西安交通大学实验报告

课程名称：Python 数据处理　实验名称：数独矩阵

学　　院：计算机科学与技术学院　实　验　日　期　2023 年 11 月 3 日

班　　级：计算机 2201　姓　　名：戴早　学号：2224421652

诚信承诺：我保证本实验报告中的程序和本实验报告是我自己编写，没有抄袭。

邀请码：kFpix1

分工：独立完成

老师好！因为我是补修学生，不认识也无法找到我们组的同学，所以本次实验前期一直是我自己独立完成的，后来才联系上同组的一个同学，跟同学商量后决定我一个人交一份实验报告。所以本次实验的**代码和实验报告**全由**本人（戴早）一人完成**。望老师谅解我的情况！谢谢老师！

分组实验我自己的关联项目点进去变成了别人的项目，如果看不到我的项目，麻烦老师用邀请码查看我的项目。谢谢老师！

一、实验任务

使用 C 语言实现以下功能：

1) 9×9 数组格式化输出；
2) 随机生成一个 9×9 的不完整的矩阵；
3) 判断一个不完整的矩阵是否满足“数独矩阵”的定义；
4) 对一个不完整的“数独矩阵”进行判断，如果满足“数独矩阵”则补充完整，使之成为

5) 一个完整的“数独矩阵”。

二、实验环境

1) 硬件：华为泰山服务器
2) 软件：华为 Openeuler 操作系统、gcc

三、实验内容与结果

（写出题目描述，源程序和运行结果的截图）

1)

【题目描述】

1. 9×9 数组格式化输出：

要求：

- 要求把数组按照“数独”的格式输出；
- 格式不固定，只要看起来像是数独就行。

思路：用 for 循环循环数组中的每一个数，并在每一次循环时加以判断，确定|、-、空格、换行出现的位置。

【源程序】

```
#include <stdio.h>
int board[9][9]={{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}};
void f1()
{
    printf("|");
    for (int i = 0; i < 23; i++)
    {
        printf("-");
    }
    printf("|\n");
}
```

```c
int main()
{
    printf("格式化后的数组：\n");
    for (int i = 0; i < 9; i++)
    {
        if (i % 3 == 0)
        {
            f1();
        }
        printf("| ");
        for (int j = 0; j < 9; j++)
        {
            if (j % 3 == 0 && j)
            {
                printf("| ");
            }
            printf("%d ", board[i][j]);
        }
        printf("|\n");
    }
    f1();
}
```

【运行结果】

```
IP address:     172.17.0.1
Users online:   4
To run a command as administrator(user "root"),use "sudo <command>"
.
[s2224421652@localhost ~]$ cd "/home/s2224421652/sudoku_matrix"
[s2224421652@localhost sudoku_matrix]$ gcc t1.c -o t1.out
[s2224421652@localhost sudoku_matrix]$ ./t1.out
格式化后的数组：
|-----------------------|
| 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 |
|-----------------------|
[s2224421652@localhost sudoku_matrix]$
```

2)

【题目描述】

2. 随机生成一个 9×9 的不完整的矩阵：

要求：

- 数组内任意一个元素 x 范围为：1<=x<=9；
- 数组每行仅有三个互不相同的数字 x（x∈[1,9]），其他位置为数字 0 或者字符‘.’（取决于之前选择了整型数组还是字符数组）；
- 数组中 1-3 行中包含了 1-9 中所有的 9 个数字，4-6 行和 7-9 行也分别是如此；
- 使用之前实现的格式化方法输出该数组。

思路：对每连续三行，随机行坐标和列坐标，在确保每行至多三个数的基础上将 1-9 依次填入。

【源程序】

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int board[9][9] = {0}, f[9] = {0};
void f1()
{
    printf("|");
    for (int i = 0; i < 23; i++)
    {
        printf("-");
    }
    printf("|\n");
}
void operate()
{
    for (int i = 0; i < 9; i++)
    {
        if (i % 3 == 0)
        {
            f1();
        }
        printf("| ");
        for (int j = 0; j < 9; j++)
        {
            if (j % 3 == 0 && j)
```

