CPSC 170 Spring 2008
Assign 3 : Sudoko Solver
March 19

Description

Sudoku is a puzzle that has been found in most newspapers since in gained popularity in 2005. Typically the puzzle is defined as follows:

- There is a 9x9 grid that is subdivided into nine 3x3 boxes.
- The object of the puzzle is to fill the grid such that the digits 1 to 9 are contained (without repeats) in
  - each row
  - each column
  - each box
- *Seed digits* are initially placed in the grid to define the puzzle and to limit where additional digits can be added by the person solving the puzzle.
- A puzzle is considered *valid* if the seed digits constrain the puzzle so that there is exactly one way to fill the remaining grid spaces.

When trying to solve a Sudoku, people rely logical reasoning to determine where certain digits can and cannot be placed in the grid. This is only fun when you are working on a valid puzzle. If there is more than one solution, it would be impossible to reason through; at some point you would have to guess. Clearly having no solution would be quite disturbing too. Is it possible to write a computer program that a) solves these puzzles and b) can discriminate valid puzzles from invalid ones? Yes – and you are going to write it!

To Do

Instead of trying to reason through the solution, you are going to use a brute-force approach to attempt to solve the puzzle and determine if your solution is unique. You will need to do the following:

- Create a simple Sudoku class.
  - instance data includes:
    - *int boardSize*
    - *int boxSize*
    - *int[][] board*
    - *int numSolutions*
  - A constructor that takes 2 parameters: size and scan
    - size is assigned to *boxSize*

      Note: boxSize for a traditional Sudoku is 3. Your program should work for an arbitrary boxSize (within the bounds of *int* and memory).
    - scan reads input and is used to initialize the board
  - A method that can be used to print the board.

- Create a SudokuSolver class that simply creates and prints a new Sudoku object

  Note: It is probably in your best interest to make this a *file* scanner (get the file name from the command line) so you aren't always entering the same information by hand.

- Test your code to make sure that you are adequately creating and printing the puzzle board. You can download Puzzles.tgz for some sample files. **You should also create a couple of your own**. Note the format: an empty cell is filled with a "0".
- To solve the puzzle, you will need three "helper" methods to determine if a move is valid:
  - boolean validRow(int num, int row)
  - boolean validColumn(int num, int col)
  - boolean validBox(int num, int row, int col)

  These should all be private; external classes should not have aceess to them - instead these methods will all be called by an overall method called *validMove(int num, int row, int col).*

- Verify that these methods work ***completely*** by calling validMove from your SudokuSolver class with some sample puzzle files. You will probably need to create a few of you additional puzzles that provide good test conditions for these methods.

- Write a recursive function in the Sudoku class

  ```
  boolean solvePuzzle (int num, int row, int col)
  ```

  that fills in a solution to the puzzle if one exists and then return true if a solution can be found and returns false otherwise. Some things that you will need to think about:
  - What do I do when I reach the end of a row?
  - How do I know that I have found a solution?
  - What do I do if I tried an invalid number?
  - What if the the grid spot was already filled in (i.e. it was one of the puzzle constraints)
  - What do I do if I run out of digits to try in this spot?

  Note: You are required to write this function without using **ANY** additional loops. You may only call the validMoves method, which will use the loops in the helper methods.

- Test your code to verify that this method can be used to identify the existence of a solution for an input puzzle. If a solution is found, print the puzzle (from the driver program).

- Create a new function

  ```
  boolean isValidSudoku (int num, int row, int col)
  ```

  that returns true if there is exactly one solution, false otherwise.
  - This method will look very, very similar to solvePuzzle.
  - There is no need to find all solutions; after you have found two, you know the puzzle is invalid.
  - The method should print any viable solutions that are found (to prove that multiples exist)

Turn in a hard-copy and e-mail a tar file containing your source code and your test puzzles