Sudoku Solver

Kanishk Bhat 20103233

Aayush Garg 20103212

Anish Jain 20103226

Yatin Garg 20103229

SUDOKU is a classic game of numbers. The commonly popular SUDOKU puzzle is a grid consisting of 9 sub-grids placed as a 3 x 3 matrix, wherein a sub-grid is a 3 x 3 matrix of cells, which may contain a number out of 1 through 9.

The puzzle is solved by placing a number, out of 1 through 9, in each of the empty cell under the following restrictions:

- ✓ 2 cells in a row do not contain the same number
- ✓ 2 cells in a column do not contain the same number
- ✓ 2 cells in a sub-grid do not contain the same number

A SUDOKU puzzle

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

Solution

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

SUDOKU Solver is a computer program that provides solution to a given SUDOKU puzzle.

A two-dimensional array stores the numbers as provided in the puzzle. The number 0 in a cell indicates that the cell is empty.

The solution to the puzzle is constructed by searching for the empty cells and replacing the number 0 by a number from 1 through 9 such that the criteria are fulfilled.

The array is traversed in row major order and cells are filled in the same order itself.

On placing a number in the empty cell, the next empty cell is searched for. Once, the last empty cell is filled successfully, the complete solution gets constructed.

In case, for a given cell, every number violates the set of conditions, then the cell filled previously is tried with another number, such that a valid entry is made in the given cell. In tracking back the previously filled cell, reaching the initial cell indicates that a valid solution to the given puzzle does not exist.

A recursive algorithm is designed to prepare the SUDOKU Solver.

Backtracking algorithm

Essentially, numbers are tried in empty spots until there aren't any that are possible, then different numbers are tried in the previous slots by backtracking.

This particular problem is very appropriate for backtracking, as a digit in the wrong location often quickly indicates that the solution is infeasible.

Let n be the count of entries which are left.

Step 1: If n = 0, i.e. there are no entries left, SUCCESS

Step 2: Otherwise, find a square that is not yet filled.

Step 3: For each digit from 1 to 9, place the digit in that square and see whether the solution is feasible, and if so call backtracking algorithm recursively, where

- if the algorithm indicates success, we are finished,
- Otherwise, try the next digit.

Step 4: if no digit works, there is no solution.

