CSCI-1200 Data Structures — Spring 2015
Lab 10 — Sets and Sudoku

Checkpoint 1

For the first checkpoint, we will explore the implementation of the `ds_set` class, along with the use of recursive functions to manipulate binary search trees. Download and examine the files:

http://www.cs.rpi.edu/academics/courses/spring15/csci1200/labs/10_sets/ds_set.h
http://www.cs.rpi.edu/academics/courses/spring15/csci1200/labs/10_sets/test_ds_set.cpp

The implementation of `find` provided in `ds_set.h` is recursive. Implement and test a non-recursive replacement for this function.

To complete this checkpoint: Show one of the TAs your new code. Be prepared to discuss the running time for the two different versions of `find` for various inputs.

Checkpoint 2

The implementation of the copy constructor and the assignment operator is not yet complete because each depends on a private member function called `copy_tree`, the body of which has not yet been written. Write `copy_tree` and then test to see if it works by "uncommenting" the appropriate code from the main function.

To complete this checkpoint: Present your solution to one of the TAs.

Checkpoint 3

For the last checkpoint, we will write a simple Sudoku puzzle solver using the STL `set` container class. Sudoku puzzles are typically played on a 9x9 grid in which some of the cells have been initialized and many of the cells left blank. The instructions are simply to: "Fill in the grid so that every row, every column, and every 3x3 box contains the digits 1 through 9". If you are unfamiliar with Sudoku puzzles you may read about them here: http://en.wikipedia.org/wiki/Sudoku

Please download the following files which contain a partial implementation of a Sudoku puzzle solver:
http://www.cs.rpi.edu/academics/courses/spring15/csci1200/labs/10_sets/sudoku.h
http://www.cs.rpi.edu/academics/courses/spring15/csci1200/labs/10_sets/sudoku.cpp
http://www.cs.rpi.edu/academics/courses/spring15/csci1200/labs/10_sets/puzzles.txt

First familiarize yourself with the code and how we will be representing partially solved Sudoku puzzles. The 2D grid is stored using nested vectors. The elements of the inner vector are sets of integers, which represent the possible values for that cell. We know we're done when each cell has exactly one possible value. If in the course of solving the puzzle we end up with no values in a particular cell, we know the puzzle is impossible to solve. The program handles 4x4 puzzles with 2x2 blocks and 9x9 puzzles with 3x3 blocks (and even larger puzzles too!)

Part 1: Finish the implementation of the `Sudoku` class constructor and the `Sudoku::IsSolved` member function. At this point you should be able to compile and test your program on the input file (using file re-direction). The provided code for this assignment will print out the contents of each cell in the puzzle. At this point, every number (1-4 on the small puzzles and 1-9 on the large puzzles) will be an possible value for each unspecified cell in the puzzle.

Part 2: In the next step you will propagate information from the values in the grid which are known. For example if "7" is the only remaining choice for the cell (i, j) then we can remove "7" from all the other cells on the ith row and the jth column and the corresponding block. Follow the structure in the code provided and complete the `Sudoku::Propagate` member function. Test your program on the provided input.

To complete this checkpoint and the entire lab: Show a TA your completed program. Be prepared to discuss why it can only solve some of the puzzles. Think about how you would extend the program to solve some of the more difficult puzzles.