1101 計算機概論作業 8
繳交期限：1 月 7 日星期五下午 5:00

題目：將 1 至 9 的整數以下列方式排列成為方陣時，每一列、每一欄以及兩組對角線上的數字，加總都是 15。如果是 1 至 16 的數字，則加總都是 34。這個作業將引導你寫出一支產出這種方陣的程式。寫這支程式最容易的方法，就是使用作業 7 的架構。

2	9	4
7	5	3
6	1	8

1	12	13	8
2	14	7	11
15	3	10	6
16	5	4	9

程式必須使用下列 structure。

```
const int DIMENSION = 3;
struct sGame  {
   int x, y;
   int board[DIMENSION][DIMENSION];

};
```

程式請使用下列的 main()函數，不要更動。如果你與作業七比較，會發現這兩個看來不太相同的題目，其實 main()函數十分相像。其中所呼叫的各函數將在後面逐一說明。

```
void main()
{
    struct sGame game;
    bool done = false;
    game = Initialize();
    do {
        game = TryTrySee(game);
        if (OK(game)) game = NextCell(game);
        else game = PrevCell(game);
        done = (game.y == DIMENSION) || (game.y < 0);
    } while (!done);
    Output(game);
    return;

}
```

Initialize()

這個函數必須具備以下功能：

1. 宣告一個 `struct sGame` 變數，名稱不限（以下假設名稱為 game）。

2. 將 game.x 以及 game.y 均設定初始值為 0。
3. 將 game.board 的所有元素之值均設為 0。
4. 回傳 game。

`struct sGame NextCell(struct sGame game)`

這個函數必須具備以下功能：

1. 將 game.x 的值增加 1。
2. 檢查 game.x 的值是否等於 DIMENSION。如果是，就將 game.x 的值設定為 0，同時將 game.y 的值增加 1。

也就是說，這個函數的功能是將「現在正在填的位置」移到下一格。如果已經是在這行的最後一格，則移到下一行的第 0 格。

`struct sGame PrevCell(struct sGame game)`

這個函數必須具備以下功能：

1. 將 game.board[game.x][game.y]的值設定為 0。
2. 將 game.x 的值減少 1。
3. 檢查 game.x 的值是否小於 0。如果是，就將 game.x 的值設定為 DIMENSION-1，同時將 game.y 的值減少 1。

也就是說，這個函數的功能是將「現在正在填的位置」倒退到前一格。如果已經是在這行的 0 格，則移到前一行的最後一格，也就是第 DIMENSION-1 格。

`void Output(struct sGame game)`

將 game.board 的內容，以方陣的形式呈現出來，如上面各圖所示。

`struct sGame TryTrySee(struct sGame game)`

這個函數的任務，是為「現在正在填的位置」找到一個合格的數值，也就是要為 game.board[game.x][game.y]找到合格的數值。方法是：

1. 將這格的數值增加 1。注意：直接增加 1，不要先設定 0 或其他初始值。該設定的初始值，在 Initialize()中都已經做過了。
2. 呼叫 **OK()**來檢查這個值是否合格。如果合格就結束函數，回傳 game。
3. 如果 **OK()**認為不合格，就檢查這格的數值是否已經等於 DIMENSION 的平方。若已經等於，則結束函數，回傳 game。
4. 否則回到步驟 1。

`bool OK(struct sGame game)`

這個函數的任務，是檢查「現在正在填的位置」的數值（也就是 game.board[game.x][game.y]的值）是否合格。如果合格就回傳 true，否則回傳 false。而合格的條件，是必須通過以下**每一項**的檢查。

1. 與方陣中其他的數字有沒有重複。如果有任何一個重複就是不合格。
2. 與[game.x][game.y]同一列的數字（也就是[game.x][*]）如果都已經填上去，則其加總必須等於目標值（如果是 3x3，目標值是 15；如果是 4x4，目標值是 34，依此類推）。如果有些尚未填上去（也就是其值為 0），則加總必須小於目標值。如果不滿足這些條件就是不合格。
3. 與[game.x][game.y]同一欄的數字（也就是[*][game.y]）的加總檢查，方法同上。
4. 如果[game.x][game.y]位在任一對角線上，就要檢查這條對角線的加總是否合格，邏輯同上。注意同一格有可能同時位在兩條對角線上。

回去對照前面所提供的 main()函數，就會發現程式在執行的演算法如下。這種演算法稱為「回溯法」，是一種人工智慧技術很常用的演算法。

1. 初始化，並設定為由(0,0)開始。
2. 為「現在正在填的位置」找到一個數值。
3. 如果「現在正在填的位置」的數值是合格的，則進到下一格。
4. 否則，「現在正在填的位置」的數值不合格，表示這格無法找到合格的數值。此時就退到前一格。
5. 如果倒退到(0,0)的前面，或者進到最後一格的後面，就終止演算法並輸出結果。

正常狀況下，在上列第 5 步不應該會發生倒退到(0,0)的前面的狀況。但是如果程式有錯，就有可能會發生這種狀況。因此我們將這項檢查也納入在這裡。如果發生了倒退到(0,0)的前面的狀況，就表示該檢查程式了。