},
                              {0, 0, 0, 7, 5, 0, 0, 0, 9},
                              {9, 0, 7, 3, 6, 4, 1, 8, 0},
                              {5, 0, 6, 0, 8, 1, 0, 7, 4},
                              {0, 8, 0, 0, 0, 0, 0, 0, 0}};
    //if puzzle is solved, print the solution; otherwise, convey that the puzzle was not solved
    if (attemptPuzzle(Sboard)) displaySboard(Sboard);
    else printf("The puzzle was not solved");

    return 0;
}
```

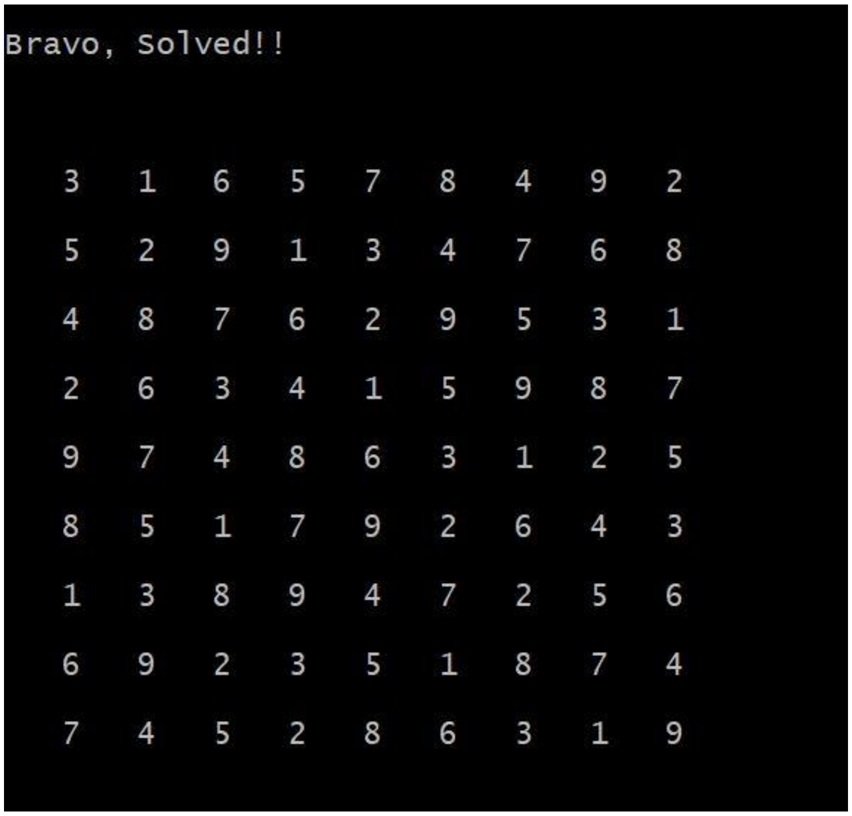


```
Bravo, Solved!!


   7   4   8   6   3   5   2   9   1

   1   9   3   4   7   2   6   5   8

   6   5   2   8   1   9   4   3   7

   2   6   5   9   4   8   7   1   3

   8   7   9   1   2   3   5   4   6

   3   1   4   7   5   6   8   2   9

   9   2   7   3   6   4   1   8   5

   5   3   6   2   8   1   9   7   4

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

Second test case and its result:

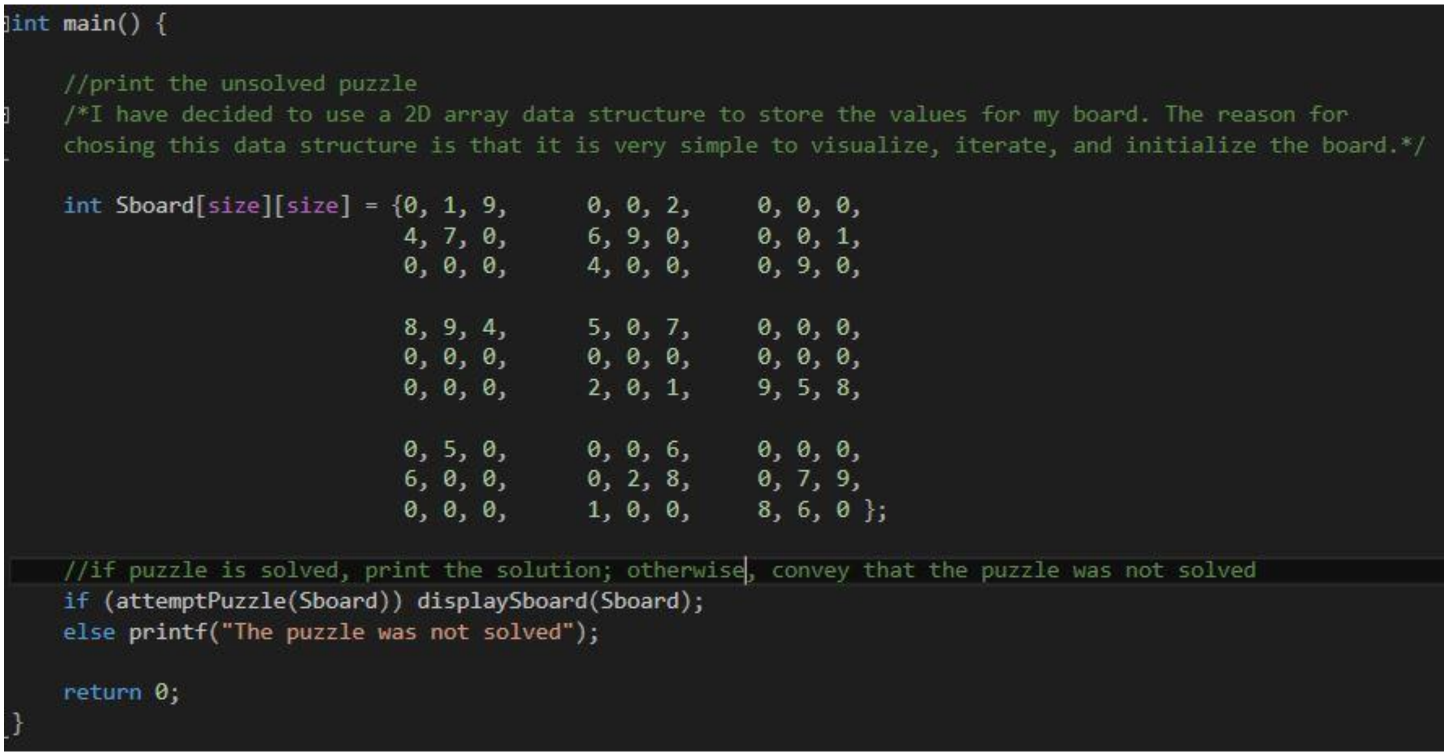


```
int main() {

    //print the unsolved puzzle
    /*I have decided to use a 2D array data structure to store the values for my board. The reason for
    chosing this data structure is that it is very simple to visualize, iterate, and initialize the board.*/

    int Sboard[size][size] = {{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 puzzle is solved, print the solution; otherwise, convey that the puzzle was not solved
    if (attemptPuzzle(Sboard)) displaySboard(Sboard);
    else printf("The puzzle was not solved");

    return 0;
}
```

