MSc (Computing Science) 2014-2015
C/C++ Laboratory Examination

Imperial College London

**Tuesday 13 January 2015, 11h00 – 13h10**

- ☞ You are advised to use the first 10 minutes for reading time.
- ☞ You must complete and submit a working program by 13h10.
- ☞ Log into the Lexis exam system using your DoC login as both your login and as your password (**do not use your usual password**).
- ☞ You are required to add to the pre-supplied header file **sudoku.h**, pre-supplied implementation file **sudoku.cpp** and to create a **makefile** according to the specifications overleaf.
- ☞ You will find source files **sudoku.cpp**, **sudoku.h** and **main.cpp**, and data files **easy.dat**, **easy-solution.dat**, **medium.dat**, **mystery1.dat**, **mystery2.dat** and **mystery3.dat** in your Lexis home directory (**/exam**). If one of these files is missing alert the invigilators.
- ☞ **Save your work regularly.**
- ☞ Please log out once the exam has finished. No further action needs to be taken to submit your files.
- ☞ No communication with any other student or with any other computer is permitted.
- ☞ You are not allowed to leave the lab during the first 30 minutes or the last 30 minutes.
- ☞ **This question paper consists of 8 pages.**

## Problem Description

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | 1 | | 8 | 3 | | |
| 2 | 4 | | | 5 | | | | |
| | | 8 | | | | | 6 | 1 |
| | | 4 | | | 9 | | | 3 |
| | 6 | | | | | | 2 | |
| 3 | | | 8 | | | 1 | | |
| 1 | 7 | | | | | 9 | | |
| | | | | 1 | | | 5 | 2 |
| | | 2 | 7 | | 4 | | | |

Figure 1: A Sudoku puzzle board

Sudoku[1] is a popular number-placement puzzle played on a $9 \times 9$ board. Starting with a partially completed board of the kind shown in Figure 1, the objective is to fill the board with digits so that each row of the board, each column of the board and each of the nine $3 \times 3$ sub-boards (outlined by the thick black lines in the figure) contain all of the digits from 1 to 9. Ideally Sudoku problems have just one unique solution.

Typically Sudoku players use logic to deduce what digit should be assigned to a particular board position. For example, one can readily work out that the highlighted square towards the bottom right of Figure 1 must contain the digit 1. This is because the lower-right $3 \times 3$ sub-board must contain a 1 and there is a 1 in the 7th and 8th rows and in the 7th and 9th columns already.

Your challenge here is to code a Sudoku puzzle solver. To aid you in developing your program, you are supplied with five test Sudoku boards: the "easy"[2] one shown in Figure 1, one of "medium" difficulty[3], and three "mystery" boards. The three "mystery" boards form the basis of a bonus challenge to be described later.

[1] A contraction of the Japanese *Suji wa dokushin ni kagiru* which translates as "The digits must be single".
[2] Source: `http://www.theguardian.com/lifeandstyle/2014/dec/22/sudoku-3000-easy`
[3] Source: `http://www.theguardian.com/lifeandstyle/2014/dec/31/sudoku-3007-medium`

## Pre-supplied functions and files

You are supplied with some data files representing Sudoku boards: **easy.dat**, **medium.dat**, **mystery1.dat**, **mystery2.dat** and **mystery3.dat**. We will be storing these boards in our program in a two-dimensional ($9 \times 9$) array of characters. The solution to the "easy" Sudoku board is given in the data file **easy-solution.dat**.

You can use the UNIX command **cat** to inspect these files, e.g.

```
% cat easy.dat
...1.83..
24..5....
..8....61
..4..9..3
.6.....2.
3..8..1..
17....9..
....1..52
..27.4...
```

```
% cat easy-solution.dat
697128345
241653897
538497261
714269583
865371429
329845176
176582934
483916752
952734618
```

You are also supplied with some helper functions (with prototypes in **sudoku.h** and implementations in the file **sudoku.cpp**):

- `void load_board(const char *filename, char board[9][9])` reads in characters from the file with name `filename` into the two-dimensional character array `board`.
- `void display_board(char board[9][9])` displays the 2D character array `board` in a friendly layout familiar to Sudoku players. Row indices (in the form of letters 'A' to 'I') and column indices (in the form of digits '1' to '9') are included in the output to help with the identification of particular board positions.

