CS110: Computer Programming Lab
Department of CSE
IIT, Guwahati

Module 04 Stage 01 Exercise 08

Exercise

Sudocu is a popular puzzle game with simple rules. It has a matrix with 9 rows and 9 columns. The matrix is also viewed as a 3-by-3 matrix of boxes; each box is a matrix of 3 columns and 3 rows.

Here is an example game:

```
9 6 7  4 0 0  0 0 0
0 0 0  0 0 0  1 0 0
0 8 0  6 0 3  7 2 0

0 0 0  2 1 0  0 0 0
0 0 0  9 0 6  2 0 0
1 2 0  0 3 0  0 0 6

0 3 6  0 0 0  0 0 8
7 0 0  0 0 0  0 9 0
0 9 0  0 0 8  0 3 0
```

You may wish to store this as string of length 81 char in your program to avoid repeated typing of this input.

```
char board[9][9],
     pre[] = "967400000000000100080603720000021000"
        "000090620012003000603600000870000090090008030";
```

A long string can be spread over two lines as is done above. Note that I have decided to use a 2-dimensional array (matrix) with the element type `char`. Also, I have used `'0'` instead of the traditional blank character used in Sudoku to ease printing and reading of the game board.

The rules of the game are to fill characters `'1'` to `'9'` in all positions such that no box, row or column has a repeating symbol and no position has `'0'`.

This is your programming assignment in this exercise. My program is about 100 lines and has a number of recursive functions. I will help you with some ideas.

Function `main()` is mainly for intialising the game board and printing the results. Some initial Sudocu configurations may have no solution! So please attend to this issue in your program.

Recursive function `int solve(char board[9][9], int r, int c)` fills square `(c, r)` with a symbol and then recursively calls `solve()` to fill the remaining squares on the board. If position `(c, r)` was filled in the original configuration the function simply makes a recursive call to fill the remaining positions. The recursive call be made with `(c, r)` values set for the next square.

If the position `(c, r)` is marked by `'0'`, it needs a non-conflicting digit as a symbol. Possible digits will be tried one-by-one and tested for conflict-free values. When such a digit is found, the recursive call will be made to fill the other positions. If attempt to fill all remaining unfilled positions succeeds with the chosen digit, a success is reported to the calling function. If attempt fails, a different digit may be chosen and the recursive attempt again made to fill the remaining unfilled positions on the board.

You may wish to use three other functions to gracefully organize your program. These functions are to determine if the chosen digit in location `(c, r)` conflicts with the other entries in its row, column, or box. Their prototype declarations are listed here:

```
int conflictingColumn (char board[9][9], int r, int c);
int conflictingRow (char board[9][9], int r, int c);
int conflictingBox (char board[9][9], int r, int c);
```

These functions return 1 if they find that the digit at location `(c, r)` is also found in (respectively) column `c`, row `r`, or box that contains square `(c, r)`.

A final solution for the problem is copied here:

```
Solved. Well done!
9 6 7 4 2 1 8 5 3
3 5 2 7 8 9 1 6 4
4 8 1 6 5 3 7 2 9
6 4 9 2 1 5 3 8 7
8 7 3 9 4 6 2 1 5
1 2 5 8 3 7 9 4 6
5 3 6 1 9 2 4 7 8
7 1 8 3 6 4 5 9 2
2 9 4 5 7 8 6 3 1
```

Error Reporting and Suggestions for Improvements

My sincere apologies if the document has errors or mistakes. Please report errors in this document to vmm@iitg.ernet.in. Also, I welcome suggestions and advice to improve the quality of the document for the students of CS110.