MATH 495R, HOMEWORK 4
SOLVING SUDOKU, PART 1

In this lab we will begin to write a program to solve Sudoku puzzles. This week we will write a program that will solve particularly easy sudoku puzzles; next week we will expand it to solve any sudoku.

A sudoku is a 9×9 grid containing digits in some cells, as below.

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

The goal of the puzzle is to insert numbers into the empty squares in such a way that each digit from 1 to 9 occurs exactly once in each row, each column, and each of the nine 3×3-subsquares of the puzzle.

In this lab, we will create some basic functions to solve simple sudoku puzzles. The puzzles that I supply you will consist of a string of 81 digits from 0 to 9, with 0 indicating an empty cell, and each group of nine digits representing a row in the sudoku puzzle. For instance, the sudoku puzzle above would be entered as

400870020080000400006300801700100080612098734000060019193427500807010302020003000

1. To begin, write a function `convert(string)` that converts a string of 81 digits into an array of 81 numbers. You may either use a one-dimensional array of 81 numbers, or a 9×9 array (i.e an array of 9 vectors each containing 9 numbers).
2. Create a function `entry(a,b,grid)` that takes an array of the type you chose in part 1, and returns the (a, b) entry of the Sudoku grid (where the rows and columns of the sudoku grid are labeled with integers from 1 to 9).
3. Create a function `ispossible(a,b,n,grid)` that returns TRUE if n is a possible value for the (a, b) entry of `grid`, and FALSE otherwise. To determine if n is a possible value, it should examine all the nonzero digits in the same column, row, and 3×3 subsquare as (a, b); n is possible if and only if it matches none of these values. If the (a, b) entry already has a nonzero value, this value should be ignored in determining the possible values.
4. Create a function `computepossibilities(a,b,grid)` that returns a list of all possible n for which `ispossible(a,b,n,grid)` is TRUE.

5. Create a function `set(a,b,n,grid)` that sets the (a, b) entry of the grid to the number n (where n is a digit from 1 to 9). If the (a, b) entry of the grid is already nonzero, or if n is not a possible value that can fit into the (a, b) entry of the grid, the function should do nothing and return a value FALSE, otherwise it should set the (a, b) entry of the grid to n and return a value of TRUE.
6. Create a function `singletons(grid)` that checks each of the entries of the grid, and if the entry is empty and has only one possible value, fills it in with the only possible value. If there are no such entries, the function should return FALSE, otherwise it should return TRUE.
7. Create a procedure `simplesolve(grid)` that runs `singletons(grid)` over and over until it returns a value of FALSE.

At this point, the function `simplesolve(grid)` should be able to solve very simple sudoku puzzles. In addition to the puzzle above, here is a list of ten sudokus that it should be able to solve.

004300209005009001070060043006002087190007400050083000600000105003508690042910300
040100050107003960520008000000000017000906800803050620090060543600080700250097100
600120384008459072000006005000264030070080006940003000310000050089700000502000190
497200000100400005000016098620300040300900000001072600002005870000600004530097061
005910308009403060027500100030000201000820007006007004000080000640150700890000420
100005007380900000600000480820001075040760020069002001005039004000020100000046352
009065430007000800600108020003090002501403960804000100030509007056080000070240090
000000657702400100350006000500020009210300500047109008008760090900502030030018206
503070190000006750047190600400038000950200300000010072000804001300001860086720005
060720908084003001700100065900008000071060000002010034000200706030049800215000090

- To easily test your program, you might want to write a function, `issolved(grid)` which will tell you if a sudoku grid has been completely filled in. If you have programmed `set(a,b,n,grid)` correctly and never changed entries of the grid in other ways, a completely filled in puzzle must be correct, but you might want to write a second function `checkgrid(grid)` which will check that for each possible value of (a, b), the (a, b) entry of the grid matches the value of `computepossibleentries(a,b,grid)`. Another potentially useful function would be `printgrid(grid)` which would print the sudoku grid in a human readable form (i.e. as a 3×3 grid of 3×3 grids, with appropriate spacing). You will not be graded on these functions.
- Of the functions described above, the most crucial, and probably the longest, will be `ispossible(a,b,n,grid)`. In particular, determining the entries of the 3×3 subsquare containing the (a, b) entry can be tricky. One possibility is to set $r = (a-1)//3$, $s = (b-1)//3$, and then the 3×3 subsquare will consist of the $(3r+i, 3s+j)$ entries with $1 \leq i, j < 4$.
- If you have extra time, write a function that looks at each column to see if there is a number that can only be placed in a single spot in that column, and if so, places it there. You can do the same for rows and 3×3 subsquares as well. Add these functions into your `simplesolve(grid)` function; they will speed up the solution to more difficult sudokus that we will do next week. If you finish the lab as described above and want to do more, you may want to optimize your functions to run faster; talk to Dr. Doud for ideas about how to do this.