CS 1102

A term. 2013.

Assignment #5

**Due: Tuesday October 8, 2013 @ 9:00 a.m. via turnin**

## General assignment guidelines

1. You should use the *advanced* student language. You may use any constructs in ASL, unless doing so would contradict the assignment details.
2. You must provide data definitions and templates. As data become more complex, being careful with getting started correctly becomes more important.
3. You should spend time thinking about good test cases for each part. Explain why you chose the test cases you did (a quick note in a comment in the line before the test case will usually suffice). Show us that you’ve thought through boundary conditions and ways your program could break.
4. For complicated test cases, provide an explanation for why you believe your output is correct.

## Specific guidelines for this assignment

1. You should modify the existing Sudoku solver as minimally as possible. Within the code, put a comment containing the characters “\*\*\*” before any such change, and explain what you are changing and why.
2. Start with existing Sudoku solver. Do not try to reinvent the wheel!

## Overview

Your goal in this assignment is to generalize the existing Sudoku solver to a wider variety of puzzle types. You will have to think about how to modify code to make it more flexible, and what structures you will need to hold the data concerning how this puzzle game operates.

The first two steps are optional in that you will not hand them in (unless you are unable to complete the assignment, in which case they could be used as partial credit). However, you are ***strongly***advised to complete the first two steps, as they are designed to give a gradual introduction into relaxing the assumptions in the existing code base.

## Step 1: Changing blank tiles

First, familiarize yourself with the Sudoku solver code (in the assignment directory). This code is the same that was in the videolectures, and that we looked at in class.

For this step, we will change the character that represents a blank square to be a “0” (the number zero) rather than false. After you change the constant, you will notice that most of the test cases fail, since the code was designed to look for false as indicating a blank square. So you will need to modify the code at several places.

It may be to your benefit to create a function called not-blank?, **that accepts a tile** and returns whether or not it is a blank tile.

Now change a blank tile definition to be equal to “ “ (a space character). Your not-blank? function will now break, as the “=” function is not defined for strings. You’ll need to modify the function to check *which* equality test to use. Some useful predefined functions are:

* number? Checks whether its input is a number
* string? Checks whether its input is a string
* number->string takes a number and converts it into a string
* string=? Tests two strings to see if they are identical

You will be able to know if your code is working properly if it passes all of the existing check-expects provided with the code. That is, you should not need to modify them.

## Step 2: Sujiken

Your next step is to modify the code so that it handles Sujiken. Sujiken is a Sudoku variant invented by WPI Professor George Heineman, with information available at <http://www.sujiken.com/p/about.html>

Your goal is to determine what code changes are required to solve presented Sujiken puzzles. The easiest approach will be to modify the definition of UNITS within the code to correspond to the list positions in a Sujiken puzzle.

Of course, your Sujiken puzzle should handle a variety of types of blank tiles (as in Step 1).

## Step 3: Generalizing

Your final step is to generalize the Sudoku solver so that it accepts a data structure representing a game description and an initial board, and creates a solution for that game (if a solution exists). Your structure should store the following information

* What is the set of tiles used for this game? (might not be numbers)
* What is a blank square?
* What is the list of UNITS for this game? The list might not be of length 3
* The size of the board may vary

Create a data definition for your structure, and explain how it operates. There are several options for how to proceed with this part of the assignment, so be sure we understand what you are doing..

Modify the Sudoku solver code to access your structure to use the appropriate set of tiles, symbol for blank, list of UNITS, and board. You should create functions to replace the constants, where the function accesses the appropriate portion of your data structure.

You may assume that all puzzles will use the basic test of ensuring that no two of the same tile may occur within any of the units. That is, you do not have to represent puzzles that have rules such as “no unit may sum to 17”.

## Step 4: Sample invocations to create

You should create the following functions to demonstrate your codes functionality. These function should create an appropriate structure that you created in Step 3, and pass that structure to the generic Sudoku solver. That is, these functions should not contain code that performs the search. They should simply create the data structures, and let your existing code handle the solving.

1. solve-Sudoku(board) takes in a board, represented a one-dimensional list, and produces a solution (if one exists). You may assume that tiles are presented as naturals in the range [1,9], and blank tiles will be a “ “.
2. solve-Sujiken(board) takes in a board, represented a one-dimensional list, and produces a solution (if one exists). You may assume that tiles are presented as naturals in the range [1,9], and blank tiles will be a 0.
3. solve-showcase will give you an opportunity to show how far you got in this assignment. You should highlight any advanced functionality which you have been able to create. So if you can handle tiles that are letters, with boards that involve more squares, and UNITS that are more complex, you should create (and document!) such an example here.

## Hints

1. Start slowly. Work on understanding the assumptions in the code, and understanding how to relax those assumptions before creating a data structure in Step 3.