```
            {
                printf("| ");
            }
            printf("%d ", board[i][j]);
        }
        printf("|\n");
    }
    f1();
}
void Random(int n)
{
    int k = 9;
    while (k)
    {
        int i = rand() % 3, j =rand() % 9;
        i += (n - 1) * 3;
        if (f[i] == 0)
        {
            if (board[i][j] == 0)
            {
                board[i][j] = k;
                k--;
            }
        }
        int cnt = 0;
        for (int m = 0; m < 9; m++)
        {
            if (board[i][m])
            {
                cnt++;
            }
        }
        if (cnt == 3)
        {
            f[i] = 1;
        }
    }
}
int main()
{
    printf("随机生成一个 9×9 的不完整的矩阵：\n");
    srand(time(NULL));
    Random(1);
    Random(2);
```

```
    Random(3);
    operate();
}
```

【运行结果】

```
[s2224421652@localhost sudoku_matrix]$ gcc t2.c -o t2.out
[s2224421652@localhost sudoku_matrix]$ ./t2.out
随机生成一个9×9的不完整的矩阵:
|----------------------|
| 0 0 1 | 0 0 0 | 5 0 4 |
| 7 0 6 | 0 2 0 | 0 0 0 |
| 0 0 3 | 8 0 9 | 0 0 0 |
|----------------------|
| 0 1 0 | 6 0 0 | 0 4 0 |
| 0 0 0 | 9 8 0 | 0 7 0 |
| 2 0 5 | 0 0 0 | 3 0 0 |
|----------------------|
| 6 0 0 | 0 3 0 | 0 5 0 |
| 9 0 0 | 0 4 0 | 7 0 0 |
| 0 0 1 | 8 0 0 | 0 2 0 |
|----------------------|
[s2224421652@localhost sudoku_matrix]$
```

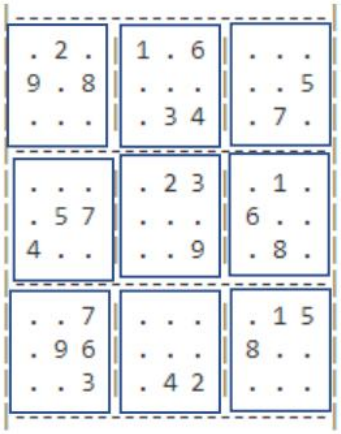
3)

【题目描述】

3. 判断一个不完整的矩阵是否满足“数独矩阵”的定义：

“数独矩阵”的定义：

- 数字 1-9 在每一行最多只能出现一次，不完整时可能是 0 或 1 次；
- 数字 1-9 在每一列最多只能出现一次，不完整时同上；
- 数字 1-9 在每一个被分隔开的 3x3 的矩阵内最多只能出现一次，不完整时同上；
- 见下图，被分隔开的 3x3 的矩阵指每个方框框起来的部分。



思路：循环每行、每列、每个 3*3 矩阵，计算 1-9 每个数出现的次数

【源程序】

```
#include <stdio.h>
int f[9] = {0};
int t1[9][9] = {0}, t2[9][9] = {0}, t3[9][9] = {0};
int board[9][9] = {{5, 3, 0, 0, 7, 0, 0, 0, 0},
{6, 0, 0, 1, 9, 5, 0, 0, 0},
{0, 9, 8, 0, 0, 0, 0, 6, 0},
{8, 0, 0, 0, 6, 0, 0, 0, 3},
{4, 0, 0, 8, 0, 3, 0, 0, 1},
{7, 0, 0, 0, 2, 0, 0, 0, 6},
{0, 6, 0, 0, 0, 0, 2, 8, 0},
{0, 0, 0, 4, 1, 9, 0, 0, 5},
{0, 0, 0, 0, 8, 0, 0, 7, 9}};
void f1()
{
    printf("|");
    for (int i = 0; i < 23; i++)
    {
        printf("-");
    }
    printf("|\n");
}
void operate()
{
    for (int i = 0; i < 9; i++)
    {
        if (i % 3 == 0)
        {
            f1();
        }
        printf("| ");
        for (int j = 0; j < 9; j++)
        {
            if (j % 3 == 0 && j)
            {
                printf("| ");
            }
            printf("%d ", board[i][j]);
        }
        printf("|\n");
    }
    f1();
```