To illustrate the use of the above functions, consider the code:

```
char board[9][9];
load_board("easy.dat", board))
display_board(board);
```

This results in the output:

```
Loading Sudoku board from file 'easy.dat'... Success!
    1   2   3   4   5   6   7   8   9
  +===========+===========+===========+
A |   :   :   | 1 :   : 8 | 3 :   :   |
  +---+---+---+---+---+---+---+---+---+
B | 2 : 4 :   |   : 5 :   |   :   :   |
  +---+---+---+---+---+---+---+---+---+
C |   :   : 8 |   :   :   |   : 6 : 1 |
  +===========+===========+===========+
D |   :   : 4 |   :   : 9 |   :   : 3 |
  +---+---+---+---+---+---+---+---+---+
E |   : 6 :   |   :   :   |   : 2 :   |
  +---+---+---+---+---+---+---+---+---+
F | 3 :   :   | 8 :   :   | 1 :   :   |
  +===========+===========+===========+
G | 1 : 7 :   |   :   :   | 9 :   :   |
  +---+---+---+---+---+---+---+---+---+
H |   :   :   |   : 1 :   |   : 5 : 2 |
  +---+---+---+---+---+---+---+---+---+
I |   :   : 2 | 7 :   : 4 |   :   :   |
  +===========+===========+===========+
```

You are also supplied with a main program in **main.cpp**.

### Specific Tasks

1. Write a Boolean function `is_complete(board)` which takes a $9 \times 9$ array of characters representing a Sudoku board and returns true if all board positions are occupied by digits, and false otherwise. Note you do *not* need to check whether each digit is logically valid.

   For example, the code:

   ```
   load_board("easy.dat", board);
   cout << "Board is ";
   if (!is_complete(board))
     cout << "NOT ";
   cout << "complete." << endl;
   ```

   should display the output

```
Loading Sudoku board from file 'easy.dat'... Success!
Board is NOT complete.
```

Similarly, the code:

```
load_board("easy-solution.dat", board);
cout << "Board is ";
if (!is_complete(board))
  cout << "NOT ";
cout << "complete." << endl;
```

should display the output

```
Loading Sudoku board from file 'easy.dat'... Success!
Board is complete.
```

2. Write a Boolean function `make_move(position, digit, board)` which attempts to place a digit onto a Sudoku board at a given position. Here `position` is a two-character string denoting row and column board coordinates (e.g. "`I8`" ), `digit` is a character denoting the digit to be placed (from `'1'` to `'9'`), and `board` is a two-dimensional character array. If `position` is invalid (e.g. because the coordinates are out of range), or the placing of the digit at `position` is invalid (e.g. because it would result in two copies of the same digit in the same row), then the return value of the function should be `false`, and `board` should be unaltered. Otherwise, the return value of the function should be `true` and `board` should be updated to reflect the placing of `digit` at `position`.

   For example, the code:

```
load_board("easy.dat", board);
cout << "Putting '1' into I8 is ";
if (!make_move("I8", '1', board))
  cout << "NOT ";
cout << "a valid move." << endl;
```

   should result in the output:

```
Loading Sudoku board from file 'easy.dat'... Success!
Putting '1' into I8 is a valid move.
```

   and board cell I8 should be `'1'`.

3. Write a Boolean function `save_board(filename, board)` which outputs the two-dimensional character array `board` to a file with name `filename`. The return value should be `true` if the file was successfully written, and `false` otherwise.

   For example, the code:

   ```
   load_board("easy.dat", board);
   if (save_board("easy-copy.dat", board))
     cout << "Save board to 'easy-copy.dat' successful."
       << endl;
   else
     cout << "Save board failed." << endl;
   cout << endl;
   ```

   should result in the output:

   ```
   Loading Sudoku board from file 'easy.dat'... Success!
   Save board to 'easy-copy.dat' successful.
   ```

   with **easy-copy.dat** having identical contents to **easy.dat**.

