CMPS 12A
Introduction to Programming
Spring 2010

Programming Assignment 6

Due Thursday June 3, 10:00 pm

In this assignment you will complete the Sudoku solver you began in programming assignment 5. Your program, which will be called `Sudoku.java`, will read from an input file whose name is given on the command line, solve the puzzle encoded in that file, print the original puzzle and its solution to standard output, then quit. The input and output formats, and command line interface will be exactly those you implemented in pa5.

Required Features

There are many techniques for solving Sudoku puzzles, and you should read the Wikipedia article mentioned in pa5, and its references, to learn about them. In this project you will implement a strategy called "candidate elimination". Basically this means that for each cell in the puzzle, you keep track of exactly which numbers remain as possible candidates for that cell.

A Sudoku Puzzle

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

Solution

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

For example, starting with the initial unfilled grid above, we may conclude that cell (8, 5) (which means row 8, column 5) cannot contain anything already in row 8 (namely 6, 3, 9, or 7), already in column 5 (3, 2, 4, or 9), and already in its "box" (3, 4, 5, 6, 7, or 9). Thus the only possibilities left for cell (8, 5) are the numbers 1 and 8. Similarly one can check that the remaining candidates for cell (3, 9) are 3 and 5, and cell (5, 5) must be either 5 or 6. Also notice that cells (7, 6) and (1, 1) have only one candidate remaining, namely 2 and 9 respectively, and therefore those values can be filled in. Once new values are entered, they can be used to eliminate candidates in their row, column, and box. Thus one can make several passes over the puzzle, listing the possible candidates for unfilled cells, filling in the cells with only one candidate, then using those new entries to impose additional constraints on the other cells by eliminating some of their remaining candidates. This process continues until all cells are filled and the puzzle is solved. Note that this process works for the so-called "easy Sudoku", but not for more difficult instances of the puzzle. You may assume that all testing we do will be on easy Sudoku puzzles.

Your program will keep track of the remaining candidates for each cell by using a 3-dimensional `int` array. If you call this array `possible` for instance, its declaration would be

```
int[][][] possible  = new int[10][10][10];
```

It is recommended (though not required) that you avoid 0 as a row or column index in this array, just as was recommended in pa5 for the 2-dimensional array representing the puzzle grid itself. However, as explained below, "level" 0 in this array can be put to good use. How will array `possible` keep track of the remaining candidates for each cell in the puzzle grid? For each k in the range $1 \le k \le 9$, the value stored in `possible[i][j][k]` will equal 1 if k **is** a remaining candidate for cell (i, j), and will equal 0 if k **is not** a remaining candidate for cell (i, j). The value stored in `possible[i][j][0]` will be the **number of** remaining candidates for cell (i, j). Initialization of array `possible` should proceed as follows:

- If cell (i, j) is initially unfilled: set `possible[i][j][0]` $= 9$ and `possible[i][j][k]` $= 1$ for all k in the range $1 \le k \le 9$.
- If cell (i, j) is initially filled with the number m (where $1 \le m \le 9$): set `possible[i][j][0]` $= 1$, `possible[i][j][m]` $= 1$, and `possible[i][j][k]` $= 0$ for all $k \ne m$ in the range $1 \le k \le 9$.

In addition to the four methods mentioned in the pa5 project description, two methods are required in this project with the following names and signatures.

```
static void updatePossible(int[][] G, int[][][] P)
static void updateGrid(int[][] G, int[][][] P)
```

Method `updatePossible()` will use the entries in the puzzle grid `G` to eliminate candidates from the `possible` array, known locally as `P`. Following the above example, `updatePossible()` would set `P[8][5][k]` equal to 0 for $k = 2, 3, 4, 5, 6, 7,$ and 9. The value stored in `P[8][5][k]` should retain its initial value of 1 for $k = 1$, and $k = 8$ since these are the remaining possibilities for cell (8, 5). Finally, `P[8][5][0]` should be reduced from its initial value of 9 to 2, since 7 candidates have been eliminated and 2 remain. Note that method `updatePossible()` alters its array parameter `P`, but not `G`. Method `updateGrid()` will fill currently unfilled cells in the puzzle grid `G` whose correct values can be deduced from the information in `P`. Observe that this method alters array parameter `G`, but not `P`.

Method `main()` will perform all the necessary initialization of the puzzle array `grid` as well as array `possible`. Array grid will be initialized by calling method `getGrid()` from pa5. You may choose to initialize array `possible` within `main()`, or you may write a special purpose method of your own to do this. After initialization is complete, function `main()` will print the unfinished puzzle by calling `printGrid()`, then enter a loop resembling

```
while(!isFilled(grid)){
   updatePossible(grid, possible);
   updateGrid(grid, possible);
}
```

When this loop terminates, the puzzle has no empty cells, and so is presumably solved if your update methods work properly. All that remains is to call `printGrid()` one last time to print out the solved puzzle.

Remarks

Most students will find this to be a challenging project, so you are urged to begin early. Break the project into smaller manageable pieces, build and test the pieces one at a time and slowly assemble the entire program. Once a method is written, test it by calling it on specific `grid` and `possible` arrays which you

prescribe, then print out the current state of both arrays to see if your function is having the expected effect. It may be useful in this regard to write a method with the signature

```
static void printPossible(int[][][] P)
```

to print out the state of array `possible`. Since this method would be for diagnostic purposes only, the output format would be entirely up to you. Of the two required methods, `updatePossible()` is by far the more complicated one to implement. Only when you are sure both methods are working, should you set up the above loop.

Your source file for this project will be called `Sudoku.java`, and should be submitted to the assignment name `pa6`. You will also submit a `Makefile` with this project similar the one you submitted for pa5 which will create an executable jar file called `Sudoku`. This Makefile should also include targets called `clean` and `spotless` as described in pa5. Your Makefile need not include a submit utility.