```
}
void judge()
{
    for (int i = 0; i < 9; i++)
    {
        for(int j = 0; j < 9; j++)
        {
            if (board[i][j])
            {
                t1[i][board[i][j] - 1]++;
                t2[j][board[i][j] - 1]++;
                t3[(i / 3) * 3 + j / 3][board[i][j] - 1]++;
            }
        }
    }
    for (int i = 0; i < 9; i++)
    {
        for (int num = 0; num < 9; num++)
        {
            if (t1[i][num] > 1)
            {
                printf("False:Invalid initial Sudoku matrix!\n");
                printf("      The number %d in the row %d has been used!", num + 1, i + 1);
                return;
            }
            else if (t2[i][num] > 1)
            {
                printf("False:Invalid initial Sudoku matrix!\n");
                printf("      The number %d in the col %d has been used!", num + 1, i + 1);
                return;
            }
            else if (t3[i][num] > 1)
            {
                printf("False:Invalid initial Sudoku matrix!\n");
                printf("      The number %d in the block %d has been used!", num + 1, i + 1);
                return;
            }
        }
    }
    printf("True:Valid initial Sudoku matrix!");
}
int main()
{
    printf("The original Sudoku matrix:\n");
```

```
    operate();
    judge();
}
// int board[9][9] = {{8, 3, 0, 0, 7, 0, 0, 0, 0},
// {6, 0, 0, 1, 9, 5, 0, 0, 0},
// {0, 9, 8, 0, 0, 0, 0, 6, 0},
// {8, 0, 0, 0, 6, 0, 0, 0, 3},
// {4, 0, 0, 8, 0, 3, 0, 0, 1},
// {7, 0, 0, 0, 2, 0, 0, 0, 6},
// {0, 6, 0, 0, 0, 0, 2, 8, 0},
// {0, 0, 0, 4, 1, 9, 0, 0, 5},
// {0, 0, 0, 0, 8, 0, 0, 7, 9}};

// int board[9][9] = {{5, 3, 0, 0, 7, 0, 0, 0, 0},
// {6, 0, 0, 1, 9, 5, 0, 0, 0},
// {0, 9, 8, 0, 0, 0, 0, 6, 0},
// {8, 0, 0, 0, 6, 0, 0, 0, 3},
// {4, 0, 0, 8, 0, 3, 0, 0, 1},
// {7, 0, 0, 0, 2, 0, 0, 0, 6},
// {0, 6, 0, 0, 0, 0, 2, 8, 0},
// {0, 0, 0, 4, 1, 9, 0, 0, 5},
// {0, 0, 0, 0, 8, 0, 0, 7, 9}};
```

【运行结果】

board0：

```
| 9 0 0 | 0 4 0 | 7 0 0 |
| 0 0 1 | 8 0 0 | 0 2 0 |
|-----------------------|
[s2224421652@localhost sudoku_matrix]$ gcc t3board0.c -o t3board0.o
ut
[s2224421652@localhost sudoku_matrix]$ ./t3board0.out
The original Sudoku matrix:
|-----------------------|
| 5 3 0 | 0 7 0 | 0 0 0 |
| 6 0 0 | 1 9 5 | 0 0 0 |
| 0 9 8 | 0 0 0 | 0 6 0 |
|-----------------------|
| 8 0 0 | 0 6 0 | 0 0 3 |
| 4 0 0 | 8 0 3 | 0 0 1 |
| 7 0 0 | 0 2 0 | 0 0 6 |
|-----------------------|
| 0 6 0 | 0 0 0 | 2 8 0 |
| 0 0 0 | 4 1 9 | 0 0 5 |
| 0 0 0 | 0 8 0 | 0 7 9 |
|-----------------------|
True:Valid initial Sudoku matrix![s2224421652@localhost sudoku_matr
ix]$
```

board1：

