CX 4010 / CSE 6010
Assignment 4
Logic Puzzle

Due Date: 11:59pm on Thursday, September 24
Submit a single zipfile as described herein to Canvas

In this assignment, you will work in small groups to develop a program to solve a particular type of puzzle loosely based on Tic-Tac-Toe. You will be given a square grid with a certain number of X's and O's marked in the grid. Your goal is to fill in the rest of the grid with X's and O's ensuring that three rules are followed:

- Each row and each column must have the same number of X's as the number of O's.
- No more than two X's or O's can appear in succession in any row or column.
- Each row is different from each other row and each column is different from each other column.

As an example, consider the following input grid.

	X				
O					
		O		X	
	X			X	O
					O
			O		

We can iteratively fill in empty squares in the grid; note that the solution is unique, but the steps taken to obtain the solution may not be. An example of a series of steps is given below.

X	X	O	X	O	O	Use rule #2 in columns 5 and 6.
O	O	X	X	O	X	Use rule #2 in rows 3 and 5.
O	X	O	O	X	X	Use rule #2 in row 3 again follows by rule #1 to complete row 3.
X	X	O	O	X	O	Use rule #2 in column 1 and rule #1 to complete column 2.
X	O	X	X	O	O	Use rule #2 in rows 1 and 6 and rule #1 to complete rows 2 and 4.
O	O	X	O	X	X	Finishing up….

You must determine and implement

- how to read in the starting grid,
- how to represent the grid within the program,
- how to complete the grid using the rules and starting information, and
- how to divide the work across your team.

Assume the input data file has the following format. The first line contains the single integer `grid_size`, which is both the number of rows and the number of columns. The following `grid_size` lines contain the data items, with each line representing the corresponding row with its `grid_size` values, separated by spaces. X's and O's appear as such, with unknown items given as ?. For example, the first row of the example above would appear as below.

```
? X ? ? ? ?
```

You are provided with three input files and the goal is for your program to work for all of them. In addition, you may wish to construct some additional test cases, in particular to ensure that each rule is being considered.

Note that you will need to figure out strategies for filling in the missing entries in the grid in accordance with the rules. In other words, there is no given approach that you should follow; instead, you will develop your own approach, which could include various search strategies or heuristics that are useful but possibly may not work in all future cases.

It is recommended that you run a final validation check at the conclusion of your program to ensure all rules have been followed and that the initial values provided remain unchanged.

Once you have computed your solution, your program should output the completed grid to the screen.

Team assignments will be available in Canvas. If you are assigned to a team of 3, you should come up with a division of labor such that two students are assigned the same sub-tasks, so that overall your team will have two implementations to solve the problem, each with one portion developed by a single student and with the other portion developed by one of the other two students.

Submission information

You should submit to Canvas **a single zipfile** that is named according to your Georgia Tech login—the part that precedes @gatech.edu in your GT email address. To receive full credit, your code must be well structured and documented so that it is easy to understand. Be sure to include comments that explain your code statements and structure.

The zipfile should include the following files:

(1) your code (all .c and .h files), with the main code that handles arguments, function calls, error handling, etc. named **main.c** and any helper functions (e.g., for reading the grid, solving, etc.) in files named **tictactoe.h/tictactoe.c**. If you are using linux or Mac OS, we recommend you use a **makefile** to compile and run your program, and you should include it if so.

(2) a **README** text file (not formatted in a word processor, for example) that includes the compiler and operating system you used for compiling and running your code along with instructions on how to compile and run your program.

(3) a series of slides composed in PowerPoint or similar software, saved either in PowerPoint or as a PDF and named **slides.pptx** or **slides.pdf**, in the following order:

- 1 slide: your name and a brief explanation of how you developed/structured your program. This should not be a recitation of material included in this assignment document but should focus on the main structural and functional elements of your program (e.g., the purpose of any loops you used, the purpose of any structs you may have used, the purpose of any functions you created, etc.).
- 1-2 slides: a description of your approach for generating a solution, how/why you developed it in that way, and any ideas you may have for future improvements.

- 1 slide: evidence of correct operation for the test cases and some ideas about how well you believe your code would work on general puzzles of this form (including your reasoning for your assessment).
- 1 slide: a description of the division of labor across the team for this assignment (who did what) with specific references to the code.