

HW 7 - Sudoku Solver

Due: December 13, 2006, 11:59 PM

But note that this overlaps the last closed assignment.

105 points

1 Sudoku

A Sudoku puzzle is a Sudoku grid (see Closed Assignment 2) with some squares filled in. The challenge of the puzzle is to find the remaining numbers so that a solved puzzle results. Fig. 1 shows a Sudoku puzzle.

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

Figure 1: A Sudoku puzzle (From Will Shortz, *Sudoku: Easy to Hard*, St. Martin's Griffin, 2005).

A solved Sudoku puzzle is a 9×9 grid filled with the digits 1–9 with the following 3 properties:

- Every column contains the symbols 1–9 exactly once
- Every row contains the symbols 1–9 exactly once
- Each of the 3×3 minigrids contains the symbols 1–9 exactly once.

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

Figure 2: A solved Sudoku puzzle.

Figure 2 shows the solution of the puzzle from Fig. 1.

We have defined a file format for Sudoku puzzles: A puzzle file consists of 9 lines, each of which contains 9 integer values separated by spaces. The integer values are not necessarily in the range 1–9, and 0. A zero indicates an empty space in the puzzle, so the unsolved puzzle of Fig. 1 is given by the following puzzle file contents:

```

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

```

2 Solution techniques

In this assignment you are to write a C++ code that will try to solve a Sudoku puzzle. It must implement at least the following solution techniques:

- scanning and elimination
- singleton identification

These two techniques are sufficient to solve easy and many moderately difficult Sudoku puzzles. The puzzle in Fig. 1 can be solved using the scanning techniques alone; the more difficult puzzle in Fig. 3 requires both techniques.

Solution techniques require storing information about what values are possible for each square. This list of values is then used to discover the value of some cell, and then the process starts anew.

The scanning and elimination is a means to generate a list of possible values for each cell. We first list (conceptually at least) all possible values 1–9 for an empty cell, and then scan that cell’s row (and column and minigrd) to eliminate from this list all values that already appear in that row (or column or minigrd). At the end of this process, the list of possible cell entries can be examined, and if the list for some cell has been reduced to a single value, then this must be the value that belongs in that cell.

Singleton identification works by also considering the lists of possible values for each cell. We can then examine these lists for all empty cells in a row (or column or minigrd) to see if any value appears in only one of these lists. In this case, that single value must be the value for the cell where it appears.

For example, suppose that a puzzle has the row

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

and that we know (based on other information) that A can only be 3, 4 or 8, B can only be 3 or 4, and C can only be 3 or 4. The value 8 is a singleton—it appears in only one list of possibles for this row. Cell A must therefore be an 8. The second row of Fig. 3 can be partially solved using singletons.

You can use other solution techniques if you like,¹ but must implement at least these (because your code will be tested on puzzles that these techniques will solve).

3 Specific requirements

Write a C++ code called `solveSudoku.cpp` that will try to solve a puzzle read from a file. The program should take only two inputs, the name of a puzzle file to read, and the name of an output file to write. This output file should contain the puzzle, solved as far as your algorithm can take it.

4 Submission

Put your C++ source file called `solveSudoku.cpp` in your `hw7` directory by 11:50 on the due date.

¹See, for example, Wei-Meng Lee, *Programming Sudoku*, Apress, 2006. Note that the names of these strategies vary from source to source.

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

Figure 3: A harder puzzle.

Tools for checking your algorithms

A reference implementation called `solveSudokuRef` will be put in the `Materials` directory. An autograder will be made available before the due date.

A set of Sudoku files, named `p1.sudoku`, `p2.sudoku`, etc. will be placed in the `Materials` directory and on the website. Some of these puzzle files will already be solved, and some will be unsolvable.

Hint

Read the Sudoku puzzle into a `vector<vector<int> >`. But you will want some other data structure too: the reference implementation (currently) uses a

```
vector<vector<vector<bool> > >
```

to keep track of what values might appear in each cell on the grid.