4. Write a Boolean function `solve_board(board)` which attempts to solve the Sudoku puzzle in input/output parameter `board`. The return value of the function should be `true` if a solution is found, in which case `board` should contain the solution found. In the case that a solution does not exist the return value should be `false` and `board` should contain the original board.

   **For full credit for this part, your function – or helper function if you choose to use one – should be recursive**.

   For example, the code:

   ```
   load_board("easy.dat", board);
   if (solve_board(board)) {
     cout << "The 'easy' board has a solution:" << endl;
     display_board(board);
   } else {
     cout << "A solution cannot be found." << endl;
   }
   ```

   should result in the output:

```
Loading Sudoku board from file 'easy.dat'... Success!
The 'easy' board has a solution:
    1   2   3   4   5   6   7   8   9
  +===========+===========+===========+
A | 6 : 9 : 7 | 1 : 2 : 8 | 3 : 4 : 5 |
  +---+---+---+---+---+---+---+---+---+
B | 2 : 4 : 1 | 6 : 5 : 3 | 8 : 9 : 7 |
  +---+---+---+---+---+---+---+---+---+
C | 5 : 3 : 8 | 4 : 9 : 7 | 2 : 6 : 1 |
  +===========+===========+===========+
D | 7 : 1 : 4 | 2 : 6 : 9 | 5 : 8 : 3 |
  +---+---+---+---+---+---+---+---+---+
E | 8 : 6 : 5 | 3 : 7 : 1 | 4 : 2 : 9 |
  +---+---+---+---+---+---+---+---+---+
F | 3 : 2 : 9 | 8 : 4 : 5 | 1 : 7 : 6 |
  +===========+===========+===========+
G | 1 : 7 : 6 | 5 : 8 : 2 | 9 : 3 : 4 |
  +---+---+---+---+---+---+---+---+---+
H | 4 : 8 : 3 | 9 : 1 : 6 | 7 : 5 : 2 |
  +---+---+---+---+---+---+---+---+---+
I | 9 : 5 : 2 | 7 : 3 : 4 | 6 : 1 : 8 |
  +===========+===========+===========+
```

*(The four parts carry, respectively, 20%, 30%, 20% and 30% of the marks)*

### What to hand in

Place your function implementations in the file **sudoku.cpp** and corresponding function declarations in the file **sudoku.h**. Use the file **main.cpp** to test your functions. Create a **makefile** which compiles your submission into an executable file called **sudoku**.

(P.T.O. for hints)

## Hints

1. You will save time if you begin by studying the main program in **main.cpp**, the header file **sudoku.h**, the pre-supplied functions in **sudoku.cpp** and the given data files.
2. All the questions will be **much** easier if you exploit the pre-supplied helper functions.
3. Feel free to define any of your own helper functions which would help to make your code more elegant.
4. Try to attempt all questions. If you cannot get one of the questions to work, try the next one.
5. You are explicitly required to use recursion in your answer to Question 4. You are welcome to use a recursive helper function although using default parameters can avoid this. You are not obliged to use recursion in answering any other question.

## Bonus Challenge (optional)

Consider the following information about the mystery puzzle boards in **mystery1.dat**, **mystery2.dat** and **mystery3.dat**:

- One is a Sudoku board of "hard" difficulty[4].
- One is famous as the world's "hardest" Sudoku[5].
- One is actually impossible to solve[6].

For extra credit, your challenge is to solve (where possible) the mystery Sudoku puzzles and to identify which mystery board matches each of the descriptions above. Place your solutions in **mystery1-solution.dat**, **mystery2-solution.dat** and **mystery3-solution.dat**, and summarise your findings in relation to the identification of the puzzles in a text file called **bonus.txt**.

[4] Source: http://www.theguardian.com/lifeandstyle/2015/jan/02/sudoku-3009-hard

[5] `http://www.telegraph.co.uk/news/science/science-news/9359579/Worlds-hardest-sudoku-can-you-crack-it.html`

[6] Source: "The Book of Impossible Sudoku" by Paul Mutton, April 2008