Programming Assignment 3

CSCI 235 section 2
Instructor Silvano Bernabel
Due Friday November 4 (11:59 PM EST)

Follow the instructions presented in the Programming Rules document. Submit the header, source files, and Makefile. Note, your program needs to compile and run on a cslab host in order to get any credit.

Sudoku Solver

You're going to be implementing a Sudoku solver. Given an input file with the puzzle in its initial state you will be using backtracking to solve it. You will use the linked stack implementation to keep track of the numbers you will try. The input file *puzzle.txt* will contain for example the following:

```
0 0 5  2 0 9  7 8 1
6 8 0  0 7 0  4 0 0
1 0 7  8 3 0  0 6 0

0 2 6  1 0 0  3 4 0
3 0 0  6 0 2  0 0 5
0 5 1  0 4 3  6 0 8

5 0 9  3 0 6  0 7 4
2 4 0  9 5 7  0 3 0
0 6 0  0 0 0  2 0 9
```

Where an empty position is represented by 0. The rules for solving the puzzle are that:

1. No number can appear more than once in a row or in a column.
2. Each row and column must contain the numbers 1 to 9.
3. There are 9 boxes of size 3×3, and each must contain the numbers 1 to 9.

The puzzle is solved when there are no zeros left. Here's what you need to do:

1. Design an ADT for the Sudoku solver. You will decide how to store the numbers. You should index the rows and columns to start from 1.
2. Create a class called **SudokuSolver** that at a minimum implements the following **private** methods and contains these data types:
   (a) `bool insert(int number, int row, int col);` // Insert number at (row,col), then checks for invalid entries. Return false if there is an invalid entry, otherwise return true.

(b) `bool remove(int row, int col);` // removes number at (row,col), then checks for invalid entries. Return false if there is an invalid entry, otherwise return true.

(c) `bool goBack();` // go back if an entry violates any of the above rules.

(d) `int nextEmpty();` // returns the index of the first zero, from left to right, and top to bottom order. The return value must be from 0 to 81, and 0 is used to indicate no zeros were found. For example, 1 is the upper left corner. (You may need to convert to row and column format)

(e) `int getMissingInBox(int row, int col);` // Each 3×3 box is missing 3 numbers. So in the example above, for box 1 (top-left) the function would return 2349 (one integer).

(f) `int getMissingInRow(int row);` // In the example, for row 1 the function would return 346.

(g) `int getMissingInCol(int col);` // For example, for column 1 the function would return 4789.

(h) `bool readInputFile(string filename);` // reads the input file and stores the values in some data type of your choice. Return false if there is an error reading the file.

(i) `LinkedStack<int> stack;` // store the numbers to try when filling in the puzzle.

3. Include the following **public** methods in the class:

   (a) `SudokuSolver(string filename);` // loads a puzzle from the input file

   (b) `bool SolvePuzzle(int maxBackSteps)` // try to solve the puzzle before reaching maxBackSteps (for example 10,000 steps), if the puzzle can't be solved return false. This method will use the private functions to solve the puzzle using the stack as a means of backtracking.

   (c) `DisplayPuzzle()` // display the current state of the puzzle.

4. From the `main()` function you will create the **SudokuSolver** object. Call the method **SolvePuzzle()** and then **DisplayPuzzle()** if the puzzle was solved. If the puzzle is not solved after maxBackSteps is reached, ask the user if they would like to continue. If they answer yes, call SolvePuzzle() again to continue working on the puzzle for another maxBackSteps. This process will continue until the user decides to quit the program.