```
[s2224421652@localhost sudoku_matrix]$ ./t3board1.out
The original Sudoku matrix:
|----------------------|
| 8 3 0 | 0 7 0 | 0 0 0 |
| 6 0 0 | 1 9 5 | 0 0 0 |
| 0 9 8 | 0 0 0 | 0 6 0 |
|----------------------|
| 8 0 0 | 0 6 0 | 0 0 3 |
| 4 0 0 | 8 0 3 | 0 0 1 |
| 7 0 0 | 0 2 0 | 0 0 6 |
|----------------------|
| 0 6 0 | 0 0 0 | 2 8 0 |
| 0 0 0 | 4 1 9 | 0 0 5 |
| 0 0 0 | 0 8 0 | 0 7 9 |
|----------------------|
False:Invalid initial Sudoku matrix!
      The number 8 in the col 1 has been used![s2224421652@localhos
t sudoku_matrix]$
```

4)

【题目描述】

4. 对一个不完整的“数独矩阵”进行判断，如果满足“数独矩阵”则补充完整，使之成为一个完整的“数独矩阵”：

要求：

1) “数独矩阵”的定义同第三个部分的定义；
2) 完整的“数独矩阵”一定也满足“数独矩阵”的定义；
3) 程序开始时，输出“The original Sudoku matrix: ”；
4) 然后，使用之前实现的格式化输出的方法，输出需要判断的数组；
5) 如果满足定义的话，输出“True:Valid initial Sudoku matrix!”：
   - ✧ 如果求解后发现有解，则再输出“The solution of Sudoku matrix:”，再输出补充完整的数独矩阵；
   - ✧ 如果经过求解后发现没有解的话输出“No solution!”。
6) 如果不满足定义的话，
   - >1 先输出“False:Invalid initial Sudoku matrix!”；
   - >2 再输出不满足的原因，具体要求同第三部分的要求 4)；
   - >3 最后输出“No solution!”。

思路：核心算法是 dfs（递归+回溯）。首先判断初始矩阵是否满足数独定义（同上一问）。在初始矩阵满足数独定义的情况下，从左上到右下依次尝试每个空可能填入的数字，并在填完后加以判断，若当前矩阵满足数独定义，跳到下一空，若当前矩阵不满足数独定义，尝试下一个数，若尝试完所有数字后都没能

找到满足数独定义的填法，返回上一空。注意已经存在的数字不能更改。若枚举完所有情况都没有到终止条件，说明无解；若到达终止条件，有解并输出。终止条件为跳到第 81 个空之后。

【源程序】

```
#include <stdio.h>
int f[9] = {0}, copy[9][9] = {0};
int t1[9][9] = {0}, t2[9][9] = {0}, t3[9][9] = {0};
int board[9][9] = {{8, 3, 0, 0, 7, 0, 0, 0, 0},
{6, 0, 0, 1, 9, 5, 0, 0, 0},
{0, 9, 8, 0, 0, 0, 0, 6, 0},
{8, 0, 0, 0, 6, 0, 0, 0, 3},
{4, 0, 0, 8, 0, 3, 0, 0, 1},
{7, 0, 0, 0, 2, 0, 0, 0, 6},
{0, 6, 0, 0, 0, 0, 2, 8, 0},
{0, 0, 0, 4, 1, 9, 0, 0, 5},
{0, 0, 0, 0, 8, 0, 0, 7, 9}};
int ifans = 0;
void f1()
{
    printf("|");
    for (int i = 0; i < 23; i++)
    {
        printf("-");
    }
    printf("|\n");
}
void operate()
{
    for (int i = 0; i < 9; i++)
    {
        if (i % 3 == 0)
        {
            f1();
        }
        printf("| ");
        for (int j = 0; j < 9; j++)
        {
            if (j % 3 == 0 && j)
            {
                printf("| ");
            }
            printf("%d ", board[i][j]);
```

