Sudoku Solver
Practical 3 - C2

Due 9pm Tuesday 7th March
This practical is worth 25% of your coursework mark.

1 Submission

Submit your program as a zip file containing your source, and the output of your program under `stacscheck`. Submit a short (maximum three pages) report covering your design, implementation and testing, and any problems encountered and lessons learned as PDF.

You should test your program using stacscheck. Name the directory your practical is in 'Practical3-C2'. The tests, from a lab machine, can be run by `stacscheck /cs/studres/CS2002/Practicals/Practical3-C2/stacscheck`

2 Introduction

In this practical, you will write a solver for the Sudoku problem. The Sudoku problem of size n consists of a partially completed grid of size $n \times n$, where each cell contains a number between 1 and $n \times n$. A solution fills in all empty cells, under the following conditions:

1. Each row contains each of the numbers 1 to $n \times n$ exactly once
2. Each column each of contains the numbers 1 to $n \times n$ exactly once
3. If we consider the $n \times n$ grid as boxes of height n and width n, then there are n such boxes in each row and each column. Each of these boxes contains each of the numbers 1 to $n \times n$ exactly once.

Here is an example sudoku for $n = 2$,

	2	3	4
3	4	1	2
4		2	1
2		4	

and it's solution

1	2	3	4
3	4	1	2
4	3	2	1
2	1	4	3

And here is one for n=3

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

and it's solution:

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

General Notes:

1. Be sure to write a `Makefile`.
2. All your programs should accept Sudokus in the format described below. You do not have to perform error checking for incorrect inputs. Your programs should in principle work on sudokus from $n = 2$ to $n = 9$ (although for hard Sudokus you might not wait for it to finish!)

3 Sudoku IO

First of all, you will write code to parse in sudoku problems, check their consistency, and print them back out. You should do this in the following steps:

- Design a struct to store a partially solved sudoku problem.
  - Your struct will need a way of representing that a cell does not yet have a value.
  - Write helper functions to read a value out of a sudoku. You may want to store the sudoku as a 1D array, and then have functions to read and write the values at particular squares.
  - Making functions to print out, and make a copy of a sudoku, and free a sudoku. Place these in a file called `sudoku_io.c`.

  The expected input and output form for a sudoku first contains n, then the sudoku expressed a space-separated numbers (using 0 to denote an empty square). The input numbers can be between 0 and 81 (for an $n = 9$ sudoku), so make sure to pad single digits with an extra space. Here are the two example sudokus from above.

```
2
0  2  3  4
3  4  1  2
4  0  2  1
2  0  4  0

3
2  5  0  0  3  0  9  0  1
0  1  0  0  0  4  0  0  0
4  0  7  0  0  0  2  0  8
0  0  5  2  0  0  0  0  0
0  0  0  0  9  8  1  0  0
0  4  0  0  0  3  0  0  0
0  0  0  3  6  0  0  7  2
0  7  0  0  0  0  0  0  3
9  0  3  0  0  0  6  0  4
```

- Write a method called `check_list`, which takes an `int*` representing an array of integers, and an `int` representing the length of the array. This function will treat the array as the contents of a row, column or box of a partially finished sudoku and return one of 3 values (defined as `enum`).

INVALID: The array contains some value twice.

INCOMPLETE: The array is not INVALID, but is not complete.

COMPLETE: The array is not INVALID, and every value is assigned. This means the array contains the values between 1 and 9, in some permutation.

- Using `check_list`, write a function called `check_sudoku` which checks a complete sudoku. `check_sudoku` should return:

INVALID: If any row, column, box contains a number twice.

INCOMPLETE: The sudoku is not INVALID, but is not complete.

COMPLETE: The sudoku is not INVALID, and every value is assigned.

Write a program called `sudoku_check` (with a `Makefile` entry) which reads a sudoku grid from standard input, prints it back out, and then prints one of the strings `INVALID`, `INCOMPLETE` or `COMPLETE` on a blank line.

4 Basic Sudoku Solver

In this section you will write a basic sudoku solver, called `sudoku_solver`. It should accept a sudoku and solve it recursively, printing out the solution (if one exists).

You may use any algorithm, but one basic algorithm design is to solve the problem recursively, as follows:

1. Call `check_sudoku`. If the sudoku is INVALID then return. If it is COMPLETE then you have found an answer.
2. If INCOMPLETE, find an empty cell. For each value for this cell, make a copy of the sudoku, fill in that cell, and call your function recursively.

Your solver will have to keep track of how many solutions it finds, as a problem with two (or more) different solutions is broken. You should stop as soon as you find the problem has more than one solution. Your solver should accept the same input as `sudoku_check` and out output one of 3 things:

1. If the sudoku is unsolvable, print the string `UNSOLVABLE`
2. If the sudoku has more than one solution, print the string `MULTIPLE`
3. If the sudoku has exactly one answer, print the solution.

5 Extension - Advanced Sudoku Solver

You will find that there are some sudoku which your first solver finds hard to solve. `stacscheck` will place a time limit of 30 seconds on your solver. Extend your Sudoku solver to use better reasoning techniques.

Call your extended solver `sudoku_advanced`. If a solver with this name exists, `stacscheck` will run some harder sudoku on it.

Two rules you can try are:

1. Check a row, column, or box. If there is only one empty box, then you can fill in the last remaining value.
2. Given any single cell, look at the values used in the row, column and box this cell occurs in. If only one value is not used, this value must occur in this cell.

6 Policies and Guidelines

If the credit weighting and due date are different from those on MMS, the information on MMS is to be taken as definitive. If you detect a discrepancy please inform the responsible lecturer and level co-ordinator.

6.1 Marking

See the standard mark descriptors in the School Student Handbook: `http://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/feedback.html#Mark_Descriptors`

6.2 Lateness penalty

The standard penalty for late submission applies (Scheme B: 1 mark per 8 hour period, or part thereof): `http://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/assessment.html#lateness-penalties`

6.3 Good academic practice

The University policy on Good Academic Practice applies: `https://www.st-andrews.ac.uk/students/rules/academicpractice/`