CS 10 - Assignment 9

Collaboration Policy

Collaboration between students on programming assignments is **NOT** allowed under any circumstances - you may not even discuss your work with others; you may **NOT** get coding assistance, or copy code, from **ANY** outside source *(books, web sites, current or past students, your own previous submissions if you are repeating the course, etc.)*. We test every submission for copying and when we find it we treat it as flagrant academic dishonesty on the part of ***all*** parties involved, regardless of their roles as provider or consumer. The penalty is generally instant failure in the course, in addition to the usual sanctions applied by Student Judicial Affairs.

As you can see from the submission header below, we ask you to essentially take an oath that the code you submit is ***entirely your own work*****.**

At the same time, we certainly understand very well that you will frequently need help completing your programming assignment, so we provide channels where you may get that help in a way which will strengthen your programming skills: instructor and TA office hours, and the discussion forums where you can ask all the questions you want about the assignment, and even help others with the understanding that you have gained *(but not, of course, with any actual code)*.

Assignment Submission Instructions

Your assignment must be submitted as a ***simple text file*** named ***main.cpp***
Files in ANY other format (MS Word document, etc.) or with ANY other name (main, main.doc, main.txt, etc.) will ***not*** be graded.

Submit your work via the appropriate iLearn assignment link, making sure you use the correct link.

We strongly recommend that:

1. you submit at least *6 hours* before the deadline, even if you haven't completed the program - you can re-submit as often as you like, and we will only grade the final submission
2. once you have submitted your final attempt, go back and download it back to your system: then run it just to make absolutely sure that it really is the file you intended to submit!

You are budding professionals, and are expected to take full responsibility for abiding by the requirements of a contract.

The ***only reason we will ever accept*** for a missed deadline is if the system administrators inform me that either the iLearn system or the CS department servers were off-line for a significant period around the time of the deadline *(In which case we will probably have notified you of alternative procedures)*.

Remember to include the following header information at the top of your program
(DO NOT REMOVE THE // at the beginning of each line, these tell the compiler to ignore these lines)

```
// Course: CS 10 <quarter & year>
//
// First Name:
// Last Name:
// Course username: <enter the username you use to login in the lab>
// Email address: <enter your cs or UCR student email address here>
//
// Lecture Section: <e.g. 001>
// Lab Section: <e.g. 021>
// TA:
//
// Assignment: <assn1, hw2, lab3, etc.>
//
// I hereby certify that the code in this file
// is ENTIRELY my own original work.
//
// ================================================================
```

NOTE: This header MUST appear at the top of EVERY file submitted as part of your assignment (don't forget to fill in *your* details and remove the <> brackets!!).

Copy & paste this header into your file then update personal details.

(make sure spaces are copied too!)

Assignment Specifications

You are to implement a console version of the game tic-tac-toe. For this assignment, we are providing a `main.cpp` file that contains skeleton code that you must complete. We also provide complete functions for you to utilize. You are **not** allowed to change the provided functions and you are **not** allowed to changed the headers of the provided function stubs. You **must** begin with the provided skeleton code.

For the functions you must implement, we have provided only a stub. A stub is a function definition that compiles, but does not yet implement the complete specifications for that function. As you are developing the program, you should implement just the function needed for the part you are working on.

Implementation Strategies

- We provide some variables and two global **constants** for you to utilize.
- We provide string literals for winning or tie game output in comments with provided file
- We have also provided comments to help you develop the necessary algorithm for 2 users playing the game of tic-tac-toe on a computer. Use these comments along with the function descriptions below to help you develop your program. Remove the TODO part when you have completed that step.
- DO NOT try to implement the entire game at once. Instead, implement one behavior at a time, only developing a particular function when you need it.
- We highly recommend you unit test the function you are currently developing before trying to use it within the overall tic-tac-toe program. The small amount of time it takes to thoroughly unit test a single function quite often saves you a tremendous amount of time debugging the entire program.

Functions

```
/**
 *  @brief Converts character position to appropriate vector index for board
 *
 *      For example, 'a' is converted to 0, 'd' is converted to 3, etc.
 *
 *  @param position the character to be converted to a vector index
 *  @return the integer index in the vector, should be 0 to (vector size - 1)
 */
int convertPosition(char position)


/**
 *  @brief Predicate function to determine if a spot in board is available.
 *  @param board the current tic-tac-toe board
 *  @param position is an index into the vector to check if available
 *  @return true if position's state is available (not marked) AND is in bounds
 */
bool validPlacement(const vector <char> &board, int position)


/**
 *  @brief Acquires a play from the user as to where to put her mark
 *
 *      Utilizes convertPosition and validPlacement functions to convert the
 *      user input and then determine if the converted input is a valid play.
 *
 *  @param board the current tic-tac-toe board
 *  @return an integer index in board vector of a chosen valid board spot
 */
int getPlay(const vector <char> &board)


/**
 *  @brief Predicate function to determine if the game has been won
 *
 *  Winning conditions in tic-tac-toe require three marks from same
 *  player in a single row, column or diagonal.
 *
 *  @param board the current tic-tac-toe board
 *  @return true if the game has been won, false otherwise
 */
bool gameWon(const vector <char> &board)


/**
 *  @brief Predicate function to determine if the board is full
 *  @param board the current tic-tac-toe board
 *  @return true iff the board is full (no cell is available)
 */
bool boardFull(const vector <char> &board)
```

Example Run

An example of a tie game run is included at the end of this specification. You should utilize this example as well as others that you invent yourself (to test both players winning).

