ITP 165: Introduction to C++ Programming

Homework 6: Tic-Tac-Toe — Due: 11:59pm, March 4th, 2016

Goal

For this assignment, you will be making a game! You will be implementing the two-player game Tic-Tac-Toe. Part of the logic has been taken care of for you and provided to you in the form of an already started .cpp file. You will be filling in the missing code for one function and creating another from scratch. This assignment will give you practice with calling and creating functions that have input and return values.

Setup

- Create a new project in Visual Studio or Xcode called **hw06** and add the existing file named **hw06.cpp** to your project. The file is included with your homework write-up on Blackboard.
- Your **hw06.cpp** file must begin with the following (replace the name and email with your actual information):

```
// Name
// ITP 165, Spring 2016
// Homework 06
// USC email
```

Part 1: PrintBoard function

- To begin, you will be making a function to display the game board. Be sure to include the function header comments to document the name, purpose, parameters, and return of the function.
- This function will be called PrintBoard and will return nothing. It has one input parameter of a character array that is meant to hold the game board.
    - This function will display a 3x3 grid and outputs every element of the character array in the corresponding grid spot.
    - You may assume we will always have a size 9 array (the typical number of spaces on a Tic-Tac-Toe game board).
- Now you need to test your function in main. Create a `char` array of size 9 initialized with the **characters** 1-9.
- Next, call the newly implemented PrintBoard function with the `char` array from main as an input parameter to display the game board and test your function.
- Your output in Part 1 should look something like:

```
 1 | 2 | 3
-----------
 4 | 5 | 6
-----------
 7 | 8 | 9
Press any key to continue...
```

Part 2: IsValidMove function

- Next, you will fill in the function logic for the IsValidMove function above main. Be sure to look at the function signature as well as the header comments for further details on the function. Notice the input parameters are the current game board, as well as the **index** of the next attempted move by the player.
    - This function will return false if the index is outside of the valid range of indices for your character array.
    - This function will also return false if the character at the inputted index is already taken by an 'X' or an 'O'
    - This function will only return true if the player may make a move at the given index.
- Now it is time to test the function in main. First create a char to store the "current" player. Typical Tic-Tac-Toe games start with player 'X'.
- Next, setup a loop to ask the current player for input as long as they enter INvalid input.
    - Store the user's input into a variable and pass this as a parameter to your new function. Since our function returns a bool, it *may* help to store the return value from this function into a bool to use for the condition of your loop.
        - Notice our board displays the numbers 1 through 9, to signify to the user where there is a free place to move for their turn, but that arrays start at **index** 0.
    - To check if your function works for all cases, TEMPORARILY change your character array to include an 'X' or 'O'. Be sure to fix this change after Part 2 is complete.
- Your output after completing Part 2 should look similar to this: (User input in **red**)

```
 1 | 2 | 3
-----------
 4 | X | 6
-----------
 7 | 8 | 9
Pick a spot: 0
Choice outside valid range.
Try again.
Pick a spot: 5
Place already taken.
Try again.
Pick a spot: 1
Press any key to continue . . .
```

Part 3: Output player's move

- At this point in the game, the current player has entered valid input on where to make their move, so you must update the game board.
- Replace the corresponding spot on the game board with the current player's character.
    - Again, remember our game board is numbered 1 to 9, but that arrays start at index 0.
- After you have stored the player's move, call the PrintBoard function again, with the updated game board as a parameter, to show the player the move they made.
- After a player has made a turn, make sure you switch turns to the next player (if 'X' made a move, make sure 'O' will go next).

- Output after Part 3 should look similar to this: (user input in **red**)

```
 1 | 2 | 3
-----------
 4 | 5 | 6
-----------
 7 | 8 | 9
Player X, pick a spot: 0
Choice outside valid range.
Try again.
Player X, pick a spot: 1
 X | 2 | 3
-----------
 4 | 5 | 6
-----------
 7 | 8 | 9
Press any key to continue . . .
```

Part 4: Check if there is a winner

- You have one complete function provided to you that handles the logic of checking which player (if any) has won. This function is called WinnerIfAny, has a parameter of the game board array, and an integer representing the game's turn counter. The function returns a `char` signifying the "current" winner. Either `'X'` or `'O'` or `'N'` or `'S'` is returned, depending if X has won, O has won, No one has won, or a Stalemate was reached.
    - Please refer to the function code and comments for further understanding on how this function works.
- Store the return value from the WinnerIfAny function into a `char`.
- Check who is the current winner and output a case-specific message to the user for each return.
- To test if you have setup the function usage correctly, you will once again have to change the character array initialization **temporarily** (and possibly your turn counter as well).
- Example test output for Part 4 could look similar to this: (user input in **red**)

```
 1 | 2 | 3
-----------
 4 | 5 | 6
-----------
 7 | 8 | 9
Player X, pick a
spot: 5
 1 | 2 | 3
-----------
 4 | X | 6
-----------
 7 | 8 | 9
No one has won yet...
```

```
 X | X | 3
-----------
 4 | 5 | 6
-----------
 7 | 8 | 9
Player X, pick a
spot: 3
 X | X | X
-----------
 4 | 5 | 6
-----------
 7 | 8 | 9
Player X is the
winner!
```

```
 O | X | O
-----------
 X | X | O
-----------
 X | O | 9
Player X, pick a
spot: 9
 O | X | O
-----------
 X | X | O
-----------
 X | O | X
Stalemate reached!
```

