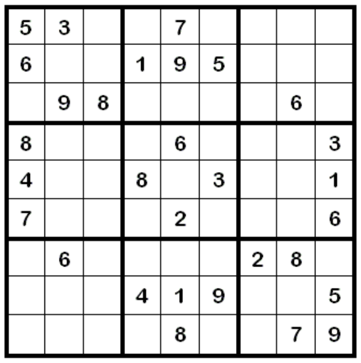
CS270
Fall 2014
Programming assignment No. 3
Due: Thursday, December 4, 11:59pm

Purpose: Gain experience in Prolog & logic programming.

Assignment: Create a **Sudoku** solver (same as earlier projects, only in Prolog this time).

Sudoku is a popular puzzle where you fill in numbers on a grid, trying to keep certain conditions true. To learn more about how Sudoku works, check out http://www.sudoku.com. You'll find a sample puzzle and an explanation of the rules.

Write a Prolog program that reads a file containing an unfinished Sudoku puzzle, then solves the puzzle using the method(s) of your choice. Writing a Prolog program that simply contains the rules for Sudoku is sufficient, as Prolog will use recursive backtracking in its attempt to find a solution. But this strategy may not be the most efficient. Adding simple constraint solver rules will improve efficiency. Once you have solved the puzzle, display the results to the screen. The input should simply contain the numbers in the puzzle delimited by commas, with the underscore character representing unknowns. For example, this puzzle:



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

would be represented in your input as (this input file is in the form of a list of lists, with each row of data in a separate list – and <u>note the final period</u>):

```
[[5,3,_,_,7,_,_,_,_],
[6,_,_,1,9,5,_,_,_],
[_,9,8,_,_,_,_,6,_],
[8,_,_,_,6,_,_,_,3],
[4,_,_,8,_,3,_,_,1],
[7,_,_,_,2,_,_,_,6],
[_,6,_,_,_,_,2,8,_],
[_,_,_,4,1,9,_,_,5],
[_,_,_,_,8,_,_,7,9]].
```

When your program works out the solution, it should print it out to the screen in a slightly prettier fashion, such as:

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

Functional Specifications: You are to provide a solution method of your own design. Your program must be written in Prolog which can be executed by the SWI-Prolog interpreter, which is what we will be using for grading. The input to your program will be in a file. To simplify this project, we will write our programs to read input only from one specific file: **`C:\cs270\prolog\sudoku.txt`**. To make your life a little easier, you will be provided with a Prolog starter file that will contain functions for reading the input board and printing a board out in the specified format. To run your solver, type "**`go.`**" at the prompt. If you want to run your solver on a different file without editing the source, you can use the **`start`** predicate; e.g., type "**`start('c:\\cs270\\prolog\\sudoku-other.txt').`**" at the prompt. If a puzzle is not solvable, the solver should report "false" or "no" (depending upon the Prolog interpreter).

There are several different ways to solve this problem with Prolog. The easiest and most efficient way is to use an optimized library of predefined predicates meant for solving integer constraint problems. SWI-Prolog contains such a set of predicates in its **clpfd** library. Information on this library is available in the **SWI-clpfd.pdf** file posted in Oak with this specification.

It is recommended that you study the predicates available in this library and determine how they can be used to help you solve this problem. Notes: (1) After getting access to the pieces of the Sudoku puzzle, the first constraints that you want to specify are the range (or domain) constraints via the **in** or **ins** predicates. (2) Next comes other constraints & predicates that specify the relationships between the variables in your rules. (3) Finally the last constraint that you should provide in your rule is the **label()** predicate which finally attempts to assign values to all your variables without violating any of your other specified constraints. Again, the **label()** predicate must be the last constraint specified. Hint: You only need to use a few of the provided **clpfd** predicates to solve this problem. [Warning: If you attempt to use the transpose() predicate and receive a warning message "No permission to import clpfd:transpose/2 into use", you can fix this simply add clpfd: to the front of your transpose predicate so that it looks like: clpfd:transpose(X,Y). The error message won't go away, but your program should now work properly. Note: my solution does not use the transpose() predicate but some students have desired to use it in the past.]

The provided Prolog starter file includes code to time your solver. At this point you can compare your C++, Racket, and Prolog solvers all solving the same puzzle – you only need to convert the puzzle to the appropriate input format. To be fair, you should compile your Racket solver before running it (you can create an executable from the DrRacket drop-down menu: Racket->Create Executable…).

SWI-Prolog also has many built-in predicates that work on lists. See the **SWI-lists.pdf** file posted with this specification.

Deadlines: Since the amount of code you must write is small relative to the prior programming projects, we will not have an intermediate checkpoint for this project. The project is to be submitted in its final form on the project due date.

Academic honesty: As stated in class, there are many solutions to Sudoku in many different programming languages available on the Internet. Do not look at the code you may find there. Using code that you find on the Internet is unethical, and of course you would miss the learning opportunity that you get by developing this yourselves. This instructor will report any violations to the university's Honor Council.