Code

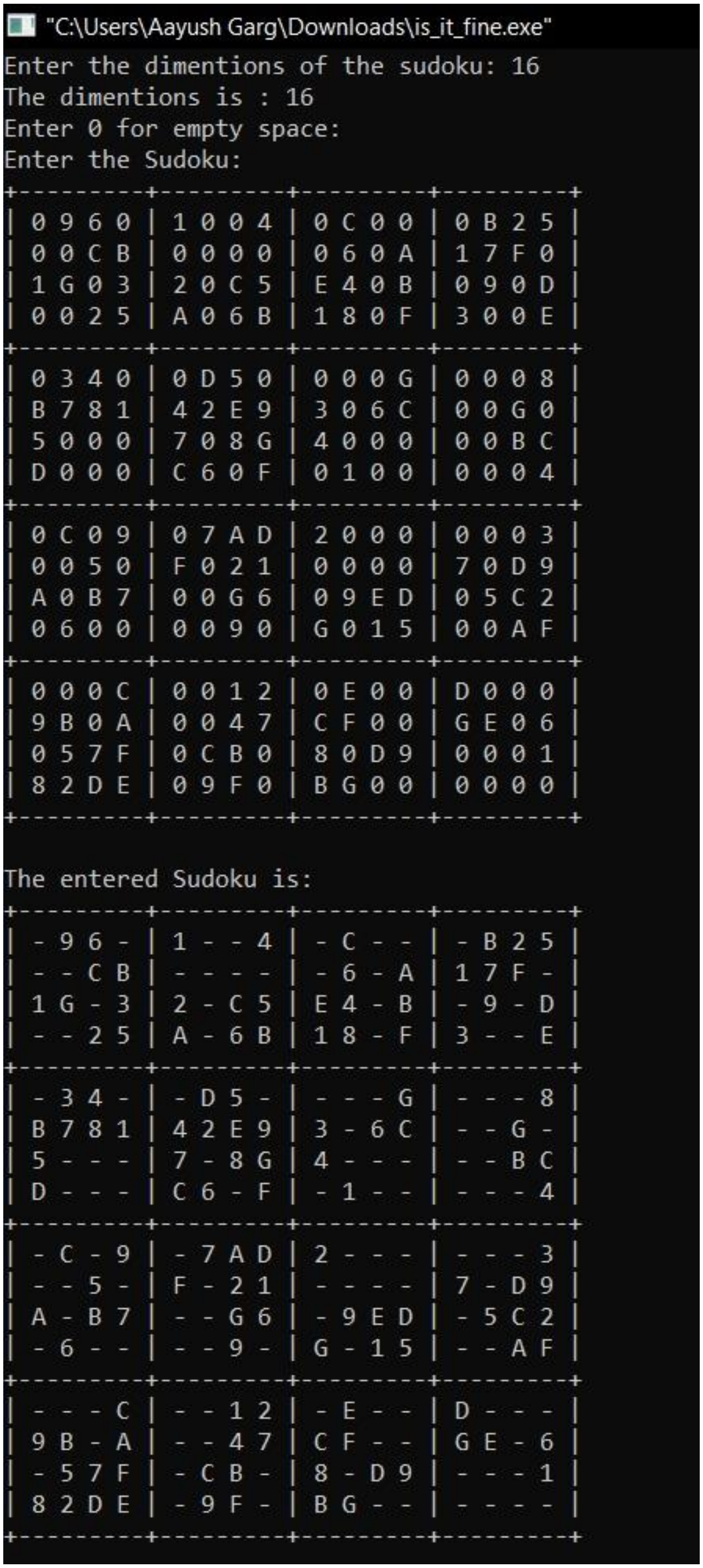
```
#include <iostream>
#include<cmath>
#include<conio.h>
using namespace std;

char dimension = 9, sub_dimension = sqrt(dimension);

void print_matrix(char matrix[][16], int dimension)
{
    for(char row = 0; row < dimension; ++row)
    {
        if(row % sub_dimension == 0 && dimension == 9)
            cout << "+-------+-------+-------+\n";
        else if(row % sub_dimension == 0 && dimension == 16)
            cout <<"+---------+---------+---------+---------+\n";
        for(char column = 0; column < dimension; ++column)
        {
            if(column % sub_dimension == 0)
                cout << "| ";
            if(matrix[row][column] == '0')
                cout << "- ";
            else
            {
                if(matrix[row][column] > '0' && matrix[row][column] <= '9')
                    cout << matrix[row][column] << " ";
                else
                {
                    cout <<(char)(matrix[row][column]+7) <<" ";
                }
            }
        }
```

```
            cout << "|\n";
        }
        if(dimension == 9)
            cout << "+-------+-------+-------+\n";
        else if(dimension == 16)
            cout <<"+---------+---------+---------+---------+\n";
    }

    bool place(char matrix[][16], char row, char column, char number)
    {
        for(char index = 0; index < dimension; index++)
            if(matrix[index][column] == number || matrix[row][index] == number)
                return false;

        char subx = (row / sub_dimension) * sub_dimension;
        char suby = (column / sub_dimension) * sub_dimension;

        for(char sub_row = 0; sub_row < sub_dimension; sub_row++)
            for(char sub_column = 0; sub_column < sub_dimension; sub_column++)
                if(matrix[sub_row + subx][sub_column + suby] == number)
                    return false;
        return true;
    }

    bool sudoku_solver(char matrix[][16], char row, char column)
    {
        if(matrix[row][column] != '0')
            if(column + 1 < dimension)
                return sudoku_solver(matrix, row, column + 1);
            else if(row + 1 < dimension)
                return sudoku_solver(matrix, row + 1, 0);
            else
                return true;
        else
        {
            for(char number = '1'; number < '1' + dimension; number++)
                if(place(matrix, row, column, number))
                {
                    matrix[row][column] = number;
                    if(column + 1 < dimension)
                    {
                        if(sudoku_solver(matrix, row, column + 1))
                            return true;
                    }
                    else if(row + 1 < dimension)
                    {
                        if(sudoku_solver(matrix, row + 1, 0))
                            return true;
                    }
                    else
                        return true;
                }
            matrix[row][column] = '0';
            return false;
        }
    }

    int main()
    {
        int temp;
        cout <<"Enter the dimentions of the sudoku: ";
        cin >>temp;
        dimension = (char)temp;
        sub_dimension = sqrt(dimension);

        cout <<"The dimentions is : " <<(int)dimension <<endl;
```

