实验 3——数独矩阵

1. 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

2. 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，使之成为一个完整的“数独矩阵”。

注：本次实验最终应该生成 4 个代码文件，分别对应四部分内容。

3. 实验内容

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。

提示：

1) 选择整型数组或者字符数组，任选其一实现格式化输出即可；
2) 后续的实验部分仍会用到该方法。

示例：

对于数组 board[9][9]：

```
char 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'}};
```

```
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}};
```

它的格式化输出为：

```
|-----------------------|
| 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 |
|-----------------------|
```

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。

提示：

1) 选择整型数组或者字符数组，任选其一实现即可；
2) 随机数可以使用 rand()函数生成，rand()函数的头文件为<stdlib.h>，rand()%10 就可以随机生成一个 0-9 之间的数（包含 0 和 9）；
3) 一般在程序运行开始时，会使用 srand(time(NULL))函数重置随机数种子，否则会因为随机数种子相同，导致每次随机产生的数字都是相同的。

time(NULL)会返回当前的时间，srand(time(NULL))就表示使用当前的时间作为随机数种子，time(NULL)函数的头文件为<time.h>，srand()函数的头文件为<stdlib.h>。

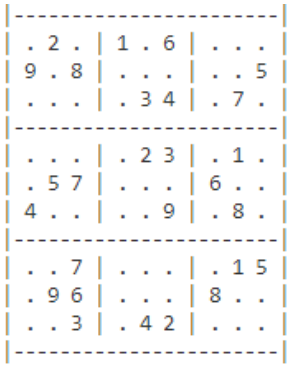
4) 一般整个程序只需要在开始的时候使用一次 srand(time(NULL))就行，因为简单程序运行速度非常快，在一秒之内就运行完成了，而 time()函数得到的时间只能精确到秒，所以在一个程序中多次调用该方法得到的时间都是一样的，这样一来，种子也就是一样的，随机数也就一样了。

示例：

```c
/*一个 rand()函数和 srand()使用示例*/
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
int main(){
    srand(time(NULL));
    for (int i = 0; i < 10; i++)
        printf("%d ",rand()%10);
    return 0;
}
```

```
Output:
4 8 4 1 6 3 5 9 0 8
```

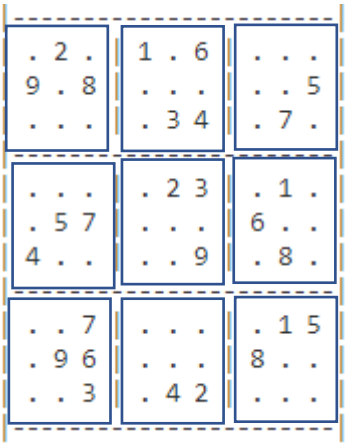
产生的 9×9 的不完整的矩阵如下：



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。



要求：

1) 程序开始时，输出“The original Sudoku matrix: ”；
2) 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；
3) 如果满足定义的话，输出“True:Valid initial Sudoku matrix!”；
4) 如果不满足定义的话，先输出“False:Invalid initial Sudoku matrix!”，再输出不满足的原因，比如“The number 1 in the col 6 has been used!”，或者“The number 5 in the col 1 has been used!”，或者“The number 3 in the block 2 has been used!”。分别表示，数字 1 在第 6 行多次使用；数字 5 在第 1 行多次使用；数字 3 在第 2 个 3x3 的矩阵内多次使用。不满足的原因可能有多个，只需要输出一个即可。

提示：

- 该部分不要求和第二题结合，不完整的“数独矩阵”可以是自己手动定义一个数组，然后手动赋值，也可以使用后面示例中给出的数组定义；
- 对于一个矩阵，只要出现一个不满足定义的值，它就一定不会满足“数独矩阵”的定义；

- 如果有需要，文档的最后会提供一种简单的思路作为参考。

示例：

```
char board0[9][9] = {{'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'}};
char board1[9][9] = {{'8', '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'}};
```

```
int board0[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}};
int board1[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}};
```

如上两个数组中，`board0` 的**执行结果**为：

```
The original Sudoku matrix:
    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
True:Valid initial Sudoku matrix!
```

而 `board1` 的**执行结果**为：

```
The original Sudoku matrix:
    8 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
False:Invalid initial Sudoku matrix!
      The number 8 in the block 1 has been used!
```

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!”。

提示：

- 一个不完整的数独（部分已被填充）不一定是可解的；
- 该部分不要求和第二题结合，不完整的“数独矩阵”可以是自己手动定义一个数组，然后手动赋值，也可以使用示例中给出的数组定义；
- 在填充不完整“数独矩阵”的过程中，任意时刻的状态（不管填充了几个数）始终都应该是要满足“数独矩阵”的定义的，所以过程中当填充了某个数字后不满足定义了，最终填充好的矩阵一定不满足“数独矩阵”的定义；
- 如果有需要，文档的最后会提供一种简单的思路作为参考。

示例：

如下两个数组 `board0[9][9]`和 `board1[9][9]`中，`board0[9][9]`满足“数独矩阵”的定义且有解；`board1[9][9]`不满足“数独矩阵”的定义，没有解；`board2[9][9]` 满足“数独矩阵”的定义但没有解。

```
char board0[9][9] = {{'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'}};

char board1[9][9] = {{'8', '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'}};

char board2[9][9] = {{'5', '2', '.', '.', '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'}};
```

```
int board0[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}};

int board1[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 board2[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}};
```

board0 运行的结果为：

```
The original Sudoku matrix:
    |----------------------|
    | 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 |
    |----------------------|
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 |
    |----------------------|
```

board1 运行的结果为：

```
The original Sudoku matrix:
    |----------------------|
    | 8 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 |
    |----------------------|
False:Invalid initial Sudoku matrix!
      The number 8 in the block 1 has been used!
No solution!
```

board2 运行的结果为：

```
The original Sudoku matrix:
    |----------------------|
    | 5 2 . | . 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 |
    |----------------------|
True:Valid initial Sudoku matrix!
No solution!
```

4. 实验报告

1) 按照实验报告模板撰写实验报告；
2) 采用互评方式批改，在实验报告截止时间前提交实验报告；在实验评价截止前提交评价结果。

5. 附录

第 3 部分的思路：

1. 分别为行、列、3×3 的矩阵块建立一个数组。
2. 以行数组为例，这个数组中记录了某一行中每个数字出现的次数，按照定义，这个次数只能小于等于 1，列数组和矩阵块数组类似。
3. 遍历原始数组，填充这三个数组。
4. 当某一个数组中记录的某个数字的次数大于 1 时,即可认定该数组不是“数独矩阵”。

第 4 部分的思路：

1. 在实现完第 3 部分后，第 4 部分基本上就是第 3 部分的重复工作。
2. 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，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任何一个数字，那么就进行回溯。当无法再回溯但仍然不能满足时，即认为无解。

如上只是提供一个思路作为参考，具体如何实现还需要自己进一步的思考。