```
        }
        printf("|\n");
    }
    f1();
}
int judge()
{
    for (int i = 0; i < 9; i++)
    {
        for(int j = 0; j < 9; j++)
        {
            if (board[i][j])
            {
                t1[i][board[i][j] - 1]++;
                t2[j][board[i][j] - 1]++;
                t3[(i / 3) * 3 + j / 3][board[i][j] - 1]++;
            }
        }
    }
    for (int i = 0; i < 9; i++)
    {
        for (int num = 0; num < 9; num++)
        {
            if (t1[i][num] > 1)
            {
                printf("False:Invalid initial Sudoku matrix!\n");
                printf("      The number %d in the row %d has been used!\n", num + 1, i + 1);
                printf("No solution!");
                return 0;
            }
            else if (t2[i][num] > 1)
            {
                printf("False:Invalid initial Sudoku matrix!\n");
                printf("      The number %d in the col %d has been used!\n", num + 1, i + 1);
                printf("No solution!");
                return 0;
            }
            else if (t3[i][num] > 1)
            {
                printf("False:Invalid initial Sudoku matrix!\n");
                printf("      The number %d in the block %d has been used!\n", num + 1, i + 1);
                printf("No solution!");
                return 0;
            }
```

```
        }
    }
    printf("True:Valid initial Sudoku matrix!\n");
    return 1;
}
void fill(int x, int y)
{
    int num;
    if (x == 9 && y == 0)
    {
        printf("The solution of Sudoku matrix:\n");
        operate();
        ifans = 1;
    }
    else
    {
        if (copy[x][y])
        {
            if (y == 8)
            {
                fill(x + 1, 0);
                if (x == 0 && y == 0 && ifans == 0)
                {
                    printf("No solution!");
                }
                return;
            }
            else{
                fill(x, y + 1);
                if (x == 0 && y == 0 && ifans == 0)
                {
                    printf("No solution!");
                }
                return;
            }
        }
        else
        {
            for(num = 1; num <= 9; num++)
            {
                if (t1[x][num - 1] == 0 && t2[y][num - 1] == 0 && t3[(x / 3) * 3 + y / 3][num - 1] == 0)
                {
                    board[x][y] = num;
                    t1[x][num - 1] = 1;
```

```
                    t2[y][num - 1] = 1;
                    t3[(x / 3) * 3 + y / 3][num - 1] = 1;
                    if (y == 8)
                    {
                        // operate();
                        fill(x + 1, 0);
                        board[x][y] = 0;
                        t1[x][num - 1] = 0;
                        t2[y][num - 1] = 0;
                        t3[(x / 3) * 3 + y / 3][num - 1] = 0;
                    }
                    else{
                        // operate();
                        fill(x, y + 1);
                        board[x][y] = 0;
                        t1[x][num - 1] = 0;
                        t2[y][num - 1] = 0;
                        t3[(x / 3) * 3 + y / 3][num - 1] = 0;
                    }
                }
            }
            if (x == 0 && y == 0 && ifans == 0)
            {
                printf("No solution!");
            }
            return;
        }
    }
}
int main()
{
    printf("The original Sudoku matrix:\n");
    operate();
    for (int i = 0; i < 9; i++)
    {
        for (int j = 0; j < 9; j++)
        {
            copy[i][j] = board[i][j];
        }
    }
    if(judge())
    {
        fill(0,0);
    }
```

```
}
// int board[9][9] = {{8, 3, 0, 0, 7, 0, 0, 0, 0},
// {6, 0, 0, 1, 9, 5, 0, 0, 0},
// {0, 9, 8, 0, 0, 0, 0, 6, 0},
// {8, 0, 0, 0, 6, 0, 0, 0, 3},
// {4, 0, 0, 8, 0, 3, 0, 0, 1},
// {7, 0, 0, 0, 2, 0, 0, 0, 6},
// {0, 6, 0, 0, 0, 0, 2, 8, 0},
// {0, 0, 0, 4, 1, 9, 0, 0, 5},
// {0, 0, 0, 0, 8, 0, 0, 7, 9}};

