CMSC 106 Project #4 Fall 2011

Due date: Friday, November 18, 2011

1 Purpose

In this project you will write a program using arrays and character input. You are a big fan of the game Sudoku but you are not always sure you are solving the puzzle correctly so your job is to write a sudoku checker. The two stages of checking are (1) checking a complete board to see if it is a valid solution and (2) checking a partially filled board to see if it is good so far.

In a sudoku puzzle solution, you have a 9x9 grid where every row must contain exactly one of each of the numbers 1-9, every column must contain exactly one of each of the numbers 1-9, and every 3x3 subsquare must contain exactly one of each of the numbers 1-9. If the sudoku puzzle is not yet completed, one or more of these could be replaced by spaces. There are 9, 3x3 subsquares that make up the 9x9 square.

2 Project description

As you read this section, you may also want to look at the `input` and their corresponding `output` as a sample the different test files provided will create. Your final product must run correctly with the `sudokuMain.c` and `sudokuHelpers.c` (both enclosed) when compiled with your `sudoku.c` when all parts are. The files whose names start with the word `test` are only there to allow you to test parts of your program as you go along. The output of your program must match the `outfiles` exactly.

The input to your program is to consist of a sequence of input lines with no prompt requesting them. These input lines correspond to the lines of the sudoku board. Except, as described in the input function comments, the input is characters (so can contain spaces and end of lines) while the values to be stored are integers. These should be given as input to the program by using input redirection. Assuming you have an executable file named a.out in the current directory, you can use the command:

```
a.out  <  inputFileName
```

This will run the program, and all input will come from the file rather than from the keyboard. These are just text files so you can create your own test files using emacs (just make sure to put 9 characters on each line followed immediately by the new line character). You should make additional test files to test various possibilities - these will not be graded, but they will help you pass all of the tests.

You must implement all of the functions as described in the sudoku.c file.

3 Project requirements

All your C programs in this course should be written in ANSI C, which means they must compile and run correctly with `gcc -Wall -Werror -ansi -pedantic-errors` on the grace system as was setup at the beginning of the course. You will lose credit if your program generates any warning messages when it is compiled. Even if you already know what they are, you may not use any C language features other than those introduced so far in lecture or lab. Note specifically that `structs` may not be used. In addition, the `goto` may not be used. Using C features not presented in the lectures before the date of the assignment, or using the `goto` statement or `structs` **will result** in losing credit.

You may not modify the `sudokuMain.c` or the `sudokuHelpers.c` file at all. All of your code must appear in the `sudoku.c` file. You may add other methods or constants to the `sudoku.c` if you would like, but they will not have a prototype in the `sudoku.h`.

All work for this project must be your own individual work. You may not work with any other student or compare code in any way. If you need help see either the instructor or TA. Obtaining help from others will be considered cheating. Getting code from a web page or other location will be considered plagerism.

4 Project requirements

1. All of your projects for this course must have a comment near their top which contains your name, your Glue ID, your section number, your TA's name, and your university ID number.

2. All your C programs in this course should be written in ANSI C, which means they must compile and run correctly when compiled with the compiler `gcc`, with the options `-Wall`, `-Werror`, `-ansi` and `-pedantic-errors`.

3. You must implement each of the functions defined in the sudoku.c file without changing the name, return type, parameter(s) or what it does.

4. make sure it runs with the test input/output files given before submitting it, and make sure you test it on others that are different from those given.

5. You **may not** use any C language features except those introduced thus far in classs. Note that using any features of C other than these will result in **losing credit** when your project is graded. For this project you may not use arrays or structures.

6. The Campus Senate has adopted a policy asking students to include the following statement on each assignment in every course: "I pledge on my honor that I have not given or received any unauthorized assistance on this assignment." Consequently you're asked to include this pledge in a comment near the top of your program. See below for important information regarding violations of academic integrity.

7. Your program should be written using good programming style and formatting. Any instances of bad style will result in losing points. The good style issuse we will be grading for in this project consists of:
   - having an initial block comment with information as stated above.
   - use of adequate descriptive comments throughout, to indicate what your program is doing and how, also make sure there are not too many comments so that readability is hindered (points lost for both too many or too few)
   - neat and readable indentation and formatting, including consistent alignment of braces and indentation of subsidiary statements, and avoiding writing any statements or comments which are wider than the standard terminal window width of 80 columns,
   - writing clear and readable code,
   - not using any global (or file scope) variables,
   - having descriptive identifiers (variable names, function names, names for symbolic constants, etc) [note: using single letter variable names or names that are not descriptive of their purpose will result in loss of credit],
   - making sure that you have no "dead code" (code that can never be reached) and making sure that you have no "code duplication" (code that already appears another place in the program - or very similar code that could have been done once by creating a helper funciton,
   - use of symbolic constants for any special values which don't change and which are used in several places in your code,
   - avoiding disallowed C language features as discussed above,
   - and, I also suggest fully blocking all loops and conditionals – use a curly brace to make it more obvious what lines are dependent on which headers.