Testing Your Solution

Testing this program will involve unit testing of each function as well as full program output testing. Output and input files will be provided on iLearn for your benefit. Please download the files to test your program. These tests do not represent all the possibilities, we expect you to to test other possibilities on your own as well as implementing the functions as defined in the specification.

You should not do the following until you have your program complete. If your program has an infinite loop, you will generate excessively large files and experience many problems.

Recall the `diff` command is utilized to compare two files, it allows comparison of a solution's output and your program's output.

```
diff -Bbiau my.out solution.out
```

Step 1 - *Compile your program*: `[user@well]$ g++ main.cpp`

Step 2 - *Execute With Input Redirection*: `[user@well]$ ./a.out < input.txt`

DO NOT skip Step 2, this will assure no infinite loops exist.

If program does not execute to completion, fix your program and go to Step 1.

Step 3 - *Use Input & Output Redirection*: `[user@well]$ ./a.out < input.txt > my.out`

As you can see, output redirection is similar except the right angle bracket is used and points to the file where output should go.

Step 4 - *Viewing differences*: `[user@well]$ diff -Bbaiu mine.out solution.out | less`

If there are lots of differences or clear screen sequences the output can be hard to read in the terminal. In that case we send the output of the command to a text reading program such as `less`. This will allow you to use the arrow keys to navigate up and down the output.

Press 'q' to quit the `less` program and return to the terminal line.

```
--- mine.out         2012-11-10 17:22:41.582953000 -0800
+++ solution.out     2012-11-10 17:22:29.870525000 -0800
@@ -19,7 +19,7 @@
Phrase: he---
-uessed so far: he
+Guessed so far: he
Wrong guesses left: 7
Enter a guess:
```

If there are differences between the two files they will be shown. The first two lines show a key for reading the output. The character preceding the file denotes the character that will precede lines in differences for that file. In this example the Lines preceded by a '-' correspond to the `mine.out` file, similarly the '+' belongs to solution output. If a line is not preceded by a key character then it exists in both.

Step 5 - *Compare the files with `diff`*: `[user@well]$ diff -Bbaiu my.out solution.out`

If no output is produced by the `diff` the files match.

Marking Guidelines

- Make sure your code compiles.
- Make sure your output matches the format of the examples: spacing, spelling, etc.
- We utilize the `diff` command to grade in some portions
- We will unit test your functions so make sure they meet the specification as they will be tested individually (separated from your program)

Basic Style Guidelines

- **indentation** - follow proper 4 space indentation as you go into loops or branches
- **spacing** - blank lines should be used to separate logic blocks
- **no line wraps** - no line of code should have more than 80 characters
- "**meaningful**" variable **names**
- **comments** - each logical block of code should have a brief explanatory comment

Tie Game Test Case

These are here for quick reference.

Please use the provided files to avoid copy and paste errors.

Input:

```
a
e
e
b
c
g
d
f
h
i
```

Output:

```
ESCc

  a  |  b  |  c
-----|-----|-----
  d  |  e  |  f
-----|-----|-----
  g  |  h  |  i

Please choose a position:

 ESCc

  X  |  b  |  c
-----|-----|-----
  d  |  e  |  f
-----|-----|-----
  g  |  h  |  i

Please choose a position:

 ESCc

  X  |  b  |  c
-----|-----|-----
  d  |  O  |  f
-----|-----|-----
  g  |  h  |  i

Please choose a position:
Please choose a position:

 ESCc

  X  |  X  |  c
-----|-----|-----
  d  |  O  |  f
-----|-----|-----
  g  |  h  |  i

Please choose a position:

 ESCc

  X  |  X  |  O
-----|-----|-----
  d  |  O  |  f
-----|-----|-----
  g  |  h  |  i

Please choose a position:

 ESCc

  X  |  X  |  O
-----|-----|-----
  d  |  O  |  f
-----|-----|-----
  X  |  h  |  i

Please choose a position:

 ESCc

  X  |  X  |  O
-----|-----|-----
```

```
    O  |  O  |  f
  -----|-----|-----
    X  |  h  |  i

Please choose a position:

 ESCc

  X  |  X  |  O
-----|-----|-----
  O  |  O  |  X
-----|-----|-----
  X  |  h  |  i

Please choose a position:

 ESCc

  X  |  X  |  O
-----|-----|-----
  O  |  O  |  X
-----|-----|-----
  X  |  O  |  i

Please choose a position:

 ESCc

  X  |  X  |  O
-----|-----|-----
  O  |  O  |  X
-----|-----|-----
  X  |  O  |  X

No one wins
```