Sudoku Project (I)

題目

以程式解答 Sudoku 問題。

輸入：Sudoku 表格中共 81 個數字，空格以 0 表示，其餘 1~9 不變。
輸出：Sudoku 解答。如果沒有解答，則印出無解。

解題思惟

1. Sudoku 表格中共 9x9=81 個數字，以二維整數陣列 puzzle[9][9]表示。
2. 本題可依次嘗試每一格所有的可能性來進行解答，以遞迴方式進行較為容易。
3. 從上到下，從左到右，依次將每一格編號為 0 到 80。遞迴解答函數從 0 出發，如果能順利走完 81 格，則表示有解；如果無有辦法順利走完，則無解。
4. 遞迴解答函數可以宣告成

```
int solve(int puzzle[][9], int pos);
```

   其中 puzzle 表示目前嘗試的表格狀態，pos 表示目前出發的位置。如果最後能順利走完 81 格，則表示有解，回傳 1；如果無法順利走完，則表示無解，回傳 0。
5. 遞迴函數要怎麼處理呢? 以下討論這個問題。
6. 考慮目前位置 pos，遞迴呼叫時 pos 會加 1（下一個位置），所以 pos 會越來越大，但如果超過 80，那表示所有的位置都走完了，問題有解。所以當 pos>80 時，直接回傳 1，不用再遞迴了，這也是遞迴函數必須要有的終止條件。
7. 如果 pos<=80，那就要處理 pos 這個位置。分兩種情況，第一種是 pos 的位置本來就有數字，第二種是 pos 的位置是空白。
8. 第一種情況（pos 位置有值），不須特別處理，直接看下一格的情況決定，所以直接返回 solve(puzzle, pos+1)即可。
9. 第二種情況，必須把所有數字嘗試一次，每一次嘗試時，先設定數字，然後呼叫下一格的函數來決定是否有解。最後當所有數字都嘗試過之後，如果沒有解的話，應該回傳 0，告訴上一級從這一個位置以下找不到解答。（目前位置往下找不到解答，不代表本題無解，有可能改變上一格的值還可以找到解答。）
10. 在第二種情況下，嘗試每一個數字時，要決定這個數字是否可以使用，這個部份可以宣告一個函數來處理:

```
int isValid(int number, int puzzle[][9], int row, int col);
```

   其中 number 代表要填入的數字，puzzle 代表目前的狀態，row 和 col 為目前的位置。（思考：pos 如何換算成 row 和 col? ）
11. 決定該數字是否可以使用的方法，就是檢查該數字所在的行或列，或者隸屬的 3x3 方塊中有沒有出現過該數字。如果都沒有出現過，則該數字是可以使用的。

程式碼

```
#include <cstdio>

void print_board(int puzzle[][9]);
int solve(int puzzle[][9], int pos);
int isValid(int number, int puzzle[][9], int row, int col);
```

```c
int main()
{
    int puzzle[9][9]= {
        {0, 0, 0, 0, 0, 0, 0, 9, 0},
        {1, 9, 0, 4, 7, 0, 6, 0, 8},
        {0, 5, 2, 8, 1, 9, 4, 0, 7},
        {2, 0, 0, 0, 4, 8, 0, 0, 0},
        {0, 0, 9, 0, 0, 0, 5, 0, 0},
        {0, 0, 0, 7, 5, 0, 0, 0, 9},
        {9, 0, 7, 3, 6, 4, 1, 8, 0},
        {5, 0, 6, 0, 8, 1, 0, 7, 4},
        {0, 8, 0, 0, 0, 0, 0, 0, 0}
    };

    if (solve(puzzle, 0)) print_board(puzzle);
    else printf("No solution!");
    return 0;
}

int solve(int puzzle[][9], int pos)
{
    if (pos==81) return 1;

    int row=pos/9, col=pos%9;
    if (puzzle[row][col]) return solve(puzzle, pos+1);

    for (int nextNum=1; nextNum<10; nextNum++) {
        if(isValid(nextNum, puzzle, row, col)) {
            puzzle[row][col] = nextNum;
            if (solve(puzzle, pos+1)) return 1;
        }
    }
    // F ailed to find a valid value, go back to previous cell for another try
    puzzle[row][col] = 0; // reset before goback
    return 0;
}

int isValid(int number, int puzzle[][9], int row, int col)
{
    int rowStart = (row/3) * 3;
    int colStart = (col/3) * 3;
    for(int i=0; i<9; ++i)
    {
        if (puzzle[row][i] == number) return 0;
        if (puzzle[i][col] == number) return 0;
        if (puzzle[rowStart + (i/3)][colStart + (i%3)] == number) return 0;
    }
    return 1;
}

void print_board(int puzzle[][9])
{
    printf(" +-------+-------+-------+\n");
    for(int i=1; i<10; ++i) {
        for(int j=1; j<10; ++j) {
            if (j%3==1) printf(" | ");
            else printf(" ");
            printf("%d", puzzle[i-1][j-1]);
        }
        printf(" |\n");
        if (i%3 == 0) printf(" +-------+-------+-------+\n");
    }
}
```