SUDOKU SOLVER

Project submitted to the

SRM University – AP, Andhra Pradesh

for the partial fulfillment of the requirements to award the degree of

Bachelor of Technology/Master of Technology in

Computer Science and Engineering

School of Engineering and Sciences

Submitted by

Perumalla Dharan - AP21110010201

Tarun Teja Kudeti - AP21110010205

Pavan Sastry NVSS - AP21110010209

Vatala Phalgun - AP21110010223

Dinesh Grandhi - AP21110010240

Under the Guidance of

Dr. Manikandan V M

SRM University–AP

Neerukonda, Mangalagiri, Guntur

Andhra Pradesh – 522 240

December , 2022

Problem

We are given a partially filled 9*9 array grid.
Our goal is to assign digits from 1 to 9 in empty spaces such that every row, column, and subgrid of size 3×3 contains exactly one instance of the digits from 1 to 9.

Approach to solve the problem

- Firstly we will create a function `input_mat` which will take input of 9*9 matrix. (the empty spaces are considered as 0)
- Then we will check if the given matrix is valid or not by using the function `is_valid`
- If the given matrix is invalid then we will terminate the program
- Else we will continue and start solving the matrix in the function `solve`
- In the solve function we will check for the empty spaces in the matrix i.e. 0’s
- When we find an empty space in the matrix we will use the function `new_assign` to assign a number in that particular empty space.
- In the function `new_assign` we will check one by one if the number can be places or not by using the function `is_safe`
- In function `is_safe` we will check that row and column as well as the 3*3 box if the number can be placed or not by using the functions `check_row ,check_col ,check_box`
- We will continue till all the numbers are placed in the given matrix
- If there are some places in which any number cannot be placed then we will return the message that the particular matrix cannot be solved.
- Then we will print the output matrix.

Source Code

```cpp
#include <iostream>
using namespace std;

void input_mat(int a[9][9]);
bool is_valid(int a[9][9]);
void find_next(int *r, int *c);
bool check_row(int r, int c, int a[9][9]);
bool check_col(int r, int c, int a[9][9]);
bool check_box(int r, int c, int a[9][9]);
bool is_safe(int r, int c, int a[9][9]);
bool assign(int a[9][9], int r, int c);
bool new_assign(int a[9][9], int r, int c);
bool unassigned(int a[9][9]);
void solve(int a[9][9], int r, int c);
int calc_box(int r, int c, int a[9][9]);
void output_mat(int a[9][9]);

int main()
{
    int a[9][9];
    input_mat(a);
    if (!is_valid(a))
    {
        cout << "Invalid sudoku" << endl;
        exit(0);
    }
    solve(a, 0, 0);
    if (unassigned(a))
    {
        cout << "Sudoku cannot be solved" << endl;
        exit(0);
    }

    output_mat(a);
    return 0;
}
```

