

CS251 Data Structures – Fall 2011

Program 3 – Sudoku Generator

50 points

Due: Nov. 4

By now we're all familiar with Sudoku puzzles, like the one below:

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

The object of this puzzle is to fill in the rest of the grid with numbers such that each row, column and 3 by 3 square contains the digits 1 through 9. The values above are chosen so that there is only one unique solution. For this project you must do one of the following two tasks:

1. Write a program which will solve a Sudoku puzzle. The highest mark you can obtain for this project is a B.
2. Write a program which will generate a Sudoku puzzle. The highest mark you can obtain for this project is an A.

First Project:

Input : Input will consist of 9 lines each containing nine digits, corresponding to the 81 squares in a Sudoku grid. If a value is a 0 it indicates a blank square; otherwise it indicates the number stored in the square.

Algorithm : You should use a branch-and-bound algorithm to solve this problem, trying all possible values in any blank square at each level of recursion. You need to keep track of which values are still valid for any square. When backtracking, care must be taken when “un-doing” this information.

Output : Output should be the solved puzzle, or some indication that it has no solution.

Second Project:

Input : Input will consist of a single integer r which specifies how many digits should be in the puzzle when it is finished. The maximum value for r is obviously 81 (i.e., a fully filled grid), and we'll use 25 as the lowest allowed value – any lower value entered should be set to 25 (you can actually go lower than 25, but it sometimes takes a while to generate a puzzle for r less than 25).

Algorithm : You should use a branch-and-bound algorithm to solve this problem, in two different capacities. First, you should write a branch-and-bound method which when given a sudoku grid returns a boolean value indicating whether or not there is a unique solution. If you solved the previous project, then this should be a small modification to that project's algorithm.

Once you have the checker working, you then must write another branch-and-bound method which, along with a random number generator, completely fills in a grid. Once you have a grid, you can start removing values from it and then check to see if you still have a puzzle with a unique solution. If you remove a value and find that there are more than two solutions, then you must put it back and try removing another one.

Output : Your program should output two items. The puzzle should be output to standard output using the format below:

```

+---+---+---+---+---+
|4| | | | | | | |
+---+---+---+---+---+
| | | | | |1| | |8|
+---+---+---+---+---+
|1|2| | |3| | |7|9|
+---+---+---+---+---+
| | | |8| | |4| | |
+---+---+---+---+---+
| |9| | | | | | |
+---+---+---+---+---+
| | | |6| | |5| |7|
+---+---+---+---+---+
| |8| | | | | |6| |
+---+---+---+---+---+
| | |5| | |4| |3| |
+---+---+---+---+---+
| | |3| |1| | | | |
+---+---+---+---+---+

```

In addition, you should output the solution to your puzzle to the file XXX.ans, where XXX are the first three letters of you last name. Use the same format for the solution. You can open a file for output in C++ as follows:

```

#include <fstream>
...
int main() {
...
    ofstream fout("XXX.ans");
    ...
    fout << ....           // you can then use fout like you would cout

```

You may work in groups of 2 for this project.