Part 5: Loop to allow for multiple turns

- At this point, the first player makes a move and then our game ends. We now want to make sure that we loop over Parts 2 to 4 in main until the game has ended.
- We will be using the `char` variable that tracks the winner of the game for the condition of our loop. Please implement the logic that our loop will continue as long as the winner is No one.
    - IMPORTANT NOTE: If you want your logic to work correctly for each turn, be sure to RESET/UPDATE your variables at the end of your loop. Also, be aware of where you declare your variables. If you need a variable for multiple iterations of a loop – you must declare it **outside** the loop's scope.
- Output after Part 5 should look similar to this: (user input in **red**)

```
 1 | 2 | 3
-----------
 4 | 5 | 6
-----------
 7 | 8 | 9
Player X, pick a spot: 1
 X | 2 | 3
-----------
 4 | 5 | 6
-----------
 7 | 8 | 9
No one has won yet...
Player O, pick a spot: 2
 X | O | 3
-----------
 4 | 5 | 6
-----------
 7 | 8 | 9
No one has won yet...
Player X, pick a spot: 3
 X | O | X
-----------
 4 | 5 | 6
-----------
 7 | 8 | 9
No one has won yet...
Player O, pick a spot: 5
 X | O | X
-----------
 4 | O | 6
-----------
 7 | 8 | 9
No one has won yet...
Player X, pick a spot: 4
```

```
 X | O | X
-----------
 X | O | 6
-----------
 7 | 8 | 9
No one has won yet...
Player O, pick a spot: 8
 X | O | X
-----------
 X | O | 6
-----------
 7 | O | 9
Player O is the winner!
Press any key to continue . . .
```

A Note on Style

Be sure to comment your code and especially your functions. Each function is expected to have the required four-line comment heading.

As we discussed in lecture, it is extremely important that your code is properly indented as it greatly adds to readability. Because of this, if you submit a code file that is not reasonably indented, you will have points deducted.

Likewise, you will lose points if your variable names are not meaningful. Make sure you use variable names that correspond to what you are actually storing in the variables.

Full Sample Output

```
 1 | 2 | 3
-----------
 4 | 5 | 6
-----------
 7 | 8 | 9
Player X, pick a spot: 1
 X | 2 | 3
-----------
 4 | 5 | 6
-----------
 7 | 8 | 9
No one has won yet...
Player O, pick a spot: 2
 X | O | 3
-----------
 4 | 5 | 6
-----------
 7 | 8 | 9
No one has won yet...
Player X, pick a spot: 3
```

```
 X | O | X
-----------
 4 | 5 | 6
-----------
 7 | 8 | 9
No one has won yet...
Player O, pick a spot: 5
 X | O | X
-----------
 4 | O | 6
-----------
 7 | 8 | 9
No one has won yet...
Player X, pick a spot: 4
 X | O | X
-----------
 X | O | 6
-----------
 7 | 8 | 9
No one has won yet...
Player O, pick a spot: 7
 X | O | X
-----------
 X | O | 6
-----------
 O | 8 | 9
No one has won yet...
Player X, pick a spot: 8
 X | O | X
-----------
 X | O | 6
-----------
 O | X | 9
No one has won yet...
Player O, pick a spot: 6
 X | O | X
-----------
 X | O | O
-----------
 O | X | 9
No one has won yet...
Player X, pick a spot: 9
 X | O | X
-----------
 X | O | O
-----------
 O | X | X
Stalemate reached!
Press any key to continue . . .
```

Deliverables

A compressed folder containing **hw06.cpp**. This can be done by:

a. WINDOWS:
   i. Select your file
   ii. Right click
   iii. Send to ->
   iv. Compressed (zipped) folder
   v. Make sure this compressed folder is named **hw06.zip**
   vi. Submit this compressed folder through Blackboard

b. OSX:
   i. Select your file
   ii. Right click
   iii. Compress **"hw06.cpp"**
   iv. Make sure this compressed folder is named **hw06.cpp.zip**
   v. Submit this compressed folder through Blackboard

Grading

Item	Points
Part 1: PrintBoard function	5
Part 2: IsValidMove function	10
Part 3: Output player's move	5
Part 4: Check if there is a winner	10
Part 5: Loop to allow for multiple turns	5
Total*	**35**

* Points will be deducted for poor code style, or improper submission.