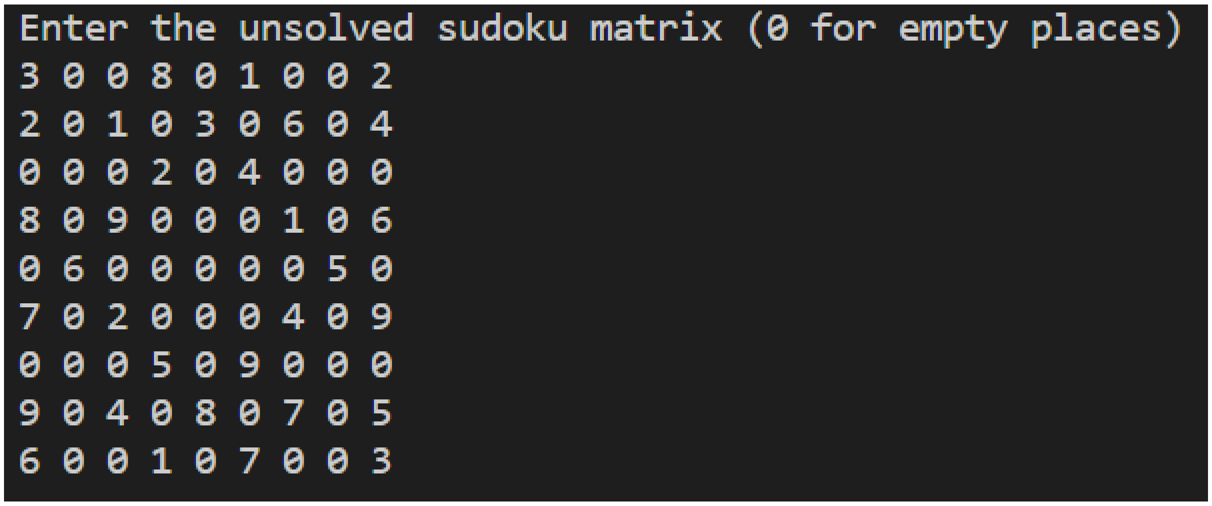
```
void find_next(int *r, int *c)
{
    (*c)++;
    if (*c > 8)
    {
        *c = 0;
        (*r)++;
        if (*r > 8)
        {
            *r = 9;
            *c = 9;
        }
    }
}

bool assign(int a[9][9], int r, int c)
{
    int i;
    for (i = 1; i <= 9; i++)
    {
        a[r][c] = i;
        if (is_safe(r, c, a))
        {
            cout << r << ',' << c << "=" << a[r][c] << endl;
            return true;
        }
    }
    a[r][c] = 0;
    return false;
}

bool new_assign(int a[9][9], int r, int c)
{
    int i;
    for (i = a[r][c] + 1; i <= 9; i++)
    {
        a[r][c] = i;
        if (is_safe(r, c, a))
        {
            // cout << r << ',' << c << "=" << a[r][c] << endl;
            return true;
        }
    }
    a[r][c] = 0;
    return false;
}

void solve(int a[9][9], int r, int c)
{
    if (r < 9 && c < 9)
    {
        if (a[r][c] == 0)
        {
        x:
            if (new_assign(a, r, c))
            {
                int new_r = r, new_c = c;
                find_next(&new_r, &new_c);
                solve(a, new_r, new_c); // 1
                if (unassigned(a))
                {
                    goto x;
                }
            }
            else
            {
                a[r][c] = 0;
                return;
            }
        }

        else
        {
            int new_r = r, new_c = c;
            find_next(&new_r, &new_c);
            solve(a, new_r, new_c); // 2
        }
    }
}
```

```
bool check_row(int r, int c, int a[9][9])
{
    for (int i = 0; i < 9; i++)
    {
        if (a[r][c] == a[r][i] && i != c)
            return false;
    }
    return true;
}

bool check_col(int r, int c, int a[9][9])
{
    for (int i = 0; i < 9; i++)
    {
        if (a[r][c] == a[i][c] && i != r)
            return false;
    }
    return true;
}

int calc_box(int r, int c, int a[9][9])
{
    int box;
    if (r >= 0 && r <= 2 && c >= 0 && c <= 2)
        box = 1;
    if (r >= 0 && r <= 2 && c >= 3 && c <= 5)
        box = 2;
    if (r >= 0 && r <= 2 && c >= 6 && c <= 8)
        box = 3;
    if (r >= 3 && r <= 5 && c >= 0 && c <= 2)
        box = 4;
    if (r >= 3 && r <= 5 && c >= 3 && c <= 5)
        box = 5;
    if (r >= 3 && r <= 5 && c >= 6 && c <= 8)
        box = 6;
    if (r >= 6 && r <= 8 && c >= 0 && c <= 2)
        box = 7;
    if (r >= 6 && r <= 8 && c >= 3 && c <= 5)
        box = 8;
    if (r >= 6 && r <= 8 && c >= 6 && c <= 8)
        box = 9;

    return box;
}

bool check_box(int r, int c, int a[9][9])
{
    int box = calc_box(r, c, a);
    int i, j;
    switch (box)
    {
    case 1:
        for (i = 0; i < 3; i++)
            for (j = 0; j < 3; j++)
            {
                if (a[r][c] == a[i][j] && r != i && c != j)
                    return false;
            }
        break;
    case 2:
        for (i = 0; i < 3; i++)
            for (j = 3; j < 6; j++)
            {
                if (a[r][c] == a[i][j] && r != i && c != j)
                    return false;
            }
        break;
    case 3:
        for (i = 0; i < 3; i++)
            for (j = 6; j < 9; j++)
            {
                if (a[r][c] == a[i][j] && r != i && c != j)
                    return false;
            }
        break;
    case 4:
        for (i = 3; i < 6; i++)
            for (j = 0; j < 3; j++)