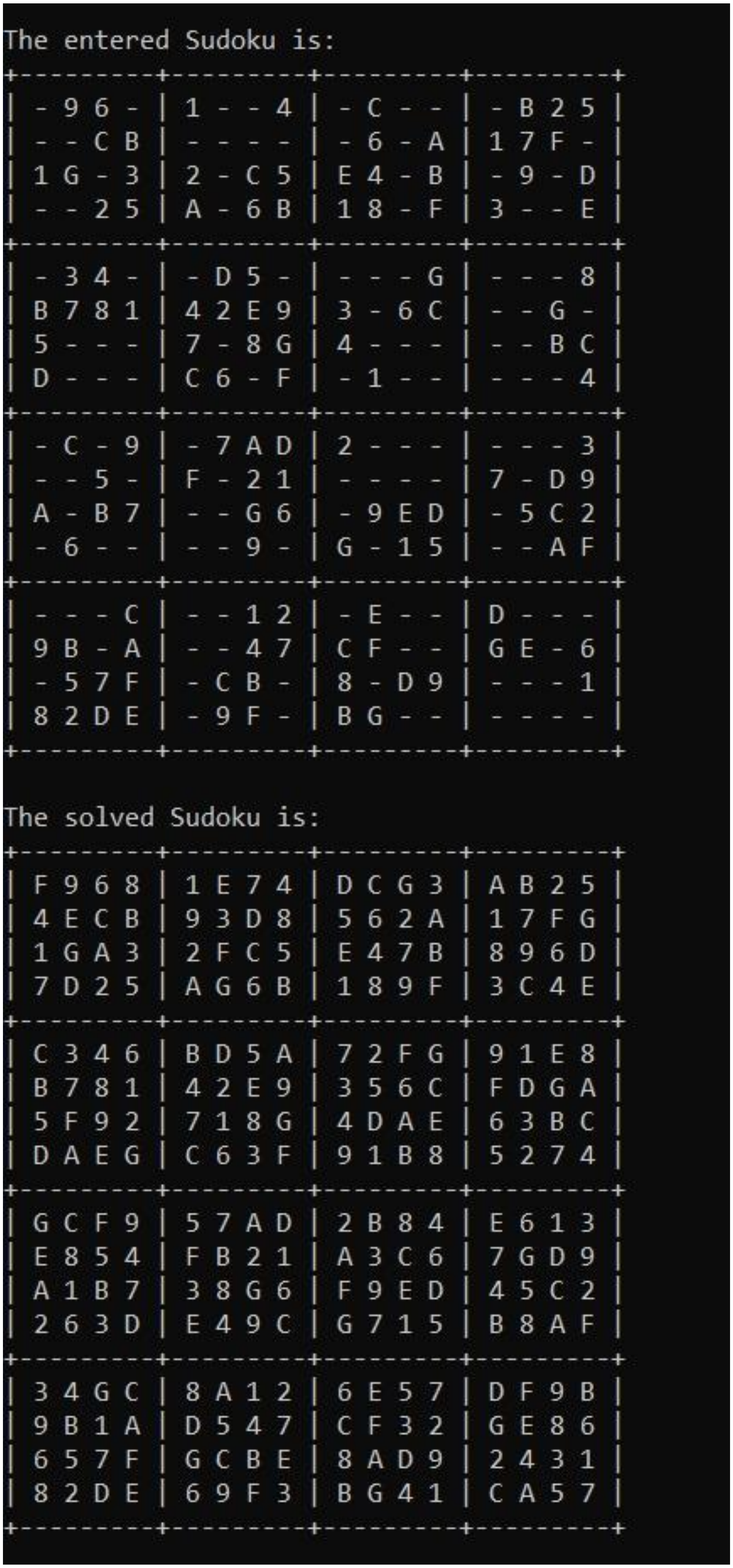
```
char matrix[16][16];
cout << "Enter 0 for empty space: " << endl;
cout << "Enter the Sudoku: \n";

for(char row = 0; row < dimension; ++row)
{
   if(row % sub_dimension == 0 && dimension == 9)
      cout << "+-------+-------+-------+\n";
   else if(row % sub_dimension == 0 && dimension == 16)
      cout <<"+---------+---------+---------+---------+\n";
   for(char column = 0, ch; column < dimension; ++column)
   {
      if(column % sub_dimension == 0)
         cout << "| ";
      ch = getche();
      if((ch >= '0' && ch <= '9') || (ch >= 'A' && ch <= 'G'))
      {
         if(ch >= 'A' && ch <= 'G')
            ch = ch - 7;
         matrix[row][column] = ch;
      }
      else
      {
         cout <<"Wrong Input !!!";
      }
      cout << " ";
   }
   cout << "|\n";
}
if(dimension == 9)
   cout << "+-------+-------+-------+\n";
else if(dimension == 16)
```

```
        cout <<"+---------+---------+---------+---------+\n";

    cout << "\nThe entered Sudoku is: \n";
    print_matrix(matrix, dimension);

    sudoku_solver(matrix, 0, 0);

    cout << "\nThe solved Sudoku is: \n";
    print_matrix(matrix, dimension);
    return 0;
}
```

