Assignment 1 – Arrays and Classes

You will submit a reflections document with every assignment. This is you chance to explain why you did what you did. Or why something you tried didn't work. You can explain what you did to test your program to make sure it works. You can also document features or bugs that inhibit the program in some way. Specifically discuss design, troubleshooting, and testing.

1. Write a program that allows two players to play a game of tic-tac-toe. You will create a game class. Do you need any data elements other than the board? What member functions do you need?

Use a two dimensional char array with 3 rows and 3 columns for the game board. Each element of the array should be initialized with an asterisk (*). The program should display the initial board. and then start a loop that does the following:

- Allow player 1 to select a location on the board for an X by entering a row and column number. Then display the board with an X replacing the * in the chosen location.
- If there is no winner yet and the board is not yet full, allow player 2 to select a location on the board for an O by entering a row and column number. Then display the board with an O replacing the * in the chosen location.

The loop should continue until a player has won or a tie has occurred, then display a message indicating who won, or reporting that a tie occurred.

- Player 1 wins when there are three Xs in a row, a column, or a diagonal on the game board.
- Player 2 wins when there are three Os in a row, a column, or a diagonal on the game board.
- A tie occurs when all of the locations on the board are full, but there is no winner.

Ask the users if they want to play again or if they want to exit the program.

Input Validation: Only allow legal moves to be entered. The row must be 1, 2, or 3. The column must be 1, 2, or 3. The (row, column) position entered must currently be empty (i.e., still have an asterisk in it).

Always make a copy of working code before making changes for other parts of the assignment! That means do it now. ☺ You'll have 3 separate programs to submit.

2. Modify your program to provide a computer opponent for the human user. You can call this AI if you want, but you are just required to provide an automated opponent. It does not need to be a smart opponent. There are not many options for a response. Remember that the board can be rotated and/or flipped. So you only need one response to an initial corner move, or an initial side move, or a starting move in the center. Or it can simply select an open space and mark it.

Give the user the choice to play first or second. So your automated opponent must have a starting move.
NOTE- Your automated player doesn't need to be smart. It doesn't need to win. It is only required to make a legal move. Nothing more.

3. Tic-tac-toe on a 3x3 board is boring. It is common knowledge that you can easily play for a draw, i.e. not losing. Modify your initial class to ask the user for the size of the grid. You did save your code, right? No automated player is required.

Prompt the user for a value N that will be the number of rows and columns for the grid. For simplicity in grading please limit N to the range 2 to 5. How do you change your input validation? You will use a dynamic array to hold the board. Remember to free your memory if the users are playing again. Must you make any other changes?

4. Create a makefile to compile each program. Remember to put all files into a zip archive and submit that zip file to TEACH before the deadline.

Grading:

- programming style and documentation (10%)
- reflections present **DESIGN** and testing (15%)

#1
- create, initialize, and correctly display the game board (5%)
- create and correctly make player one moves (10%)
- create and correctly make player two moves (10%)
- correctly determine which player won, or if it was a draw (10%)
- correct input validation (5%)

#2
- your automated opponent makes legal moves (10%)

#3
- correctly implement and use the dynamic array (10%)
- the victory conditions work correctly for each possible grid (10%)

#4
- Create a makefile (5%)