            {
                if (a[r][c] == a[i][j] && r != i && c != j)
                    return false;
            }
        break;
    case 5:
        for (i = 3; i < 6; i++)
            for (j = 3; j < 6; j++)
            {
                if (a[r][c] == a[i][j] && r != i && c != j)
                    return false;
            }
        break;
    case 6:
        for (i = 3; i < 6; i++)
            for (j = 6; j < 9; j++)
            {
                if (a[r][c] == a[i][j] && r != i && c != j)
                    return false;
            }
        break;
    case 7:
        for (i = 6; i < 9; i++)
            for (j = 0; j < 3; j++)
            {
                if (a[r][c] == a[i][j] && r != i && c != j)
                    return false;
            }
        break;
    case 8:
        for (i = 6; i < 9; i++)
            for (j = 3; j < 6; j++)
            {
                if (a[r][c] == a[i][j] && r != i && c != j)
                    return false;
            }
        break;
    case 9:
        for (i = 6; i < 9; i++)
            for (j = 6; j < 9; j++)
            {
                if (a[r][c] == a[i][j] && r != i && c != j)
                    return false;
            }
        break;
    }
    return true;
}

bool is_safe(int r, int c, int a[9][9])
{
    if (check_row(r, c, a) && check_col(r, c, a) && check_box(r, c, a))
        return true;
    return false;
}

void input_mat(int a[9][9])
{
    cout << "Enter the unsolved sudoku matrix (0 for empty places)" <<
endl;
    int i, j;
    for (i = 0; i < 9; i++)
        for (j = 0; j < 9; j++)
            cin >> a[i][j];
}

void output_mat(int a[9][9])
{
    cout << "Resultant sudoku is:" << endl
         << endl;
    int i, j, m = 0, n = 0;
    for (i = 0; i < 9; i++)
    {
        m++;
        n = 0;
        for (j = 0; j < 9; j++)
        {
            n++;
            cout << a[i][j] << " ";
            if (n % 3 == 0)
                cout << "| ";
        }

        cout << endl;
        if (m % 3 == 0)
        {
            for (j = 0; j < 8; j++)
                cout << "-  ";
            cout << endl;
        }
    }
}

bool unassigned(int a[9][9])
{
    int i, j;
    for (i = 0; i < 9; i++)
        for (j = 0; j < 9; j++)
            if (a[i][j] == 0)
                return true;
    return false;
}

bool is_valid(int a[9][9])
{
    int i, j;
    for (i = 0; i < 9; i++)
    {
        int c[10] = {0};
        for (j = 0; j < 9; j++)
            c[a[i][j]]++;
        for (j = 1; j <= 9; j++)
            if (c[j] > 1)
            {
                cout << "Same row" << endl;
                return false;
            }
    }
    for (i = 0; i < 9; i++)
    {
        int c[10] = {0};
```

```cpp
        for (j = 0; j < 9; j++)
            c[a[j][i]]++;
        for (j = 1; j <= 9; j++)
            if (c[j] > 1)
            {
                cout << "Same column" << endl;
                return false;
            }
    }

    int c[10][10] = {0};
    for (i = 0; i < 9; i++)
        for (j = 0; j < 9; j++)
        {
            c[calc_box(i, j, a)][a[i][j]]++;
        }
    cout << endl;

    for (i = 1; i < 10; i++)
        for (j = 1; j < 10; j++)
            if (c[i][j] > 1)
            {
                cout << "Same box" << endl;
                return false;
            }

    return true;
}
```

