CS 10 - Assignment 7

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 and **click on the attach button**

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.

Assignment Specifications

You must implement a simplified version of ***Hangman***. In our simplified version, the player gets **7 incorrect guesses** to guess all of the characters within a phrase. **You are required to implement the specified functions.**

Hangman Algorithm

- Get phrase to be solved (using `getline` to allow for spaces in the phrase to be guessed)
- Clear output window so phrase is not seen (clear window using clearScreen())
- Setup unsolved phrase so game has another string with dashes in place of characters (call `setupUnsolvedPhrase` function)
- Output the unsolved phrase
- Play Game until phrase is completely guessed or no wrong guesses left, 7 wrong allowed
  - Get letter guess (call `getNewGuess` function)
    (All user input after initial phrase should **ONLY** occur in the `getNewGuess` function)
  - Update string containing all characters guessed so far
  - Take action on the guess
    - Correct: update unsolved phrase using the `updateUnsolvedPhrase` function
    - Incorrect: update the wrong guess count
  - Clear the screen
  - Output
    - updated unsolved phrase - phrase with dashes for unguessed letters
    - list of characters guessed so far
    - number of wrong guesses left
- Output game over message (`You lost!` or `Congratulations!!`)

Header File

We have supplied a header file to be utilized with this assignment. In the file is a function that will clear the terminal window. You will call this function to clear the screen.

Additional Requirements and Hints

- **All** user input statements should be followed by an output statement to create a blank line.
- **All inputs** after the initial phrase should **occur only in the** `getNewGuess` function
- All input tests will use lower case characters only
- The player gets **7** wrong guesses.
- No input or output statements exist in `setupUnsolvedPhrase` or `updateUnsolvedPhrase`
- You should be utilizing a single `puzzle` variable and assigning into it the return value from `setupUnsolvedPhrase` and `updateUnsolvedPhrase`.
- **Use `at()`**: This member function can go on either side of an `'='` sign. To either get and use the value at a specified position (right side) or to assign a new value into that specific position (left side). `at()` throws exceptions when you access an invalid value, this will save you time hunting down bugs, where as using `[]` syntax may or may not throw an error.
- **`bool isalpha(char)`**: In order to check to see if a guess is proper alphabetic character you can use this built in function that takes a single character as an argument.
- ***Note:*** The functions `setupUnsolvedPhrase` and `updateUnsolvedPhrase` act in opposition to each other. This means that should have slight logical differences but are coded similarly.

Functions

The following 3 functions are **required**.

You must define, implement and correctly use these 3 functions with the exact specifications described below. In fact, just copy-and-paste these comments and function headers into your `main.cpp` file.

```
/**
   @brief Puts dashes in place of alphabetic characters in phrase.
   @param phrase the phrase to be solved
   @return the phrase with all alphabetic characters converted to dashes
*/
string setupUnsolvedPhrase(string phrase)


/**
   @brief Replaces the dashes with the guessed character.
   @param phrase the phrase to be solved
   @param unsolved the phrase with dashes for all unsolved characters
   @param guess the char containing the current guessed character
   @return the new unsolved string with dashes replaced by new guess
*/
string updateUnsolvedPhrase(string phrase, string unsolved, char guess)


/**
   @brief Gets valid guess as input.

   A guess is taken as input as a character. It is valid if
   1) it is an alphabetic character and
   2) the character has not already been guessed

   @param prevGuesses the string containing all characters guessed so far
   @return a valid guess and only a valid guess as a character
*/
char getNewGuess(string prevGuesses)
```

Testing Your Solution

Since the assignment contains output within loops, lengthier **multi-word** phrases can produce lengthy output. **Start** by testing something small, a single word where all your guesses are correct. Then move to some incorrect guesses and transition to all incorrect guesses. Then move to testing multiple word phrases. Be sure to test both valid and invalid inputs.

Reading Output Files:

ESCc within output represents the clear screen sequence

Introduction to `diff`

The `diff` command allows line by line comparison of multiple files. We utilize this to compare output of your program to output of a solution program. To do this we must generate an output file from your program to compare to the output provided.

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

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 > mine.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 - *Compare the files with `diff`*: `[user@well]$ diff -Bbaiu mine.out solution.out`

```
--- 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.

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

Step 5 - *Lots of differences*: `[user@well]$ diff -Bbaiu mine.out solution.out | less`

Always try Step 4 before Step 5. If there are lots of differences it 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 program and return to the terminal line.

<u>Example Input & Output Files</u>

These are provided at the end of the specification due to their length.

How We Test

We will test your program with both unit tests of your functions as well as output differences. This means we will attempt to test each of your functions individually, all three function specifications as well as main. After unit testing we will compare ful program output testing. This should allow partial credit based on proper function implementation as specified.

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

Input File (simple.in)

```
hello
h
e
l
d
e
o
```

Input File (simple-wrong.in)

```
hello
p
q
r
s
t
u
v
```

Output File (simple.out)

```
Enter phrase:

ESCc

Phrase: -----

Enter a guess:

ESCc

Phrase: h----

Guessed so far: h
Wrong guesses left: 7

Enter a guess:

ESCc

Phrase: he---

Guessed so far: he
Wrong guesses left: 7

Enter a guess:

ESCc

Phrase: hell-

Guessed so far: hel
Wrong guesses left: 7

Enter a guess:

ESCc

Phrase: hell-

Guessed so far: held
Wrong guesses left: 6

Enter a guess:
Invalid guess! Please re-enter a guess:

ESCc

Phrase: hello

Guessed so far: heldo
Wrong guesses left: 6

Congratulations!!
```

Output File (simple-wrong.out)

```
Enter phrase:

ESCc

Phrase: -----

Enter a guess:

ESCc

Phrase: -----

Guessed so far: p
Wrong guesses left: 6

Enter a guess:

ESCc

Phrase: -----

Guessed so far: pq
Wrong guesses left: 5

Enter a guess:

ESCc

Phrase: -----

Guessed so far: pqr
Wrong guesses left: 4

Enter a guess:

ESCc

Phrase: -----

Guessed so far: pqrs
Wrong guesses left: 3

Enter a guess:

ESCc

Phrase: -----

Guessed so far: pqrst
Wrong guesses left: 2

Enter a guess:

ESCc

Phrase: -----

Guessed so far: pqrstu
Wrong guesses left: 1

Enter a guess:

ESCc

Phrase: -----

Guessed so far: pqrstuv
Wrong guesses left: 0

You lost!
```