// int board[9][9] = {{5, 2, 0, 0, 7, 0, 0, 0, 0},
// {6, 0, 0, 1, 9, 5, 0, 0, 0},
// {0, 9, 8, 0, 0, 0, 0, 6, 0},
// {8, 0, 0, 0, 6, 0, 0, 0, 3},
// {4, 0, 0, 8, 0, 3, 0, 0, 1},
// {7, 0, 0, 0, 2, 0, 0, 0, 6},
// {0, 6, 0, 0, 0, 0, 2, 8, 0},
// {0, 0, 0, 4, 1, 9, 0, 0, 5},
// {0, 0, 0, 0, 8, 0, 0, 7, 9}};
```

【运行结果】

将与随机生成 9×9 的不完整的矩阵相关的代码注释后，并令 board 数组分别等于 board0 数组、board1 数组、board2 数组时，得到的运行结果如下：

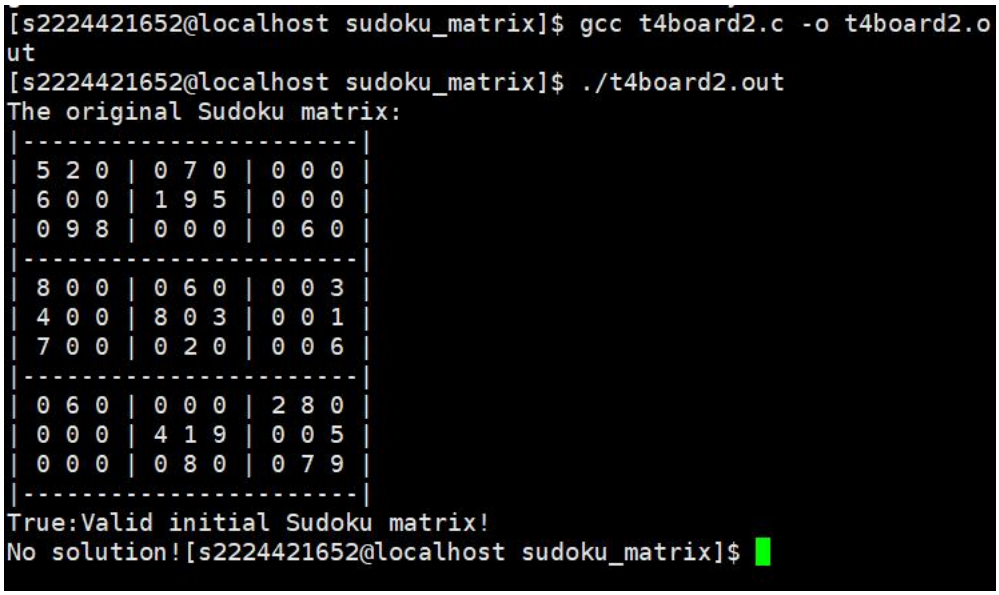
board0：

```
[s2224421652@localhost sudoku_matrix]$ gcc t4board0.c -o t4board0.o
ut
[s2224421652@localhost sudoku_matrix]$ ./t4board0.out
The original Sudoku matrix:
|----------------------|
| 5 3 0 | 0 7 0 | 0 0 0 |
| 6 0 0 | 1 9 5 | 0 0 0 |
| 0 9 8 | 0 0 0 | 0 6 0 |
|----------------------|
| 8 0 0 | 0 6 0 | 0 0 3 |
| 4 0 0 | 8 0 3 | 0 0 1 |
| 7 0 0 | 0 2 0 | 0 0 6 |
|----------------------|
| 0 6 0 | 0 0 0 | 2 8 0 |
| 0 0 0 | 4 1 9 | 0 0 5 |
| 0 0 0 | 0 8 0 | 0 7 9 |
|----------------------|
True:Valid initial Sudoku matrix!
The solution of Sudoku matrix:
|----------------------|
| 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 |
|----------------------|
[s2224421652@localhost sudoku_matrix]$
```

board1：