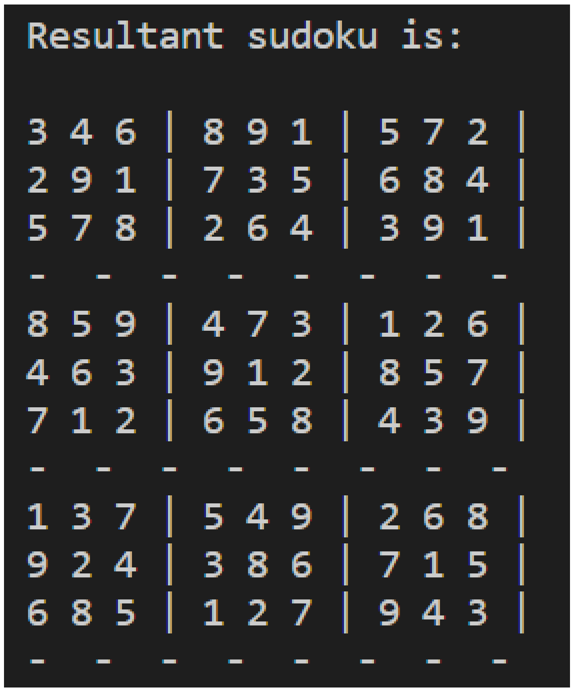
Sample Output

1. Input



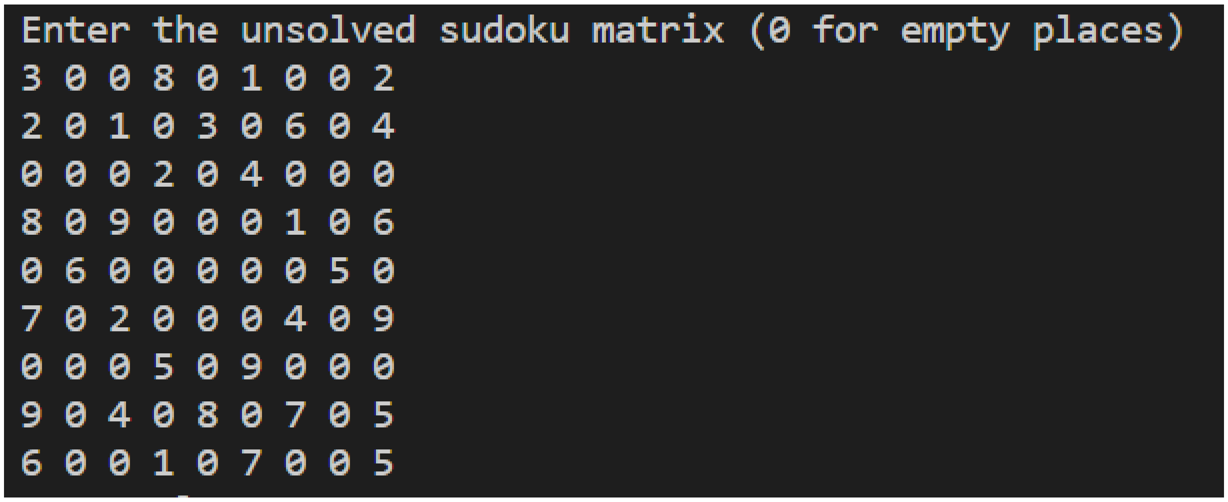
```
Enter the unsolved sudoku matrix (0 for empty places)
3 0 0 8 0 1 0 0 2
2 0 1 0 3 0 6 0 4
0 0 0 2 0 4 0 0 0
8 0 9 0 0 0 1 0 6
0 6 0 0 0 0 0 5 0
7 0 2 0 0 4 0 9
0 0 0 5 0 9 0 0 0
9 0 4 0 8 0 7 0 5
6 0 0 1 0 7 0 0 3
```

Output



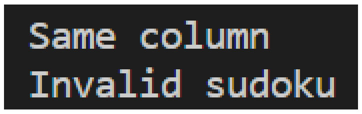
```
Resultant sudoku is:

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

2. Input



```
Enter the unsolved sudoku matrix (0 for empty places)
3 0 0 8 0 1 0 0 2
2 0 1 0 3 0 6 0 4
0 0 0 2 0 4 0 0 0
8 0 9 0 0 0 1 0 6
0 6 0 0 0 0 0 5 0
7 0 2 0 0 0 4 0 9
0 0 0 5 0 9 0 0 0
9 0 4 0 8 0 7 0 5
6 0 0 1 0 7 0 0 5
```

Output



```
Same column
Invalid sudoku
```