Project 3
Due June 20th at 11:59 PM

Summary - Write a C program that allows two people to play a game of Tic-Tac-Toe. The rules are as follows

- The game is played on a 3x3 grid
- There are two players, each player is assigned either an 'X' or 'O' token
- Players take turns placing their tokens into empty squares
- The first player to get 3 tokens in a row (horizontally, vertically, or diagonally) is the winner
- If no player gets 3 tokens in a row, the match is a draw

Required functionality:

1. All the rules listed above must be implemented.
2. If a player selects a square which is already occupied, the program should inform them the space is filled and require that they indicate a new space.
3. As soon as a player gets 3 tokens in a row, they should be declared the winner and the game should end. In other words, don't wait until all nine moves have been made to determine the winner.
4. The program should operate in a loop, asking the users if they would like to play again after the game is over
5. **Any accesses to arrays should be done with pointers and pointer arithmetic ONLY.**
    a. **There should be NO square brackets used when indexing into any arrays**
    b. **No counters should be used for looping over arrays. You should loop by using pointer arithmetic (see slide 21 of Week 4_ch12_2 powerpoint as example).**
6. **You must modularize your program by breaking it up into functions. Some example function names are**
    a. playGame
    b. printBoard
    c. checkForWinner

Extra credit functionality:

1. (2pts) If the users decides to play another game, the loser of the previous game should be the first to move. However, if a draw was reached, the first player to move should be the player who moved second in the previous game.

2. (8pts) Generalize the size of the board to be `NxN`. In order to win, a player must get N tokens in a row. **The board size should be set with a `#define N <value>` at the top of your file**. Your source files should also have the name `tictactoe_n.c`

```
[rippetoej@scln2 project3]$ ./a.out


 **** Tic-Tac-Toe ***
   |   |
---+---+---
   |   |
---+---+---
   |   |

Current move: Player 1
Enter row: 0
Enter column: 1
   | X |
---+---+---
   |   |
---+---+---
   |   |

Current move: Player 2
Enter row: 1
Enter column: 1
   | X |
---+---+---
   | O |
---+---+---
   |   |
```

… a few moves later ….

```
Current move: Player 2
Enter row: 1
Enter column: 2
 X | X |
---+---+---
 O | O | O
---+---+---
   |   | X

**** Player 2 wins! ****
Would you like to play another game? (y/n):
```

```
**** Player 2 wins! ****
Would you like to play another game? (y/n): y


 **** Tic-Tac-Toe ***
   |   |
---+---+---
   |   |
---+---+---
   |   |

Current move: Player 1
Enter row: 1
Enter column: 1
   |   |
---+---+---
   | X |
---+---+---
   |   |

Current move: Player 2
Enter row: 0
Enter column: 0
 O |   |
---+---+---
   | X |
---+---+---
   |   |
```

… a few moves later…

```
Current move: Player 1
Enter row: 2
Enter column: 0
 O |   | X
---+---+---
   | X |
---+---+---
 X |   | O

**** Player 1 wins! ****
Would you like to play another game? (y/n): n


**** Thank you for playing Tic-Tac-Toe! ****
[rippetoej@scln2 project3]$
```

Input/Output Requirements:

1. Input of player moves (i.e. choosing where to put a token) should follow the row and column format shown in the example above. The user should input them one at a time. Rows and columns should be zero ordered; for a standard 3x3 board, this means valid inputs are [0,1,2].

2. The outcome of the game should be printed exactly as shown in the example. The printf format strings for these are
   a. `"\n**** Player %d wins! ****\n"`
   b. `"\n**** The game has ended in a draw ****\n"`

   Valid player numbers are 1 and 2

3. When asking the user if they would like to play again, the only options should be `'y'` and `'n'`
4. You must display the current player to move, and do so exactly as shown in the example. The printf format string for this is

   `"\nCurrent move: Player %d\n"`

   This is important for validating an implementation of extra credit feature #1

5. If implementing extra credit feature #2
   a. You must define the board size at the top of your file with `#define N <value>`
   b. You must name your source file `tictactoe_n.c` so that we know you have implemented it

Before you submit:

1. **Make sure your program complies with all Input/Output requirements**
2. Make sure your program begins with a comment that begins with your name
3. Compile with `-Wall` flag. `-Wall` shows the warnings by the compiler. Be sure it compiles on the student cluster with no errors and no warnings.

   ```
   C99 -Wall tictactoe.c
   ```

4. Be sure your Unix source file is read & write protected. Change Unix file permission on Unix:

   ```
   chmod 600 tictactoe.c
   ```

5. Test your program thoroughly with multiple different scenarios using the supplied python script.

   ```
   python test.pyc <program name> test.txt
   ```

A successful run of the script should say PASSED.

6. Submit `ticTacToe.c` on Canvas.

Grading Details

1. A program that does not compile will result in a zero.
2. Programs which generate compiler warnings will lose 5%
3. Commenting and style worth 10%
4. Functionality worth 90%

Programming Style Guidelines

The major purpose of programming style guidelines is to make programs easy to read and understand. Good programming style helps make it possible for a person knowledgeable in the application area to quickly read a program and understand how it works.

1. Your program should begin with a comment that briefly summarizes what it does. This comment should also include your name.
2. Variable names and function names should be sufficiently descriptive that a knowledgeable reader can easily understand what the variable means and what the function does. If this is not possible, comments should be added to make the meaning clear.
3. Use consistent indentation to emphasize block structure.
4. Macro definitions (`#define`) should be used for defining symbolic names for numeric constants. For example: `#define PI 3.141592`
5. Use underscores or camel case notation to make compound names easier to read: `total_volume` or `totalVolume` is clearer than `totalvolume`.