CMPS 12A
Introduction to Programming
Spring 2007

Programming Assignment 4

Due Thursday June 7, 10:00 pm

In this programming assignment you will write a java program to solve Sudoku puzzles. If you are unfamiliar with Sudoku, try looking at `http://en.wikipedia.org/wiki/Sudoku`, or any of the hundreds of websites devoted to the puzzle. Briefly, Sudoku is a logic puzzle in which one fills a 9×9 grid with numbers in such a way that every row, column, and 3×3 "box" contains each of the numbers 1 through 9 exactly once. An instance of the puzzle consists of a partially filled 9×9 grid, and it is the solver's task to fill in the remaining cells using logical deduction.

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

It has been shown that there are 6,670,903,752,021,072,936,960 distinct solution grids. If one takes into account symmetries in the puzzle such as rotations, reflections, and permutations, then there are just 5,472,730,538 essentially different ways to fill in a grid. Each filled grid however, gives rise to a large number of distinct puzzles by removing the contents of different sets of cells. Notice that if there are not enough filled cells to begin with, then the puzzle solution is not uniquely determined. To be a valid Sudoku puzzle, there can be only one correct way to fill in the empty cells.

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 file will consist of a space separated list of 81 numbers giving the nine rows of the puzzle from left to right, and top to bottom. Extra intervening white space (spaces, tabs,

and newlines) will be ignored by your program. In this file, a blank cell will be represented by the number zero. Thus an input file for the above puzzle would look like:

Input file:

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

0 0 0  0 0 0  2 0 7
0 4 0  1 0 7  0 8 0
7 0 9  0 0 0  0 0 0

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

Notice that extra lines and spaces may be included in order to make the "box" structures more visible, but this is not part of the file format. A single line of 81 numbers would look the same to your program. You are not required to check the validity of the input file in any way, and you may therefore assume that your program will be tested only on files representing valid Sudoku puzzles.

Program output will be formatted in a similar manner, except that blank cells will be represented by dashes rather than zeros. Extra lines and spaces will be included to reveal the box structure, and one extra line will separate puzzle from solution. Thus the corresponding output (to standard out) would look like:

Output:

```
- - -  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  - - -


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
```

Your program will represent the puzzle grid as a 2-dimensional `int` array. It is recommended (though not required) that this array be of size 10×10, so that if you call the array `grid` for example, you can

ignore index zero, and refer the seventh row, fifth column of the puzzle as `grid[7][5]` rather than `grid[6][4]`. A small amount of memory is wasted in doing this, but it is worth the gain in clarity.

Required Features

If too many or too few command line arguments are given, or if the file specified on the command line does not exist or cannot be opened, your program will follow the same procedure outlined in programming assignment 3. Your program will define and use static methods with the following names and signatures:

```
static void usage()
static void getGrid(int[][] G, Scanner sc)
static void printGrid(int[][] G)
static boolean isSolved(int[][] G)
```

Method `usage()` will print a usage message to standard error then quit, as in programming assignment 3 and in the example `ReadIntegers.java` on the webpage. Method `getGrid()` will extract integers from the `Scanner` object `sc` and place them in the `int` array `G`, row by row. At the time `getGrid()` is called, the Scanner `sc` will have been initialized to point to the input file by the calling function `main()` in a `try-catch` statement. Method `printGrid()` will print the contents of its array argument to standard output in the format described above. In particular, zeros will be printed as dashes and the box structure will be apparent from the spacing. Method `isSolved()` will return `true` if all entries in its array argument are non-zero, and false otherwise.

Recommended Features

There are many techniques for solving Sudoku puzzles, and you should read the Wikipedia article mentioned above, and its references, to learn about them. It is recommended that in this project, you use the technique described there as "candidate elimination". Basically this means that for each cell in the puzzle, you keep track of exactly which numbers can go in that cell, and which numbers cannot. 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.

It is recommended that your program keep track of the remaining candidates in each cell by using a 3-dimensional `int` array. If you call this array `possible`, its declaration would be

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

You should avoid 0 as a row or column index in this array, just as 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. For k in the range $1 \le k \le 9$, the value stored in `possible[i][j][k]` should equal 1 if k **is** a remaining candidate for cell (i, j), and should equal 0 if k **is not** a remaining candidate for cell (i, j). The

value stored in `possible[i][j][0]` should 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 $1 \le k \le 9$.

Two additional methods are recommended with the following names and signatures.

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

Method `updatePossible()` should 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()` should fill currently unfilled cells in `G` whose correct values can be deduced from the information in `P`. This method alters `G` but not `P`. Method `main()` should perform all the necessary initialization of arrays `grid` and `possible`, print the unfinished puzzle by calling `printGrid()`, then enter a loop resembling

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

When this loop terminates, the puzzle is solved, and all that remains is to call `printGrid()` one last time to print out the solved puzzle.

Remarks

The design outlined in the preceding section can be made to work, but it is just one of many ways to solve the problem, and is not required. You are free to pursue other ideas if you see fit. No matter what the design, this is a moderately 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 project. Start by writing a simple program that reads the input file, initializes the puzzle grid, and just prints out the unsolved puzzle in the proper format. Of the two recommended methods, `updatePossible()` is by far the more complicated one to implement. Once these methods are written, test them by calling them outside the loop, then print out the current state of the grid to see if they are having the expected effect. 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 `pa4`. Do not fail to verify that your program was submitted properly. See lab assignment 1 for instructions on how to do this.