CSCI 340
Data Structures and Algorithms
Spring 2018

Programming Project 5 – Sudoku

Due date: Friday, May 4, No extensions!

Objective
You are to write a recursive program to solve Sudoku puzzles.

Background
A Sudoku grid is a 9x9 array with some elements already filled in. The objective of the puzzle is to fill in the remaining elements with the digits 1-9, so that every row, column and 3x3 square contain the digits 1-9. Obviously, if a row must contain the digits 1-9, there can't be any duplicates; similarly, for columns and squares. On the left below is a Sudoku puzzle, with its solution on the right.

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

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

The way people solve Sudoku puzzles is to repeatedly search the grid for locations where the value is forced. For example, there must be a 2 in the bottom row (using Java 0 based indexing, row 8) of the grid. In fact, it must be in one of the two locations in the bottom row of the lower left square, because the middle and right squares already both have 2s (and we can't have duplicates). But note that there is already a 2 in the left most column, so the bottom left location can't be a 2 either. This leaves only one place in the bottom row where a 2 can fit. In Java like syntax: grid[8][2] = 2.

Note that the rules people use to solve Sudokus get much more complex than the example in the previous paragraph, but the ideas are always about the same. The creators of Sudoku puzzles guarantee that there will always be exactly one unique solution to the puzzle. For more information on Sudoku puzzles check http://www.sudoku.com.

Solving a Sudoku Recursively
It is a non-trivial task to ask a computer to solve Sudoku puzzles in the same way people do. To do it correctly would be several thousand lines of code

To solve a Sudoku puzzle using a computer program, a much more natural choice is a recursive depth first search algorithm. Using our example from the previous page, here is the basic idea:

Start at
grid[0][0] and try each of the digits 1-9.
grid[0][0] can't be a 1, since there is already a 1 is the 0th row.
grid[0][0] can't be a 2, since there is already a 2 in the 0th column.
grid[0][0] can be a 3.
Temporarily set grid[0][0] = 3 and move on to the next location (make a recursive call).
grid[0][1] was fixed in the initial board, so grid[0][2] is the next location in which we are interested.
grid[0][2] can't be a 1, 2, 3, 4, 5, 6, 7 or 8. Do you see why?
grid[0][2] can be a 9.
Temporarily set grid[0][2] = 9 and move on to the next location (make a recursive call).
etc.
If you come to a point where there are no legal values to try, or where you have exhausted all the legal values without finding a solution, return from the recursive call and try the next value at the previous cell.

When you find the solution, that is when you find a legal value for grid[8][8] or whatever the last open grid position is, print the solution and exit.

Reading the Initial Grid:
The initial grid will be read in from a file. The file will have the format of one row of data per line, with spaces between the entries. Sample inputs and solutions are available at /home/student/Classes/Cs340/Sudoku/ Open spaces in the grid will be represented by 0s in the file. Thus the file in our example would be:

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

What to Submit
I will pass as a runtime parameter the filename to be solved, your class should create the file <inputFilename>.solved that contains the same format as the problem file, but with all the answers in the grid. I will only give you solve-able puzzles.

Submit only a single Sudoku.java.

- Use package name **sudoku** and class **Sudoku** for your main class
- I will run the following commands on your submission, for full credit this should generate output:

```
javac –d . Sudoku.java
java sudoku/Sudoku 15.sdk
```

Grading Considerations
Your program must compile and run to receive any points. Full credit will only be given for submissions that solve the puzzle, use the correct command line input and output and are well written.

Good software engineering is expected. Use lots of comments and follow appropriate indentation, capitalization and variable naming standards.