CMSC 106 Project #4 Fall 2010

Due date: Tuesday, November 2, 2010

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 `infiles` and `outfiles` as a sample of what the `sudokuMain.c` enclosed should give when compiled with your `sudoku.c` and the `sudokuHelpers.c` also enclosed. The output of your program should 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.

You must implement each of the functions as described in the sudoku.c file. You should also make test files for these modeled on the public tests included.

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, neither the `goto` nor the `continue` statement may be used, and the `break` statement may not be used in any loop. Using C features not presented in the lectures before the date of the assignment, or using the `goto` statement, or `break` or `continue` in loops **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.

Your program must have a comment near the top which contains your name, login ID, student ID, your section number, and an original description of the action and operation of the program. At the end of this comment block, you must also include the honesty statement: "I pledge on my honor that I have not given or received any unauthorized assistance on this assignment."

Your program should be written using good programming style and formatting, as discussed in class and throughout your textbook. For this project, style is considered to consist of:

- 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.
- adequate descriptive comments throughout (in addition to the comment mentioned above), to describe what your program is doing and how
- neat and proper indentation and formatting, including consistently and correctly aligning braces using any of the bracing styles illustrated in your textbook
- writing clear and readable code
- use of meaningful descriptive variable names

- use of symbolic constants for all "special" values which don't change and which are used in more than one place in your code
- avoiding disallowed C language features as discussed above
- make sure it runs with the sample input/output files given before submitting it, and make sure you test it on others that are different from those given.