CSE 20212 Fundamentals of Computing II
Spring 2015

Lab handout for Week of Feb 23

Introduction:

This week we will develop our own Sudoku solver using the C++ STL.

Many strategies exist and can be found online or your own personal experience. You are welcome to supplement this lab document with any non-brute force/recursive strategy obtained from an outside source, but you must implement it yourself (no "borrowing" of code).

If you decide to implement a simple recursive solver you will receive full credit for solving the puzzles but not for the lab. Please see the rubric that accompanies this assignment on a later page, and ask us if you have questions.

Input format for lab:

Just like the previous lab, a puzzle file will consist of 9 lines, each of which contains 9 integer values separated by a space. The integer values are either in the 1–9 (filled) or 0 (empty). Figure 1 from last week is shown again below:

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

Part 1: Implementing a solver (start in lab but you will not finish in 50 mins)

We recommend you store the Sudoku puzzle using the object from last week's lab. We also recommend storing the possible values in a 3D vector (see below).

In addition to other ideas, you must include a scanning/elimination algorithm to process the cells, which builds upon code you may have written to check for valid moves. Specifically, you should store in the 3D vector which values are possible and eliminate possible values based on the current row/column/minigrid. If only a single value is left, fill the square with that value and restart the process.

To solve the medium and hard puzzles you need to also implement the "singleton" algorithm described here. For example, suppose that a puzzle has the row:

| 1 | A | 5 | 9 | 6 | 7 | 2 | B | C |

and that we know that A can only be 3, 4 or 8, B can only be 3 or 4, and C can only be 3 or 4 (based on other information). The value 8 is a singleton because it only appears in one cell, and therefore cell A must be an 8.

These two algorithms (single possibility and singleton) will be sufficient to solve all three puzzles used for grading. Note, however, that bugs in the code may still solve the medium but not the first sample hard puzzle. For the purposes of this lab solving a hard puzzle will result in a small amount of extra credit.

There are a few online sites with sample Sudoku puzzles and a solver you can use for testing. This is one students' used last year:

http://www.sudokuwiki.org/sudoku.htm

Part 2: Putting it all together

1. The final output out your solver should be the same as the input, with all 0s correctly replaced. Please send it to the screen using cout for grading.

2. Use (and debug if necessary) your solver to solve the easy and medium puzzles posted on the course website and Piazza. As mentioned in class, we are more interested that your solver works and is commented well than which algorithm you use to solve it.

3. In your usual report, add a paragraph on the strategies in your solver, what you think worked well, what did not work, and a general summary of the lab/post-lab. This will primarily be used to give feedback and evaluate the code.

BONUS: Your code will also be run on two hard difficulty problems. If it produces the correct result, you will receive 2 extra credit points per problem (max of 4 total).

Rubric that will be used for grading

Part 1: 40 points

+2 Makefile correctly compiles code
+4 interface (.h) and implementation commented well (2 pts each)
Requested functionality present. Correctness of solver will be judged later using puzzles
- solve member function(s) that fills out the puzzle (3 pts)
- print functionality that displays the final result. (3 pts)

+3 uses fopen/streams properly for input
+5 correctly uses at least one multidimension vector in the final implementation
+ 10 stores additional information useful to solve a puzzle (aka which values are possible for each square (1-9 for empty cells, empty for filled cells) as described in the lab document).
+5 functionality that eliminate values based on the current row/column/minigrid.

+5 functionality such that if only a single value is left, the square is filled in with that value and the process is restarted

Part 2: Finishing up 35/pts

+5 final output of your solver is the same as the input, with all 0s correctly replaced and it is sent to the screen (cout)
+15 solves the easy problem correctly (from Piazza)
+15 solves the medium problem correctly (from Piazza)

Lab Report: 20 points

+4 Explains how the user uses the program.
+6 Explains how the program works internally.
+5 Explains how the program was verified.
+5 Text on what you think worked well, what did not work, and a general summary of the "how" the solver was implemented. This is effectively more detail than usual on how the program works, but from the programmer perspective.

Coder challenge!

Your code can also be run on two very hard difficulty problems. If it produces the correct result, you will receive $5 coder dollars for each correct result. NOTE: You do not have to submit a separate solver, but mention in your report you would like us to run the extra tests.

Additional coder dollars (up to $25) will be awarded for especially creative solutions at the discretion of the lead developer Dinesh. The allotment will be relative to all submissions and scaled accordingly.