Output



```
Enter the dimentions of the sudoku: 16
The dimentions is : 16
Enter 0 for empty space:
Enter the Sudoku:
+---------+---------+---------+---------+
| 0 9 6 0 | 1 0 0 4 | 0 C 0 0 | 0 B 2 5 |
| 0 0 C B | 0 0 0 0 | 0 6 0 A | 1 7 F 0 |
| 1 G 0 3 | 2 0 C 5 | E 4 0 B | 0 9 0 D |
| 0 0 2 5 | A 0 6 B | 1 8 0 F | 3 0 0 E |
+---------+---------+---------+---------+
| 0 3 4 0 | 0 D 5 0 | 0 0 0 G | 0 0 0 8 |
| B 7 8 1 | 4 2 E 9 | 3 0 6 C | 0 0 G 0 |
| 5 0 0 0 | 7 0 8 G | 4 0 0 0 | 0 0 B C |
| D 0 0 0 | C 6 0 F | 0 1 0 0 | 0 0 0 4 |
+---------+---------+---------+---------+
| 0 C 0 9 | 0 7 A D | 2 0 0 0 | 0 0 0 3 |
| 0 0 5 0 | F 0 2 1 | 0 0 0 0 | 7 0 D 9 |
| A 0 B 7 | 0 0 G 6 | 0 9 E D | 0 5 C 2 |
| 0 6 0 0 | 0 0 9 0 | G 0 1 5 | 0 0 A F |
+---------+---------+---------+---------+
| 0 0 0 C | 0 0 1 2 | 0 E 0 0 | D 0 0 0 |
| 9 B 0 A | 0 0 4 7 | C F 0 0 | G E 0 6 |
| 0 5 7 F | 0 C B 0 | 8 0 D 9 | 0 0 0 1 |
| 8 2 D E | 0 9 F 0 | B G 0 0 | 0 0 0 0 |
+---------+---------+---------+---------+


The entered Sudoku is:
+---------+---------+---------+---------+
| - 9 6 - | 1 - - 4 | - C - - | - B 2 5 |
| - - C B | - - - - | - 6 - A | 1 7 F - |
| 1 G - 3 | 2 - C 5 | E 4 - B | - 9 - D |
| - - 2 5 | A - 6 B | 1 8 - F | 3 - - E |
+---------+---------+---------+---------+
| - 3 4 - | - D 5 - | - - - G | - - - 8 |
| B 7 8 1 | 4 2 E 9 | 3 - 6 C | - - G - |
| 5 - - - | 7 - 8 G | 4 - - - | - - B C |
| D - - - | C 6 - F | - 1 - - | - - - 4 |
+---------+---------+---------+---------+
| - C - 9 | - 7 A D | 2 - - - | - - - 3 |
| - - 5 - | F - 2 1 | - - - - | 7 - D 9 |
| A - B 7 | - - G 6 | - 9 E D | - 5 C 2 |
| - 6 - - | - - 9 - | G - 1 5 | - - A F |
+---------+---------+---------+---------+
| - - - C | - - 1 2 | - E - - | D - - - |
| 9 B - A | - - 4 7 | C F - - | G E - 6 |
| - 5 7 F | - C B - | 8 - D 9 | - - - 1 |
| 8 2 D E | - 9 F - | B G - - | - - - - |
+---------+---------+---------+---------+
```



```
The entered Sudoku is:
+---------+---------+---------+---------+
