CS 2123 Data Structures

Instructor Dr. Turgay Korkmaz

Homework 1
Due date: check BB learn.utsa.edu
!!!! NO LATE HOMEWORK WILL BE ACCEPTED !!!

Tips to Save Time:

- Read and understand the assignment before you begin
- Download the provided code and data files.
- Make sure you understand the provided code and run it before modifying it
- If needed, review your class notes about 2D arrays.

Background (from our textbook (ROBERTS))

In the last several years, a new logic puzzle called Sudoku has become quite popular throughout the world. In Sudoku, we start with a 9 x 9 grid of integers in which some of the cells have been filled in with digits between 1 and 9. The game is to fill in each of the empty spaces with a digit between 1 and 9 so that each digit appears exactly once in each row, each column, and each of the smaller 3 x 3 squares. Each Sudoku puzzle is carefully constructed so that there is only one solution. For example, given the **puzzle** shown on the left of the following diagram, the unique **solution** is shown on the right:

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

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

HW- Sudoku (high-level description):

Currently you may not have learned the algorithmic strategies that you need to solve Sudoku puzzles. But you can easily

1. declare a data structure to represent a Sudoku puzzle,
2. write some functions to read Sudoku puzzle from a file,
3. check if a proposed solution follows the Sudoku rules (no duplicating values in a row, column, or outlined 3 x 3 square (we may discuss Sudoku rules in the class).
4. also write a function that can list the possible values for each empty cell according to Sudoku rules.

Instead of asking you to write a program that can solve a Sudoku puzzle, **we ask you to write a program that can do some of the simple things mentioned above.** Some code is already provided for the first two tasks. You will focus on the last two tasks and implement the necessary functions.

HW- Sudoku (detailed description and necessary steps):
The provided code hw01.c is already doing the followings:

1. Declare a minimum size data structure to store the values of a Sudoku puzzle (e.g., 9x9 array of char because we will be storing numbers between 0 and 9)
2. Read Sudoku puzzle or Sudoku solution from a file whose name is given as a command line argument.
   - The given file would be formatted as follows: there is a number that indicates if this file contains a puzzle or a solution. If the that number is 1, then this file contains a Sudoku puzzle; if that number is 2, then the file contains a Sudoku solution to be checked.
   - The file than contains 9x9=81 values representing the values on Sudoku puzzle or solution.
     a) In the case of a puzzle (the first number is 1), the remaining 9x9=81 numbers in the file will be between 0 and 9. Zero is for empty cells, 1-9 are for other cells.
     b) In the case of a solution (the first number is 2), the remaining 9x9=81 numbers in the file will be between 1 and 9.

     For example, here is a sample puzzle saved in puzzle1.txt

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

     This test file and others are already posted on the class web page so you can download them and save them under the same directory where you will develop your hw1.c. You can create other input files too. We will grade your programs using other files as well...

   - The provided code can be compiled and executed as follows:

     ```
     fox03> gcc hw01.c -o hw01
     fox03> ./hw01 puzzle1.txt
     ```

Now you are asked to extend this program by writing the necessary functions to perform the following tasks:

3. If the given file contains a Sudoku solution (i.e., the first number in the file is 2), then your program should check if the solution is a valid one according to Sudoku rules. Basically, you need to check that each row, column, and 3x3 small square contains the numbers between 1 and 9 without any duplication. Accordingly, your program prints Yes or No.

4. If the given file contains a Sudoku puzzle (i.e., the first number in the file is 1), then your program should print possible values that can appear in each empty cell, which contains 0. In the case of the above puzzle1.txt data, the output would be as follows:

```
[0][0]: 3, 7
[0][1]: 3, 9
[0][4]: 1, 6, 9
.....
```

HINTS:

- In our sudoku program, we need to check and make sure that each row, column, and small square contains the numbers 1-9 with no duplicates. How can we do this? We can search each of numbers from 1 to 9 in each raw, column, and 3x3 squares. But this might be time consuming and also result in to much linces of code. Instead, consider using an array to save information about another piece of information. For example, we could make an array `int check[10]={0};` with each index corresponding to the numbers 1,2,3,4,5,6,7,8,9 (Ignore index 0, we don't need it).

  As we traverse a row, column, or small square, we can make a record each time we find a particular number by doing `check[number]++`.

  For example:
  Suppose we are given 1 3 2 5 4 6 7 9 8 as a row in a sudoku puzzle. Traversing this row while incrementing check[] array would result in check[1]=1, check[2]=1, check[3]=1, etc. Since we found one instance of each number, this row satisfies Sudoku solution criteria. However, if are given 1 1 3 3 5 5 6 6 7 as a row, then the resulting check[] array would be check[1]=2, check[2]=0, check[3]=2, etc. This clearly does not satisfy Sudoku solution criteria.

  The key to determining if a sudoku puzzle is correct is held in the values in your check[] array. You have to figure out how the numbers are set and what they mean. The key to finding possible values for an empty spot in a sudoku puzzle is also in the check array. Good luck!

- While it is easy to travers each row or column on a 2D array, it might be challenging to determine which 3x3 small square to consider. So here are some clues:

  Let x and y be the starting row and column indices of each 3x3 small squares. So when checking a given Sudoku solution (task 3), we can easily generate x and y by starting them with 0 and incrementing them by 3. Then we can relatively access the others in each 3x3 square. However, when performing task 4 we may need to know what are x and y for a given cell i and j. In this case, you can rely on integer division which does not produce a decimal. Basically, you can determine them as follows:

  ```
  x = i/3*3;
  y = j/3*3;
  ```

  Make sure you understand how/why these work!

What to return: !!!! NO LATE HOMEWORK WILL BE ACCEPTED !!!

DO ALL YOUR WORK UNDER **abc123-hw01** folder using your own abc123

1. Follow the problem solving methodology, and solve the problem(s). Then convert your solution(s) to a C program. Include /* comments */ so that we can understand your solution(s). You can name your program as hw01.c
2. Compile and run it. Copy/paste the result in an output file, say out1.txt. (or you can use **script** command in Linux as follows)

> script out1.txt # saves everything into out1.txt until you hit **ctrl-d**
> ./hw1 puzzle1.txt
…

3. Remove executables from **abc123-hw01** folder and then **Zip** that folder
4. Then go to BB Learn, and submit your **abc123-hw01.zip** as an attachment before the deadline. Make sure your zip file contains all your files!

/* Don't forget to include comments about the problem, yourself and each major step in your program! */

You must submit your work using Blackboard Learn and respect the following rules:

1. All assignments must be submitted as a **zip** file unless it is a single pdf file.
2. Assignments must include all source code.
3. Assignments must include an output.txt file which demonstrates the final test output run by the student.
4. If your assignment does not run/compile, the output.txt file should include an explanation of what was accomplished, what the error message was that prevented the student from finishing the assignment and what the student BELIEVES to be the error.