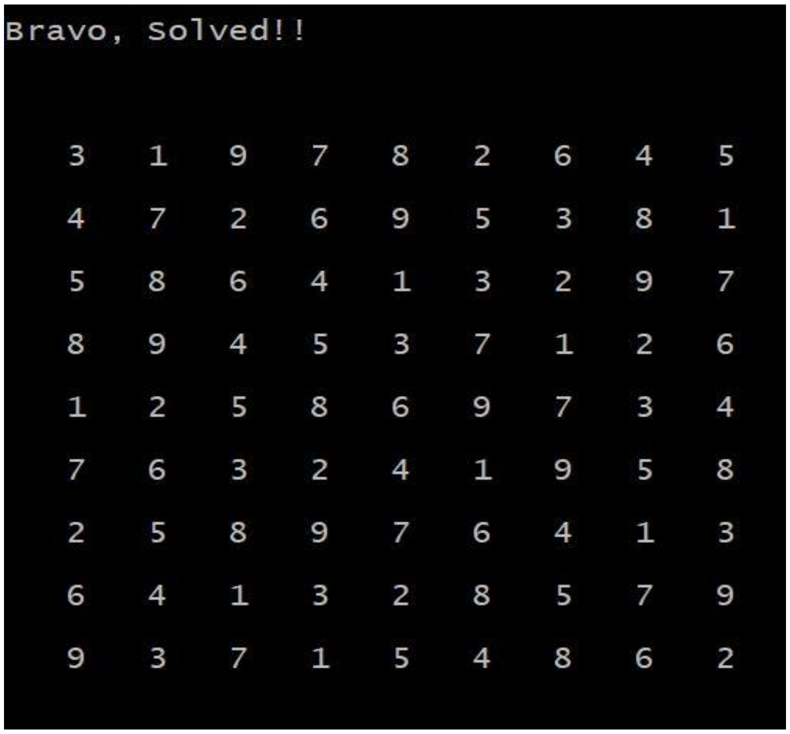


```
Bravo, Solved!!


   3   1   6   5   7   8   4   9   2

   5   2   9   1   3   4   7   6   8

   4   8   7   6   2   9   5   3   1

   2   6   3   4   1   5   9   8   7

   9   7   4   8   6   3   1   2   5

   8   5   1   7   9   2   6   4   3

   1   3   8   9   4   7   2   5   6

   6   9   2   3   5   1   8   7   4

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

Third test case and its result:



```
int main() {

    //print the unsolved puzzle
    /*I have decided to use a 2D array data structure to store the values for my board. The reason for
    chosing this data structure is that it is very simple to visualize, iterate, and initialize the board.*/

    int Sboard[size][size] = {0, 1, 9,      0, 0, 2,     0, 0, 0,
                              4, 7, 0,      6, 9, 0,     0, 0, 1,
                              0, 0, 0,      4, 0, 0,     0, 9, 0,

                              8, 9, 4,      5, 0, 7,     0, 0, 0,
                              0, 0, 0,      0, 0, 0,     0, 0, 0,
                              0, 0, 0,      2, 0, 1,     9, 5, 8,

                              0, 5, 0,      0, 0, 6,     0, 0, 0,
                              6, 0, 0,      0, 2, 8,     0, 7, 9,
                              0, 0, 0,      1, 0, 0,     8, 6, 0 };

    //if puzzle is solved, print the solution; otherwise, convey that the puzzle was not solved
    if (attemptPuzzle(Sboard)) displaySboard(Sboard);
    else printf("The puzzle was not solved");

    return 0;
}
```



```
Bravo, Solved!!


   3   1   9   7   8   2   6   4   5

   4   7   2   6   9   5   3   8   1

   5   8   6   4   1   3   2   9   7

   8   9   4   5   3   7   1   2   6

   1   2   5   8   6   9   7   3   4

   7   6   3   2   4   1   9   5   8

   2   5   8   9   7   6   4   1   3

   6   4   1   3   2   8   5   7   9

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

Thank you.