| - 9 6 - | 1 - - 4 | - C - - | - B 2 5 |
| - - C B | - - - - | - 6 - A | 1 7 F - |
| 1 G - 3 | 2 - C 5 | E 4 - B | - 9 - D |
| - - 2 5 | A - 6 B | 1 8 - F | 3 - - E |
+---------+---------+---------+---------+
| - 3 4 - | - D 5 - | - - - G | - - - 8 |
| B 7 8 1 | 4 2 E 9 | 3 - 6 C | - - G - |
| 5 - - - | 7 - 8 G | 4 - - - | - - B C |
| D - - - | C 6 - F | - 1 - - | - - - 4 |
+---------+---------+---------+---------+
| - C - 9 | - 7 A D | 2 - - - | - - - 3 |
| - - 5 - | F - 2 1 | - - - - | 7 - D 9 |
| A - B 7 | - - G 6 | - 9 E D | - 5 C 2 |
| - 6 - - | - - 9 - | G - 1 5 | - - A F |
+---------+---------+---------+---------+
| - - - C | - - 1 2 | - E - - | D - - - |
| 9 B - A | - - 4 7 | C F - - | G E - 6 |
| - 5 7 F | - C B - | 8 - D 9 | - - - 1 |
| 8 2 D E | - 9 F - | B G - - | - - - - |
+---------+---------+---------+---------+

The solved Sudoku is:
+---------+---------+---------+---------+
| F 9 6 8 | 1 E 7 4 | D C G 3 | A B 2 5 |
| 4 E C B | 9 3 D 8 | 5 6 2 A | 1 7 F G |
| 1 G A 3 | 2 F C 5 | E 4 7 B | 8 9 6 D |
| 7 D 2 5 | A G 6 B | 1 8 9 F | 3 C 4 E |
+---------+---------+---------+---------+
| C 3 4 6 | B D 5 A | 7 2 F G | 9 1 E 8 |
| B 7 8 1 | 4 2 E 9 | 3 5 6 C | F D G A |
| 5 F 9 2 | 7 1 8 G | 4 D A E | 6 3 B C |
| D A E G | C 6 3 F | 9 1 B 8 | 5 2 7 4 |
+---------+---------+---------+---------+
| G C F 9 | 5 7 A D | 2 B 8 4 | E 6 1 3 |
| E 8 5 4 | F B 2 1 | A 3 C 6 | 7 G D 9 |
| A 1 B 7 | 3 8 G 6 | F 9 E D | 4 5 C 2 |
| 2 6 3 D | E 4 9 C | G 7 1 5 | B 8 A F |
+---------+---------+---------+---------+
| 3 4 G C | 8 A 1 2 | 6 E 5 7 | D F 9 B |
| 9 B 1 A | D 5 4 7 | C F 3 2 | G E 8 6 |
| 6 5 7 F | G C B E | 8 A D 9 | 2 4 3 1 |
| 8 2 D E | 6 9 F 3 | B G 4 1 | C A 5 7 |
+---------+---------+---------+---------+
```

References

https://www.programiz.com/dsa/backtracking-algorithm

https://www.javatpoint.com/backtracking-introduction

Prepared by

20103212	AAYUSH GARG
20103226	ANISH JAIN
20103233	KANISHK BHAT
20103229	YATIN GARG