```
[s2224421652@localhost sudoku_matrix]$ gcc t4board1.c -o t4board1.o
ut
[s2224421652@localhost sudoku_matrix]$ ./t4board1.out
The original Sudoku matrix:
|----------------------|
| 8 3 0 | 0 7 0 | 0 0 0 |
| 6 0 0 | 1 9 5 | 0 0 0 |
| 0 9 8 | 0 0 0 | 0 6 0 |
|----------------------|
| 8 0 0 | 0 6 0 | 0 0 3 |
| 4 0 0 | 8 0 3 | 0 0 1 |
| 7 0 0 | 0 2 0 | 0 0 6 |
|----------------------|
| 0 6 0 | 0 0 0 | 2 8 0 |
| 0 0 0 | 4 1 9 | 0 0 5 |
| 0 0 0 | 0 8 0 | 0 7 9 |
|----------------------|
False:Invalid initial Sudoku matrix!
      The number 8 in the col 1 has been used!
No solution![s2224421652@localhost sudoku_matrix]$
```

board2：



五、实验总结（写实验的收获和存在的问题，必写）

收获：

1) 从编程之外的角度：了解到了 ssh 工具连接服务器的方法，学会了如何维护代码以及如何在多人之间共享代码，包括上传以及修改等等；初步了解了用命令行编译运行程序。这能够让我体会到现实生活中编程的感觉，并让我学习到了除了单纯的写代码外一些很有用但课堂很少涉及的知识，丰富了我的知识面。
2) 从编程之外的角度：学会了如何在编程时做到用户友好，本次实验要求输出一些提示性的语句，如“The original Sudoku matrix: ”等，这些用户友好在现实生活中能给用户提示并使用户修改自己的输入。
3) 从编程之外的角度：提高了独立解决问题的能力：在 gcc 上编译运行时中文变成了乱码，通过搜索最终找到了合适的解决方案，成功解决了问题。
4) 从编程的角度：本次实验四个任务循序渐进，学会了如何将一个复杂的任务拆解为多个步骤依次完成，最终完成复杂的任务。
5) 从编程的角度：学会了换个角度解决问题，任务 2 要求生成随机矩阵，我最开始是从随机生成数字的角度来思考的，但发现无论如何随机数字 1-9 各自出现的次数是不变的，而且随机函数不能控制 1-9 出现的频率。所以我及时更换思考的角度，变成随机生成行列坐标，并加以判断和限制（每三行，对数字 1-9，随机生成行坐标、列坐标），最终解决了问题。
6) 从编程的角度：学会了 dfs，简单来说就是递归+回溯。通过递归 + 回溯的方法枚举所有可能的填法。当递归到最后一个空格后（程序中是 x=9,y=0 的情况（0-index）），如果仍然没有冲突，说明我们找到了答案；在递归的过程中，如果当前的空格不能填下 1-9 任何一个数字，那么就回溯。当无法再回溯（程序中是 x=0,y=0 的情况（0-index））但仍然不能满足时，即认为无解。
7) 从编程的角度：debug 技能提升，完成实验所用的时间大部分花在 debug 上，在摸索中从最原始的在程序各个角落输出，逐渐改进方法，提高了 debug 的效率。
8) 从编程的角度：学会了优化程序。在判断目前的矩阵是否符合数独的定义时，原方案是多重循环进行检查，但在任务 4 中发现这种方法速度太慢，于是额外定义了三个二维数

组，对（i，j）元素，分别表示第 i 行、第 i 列、第 i 个 3*3 矩阵中数字 j+1 是否存在。结果是速度大大提升。

9) 从编程的角度：学会思考其他解法，一题多解，可以想想各种方法，既能复习以前的知识，又能拓宽思维。

存在的问题：

1) 本次实验关于小组的部分做的不太好，下次实验时要先联系好组内成员，先进行分工，再开始实验，要充分理解并体会小组存在的意义，这次实验要引以为训。
2) 自己编程时有时候没有完全想明白就开始编程，导致最后 debug 时特别痛苦。例如任务 4dfs 一开始终止条件和 return 语句想的不透彻，导致 debug 时才发现错误。下次实验要先想明白再编程，会大大提高效率。
3) 之前编程一直都是本地写完直接在平台上看运行通过没有，保存也只是保存在本地的文件夹，但其实这是不贴合现实运用的。经过本次实验，要体会现实中编程的完整流程，并学会运用一些工具，提高自己工作的效率。
4) 自己写的代码应具有可读性，并在必要地方加以注释进行解释，方便其他人检查和自己后续查阅。一方面是将解法实现的代码具有可读性，另一方面是变量名、数组名、函数名的命名具有可读性。