**CS 10 - Assignment 6: Hangman**

**Collaboration Policy**

We encourage collaboration on various activities such as lab, codelab, and textbook exercises. However, **no collaboration between students is allowed on the programming assignments**. Please be sure to read and understand our full policy at: [Full Collaboration Policy](https://docs.google.com/document/d/1WyzL3qvKLrC1UCRf178b_wYWQmEZlhDObFNFb79U63I/edit?usp=sharing)

**Submission Instructions**

Submit to [R’Sub](https://galah.cs.ucr.edu) testing, feedback and grading.

**Assignment Specifications**

The game called [Hangman](http://en.wikipedia.org/wiki/Hangman_(game)) is a spelling game: one player sets up a word or phrase, represented by a row of dashes. If the guessing player suggests a letter which occurs in the word, the other player writes it in all its correct positions. If the suggested letter does not occur in the word, the other player draws one element of the hanged man [stick figure](http://en.wikipedia.org/wiki/Stick_figure) as a [tally mark](http://en.wikipedia.org/wiki/Tally_mark). The game is over when:

* The guessing player completes the word, or guesses the whole word correctly
* The other player completes the diagram.

You must implement a simplified version of hangman in which the player gets **7 incorrect guesses** (corresponding to drawing the “hanged man”).  
**You are required to implement the specified functions and utilize any provided functions.**

**Header File**

We have supplied a header file called assn.h to be utilized with this assignment: it provides the function clearScreen() that will clear the terminal window, to be invoked at specified spots within the algorithm. To acquire the file, either run git pull on cloud9 within your terminal when inside the cs010\_assignments directory. Or download the file and upload the file into your hangman assignment directory. To utilize the file, add the file to your includes:

**#include "assn.h"** *(Note the quotes, unlike the* <> *used for including C++ libraries)*

**Hangman Algorithm**

* Get phrase to be solved from the user (using getline instead of the input operator >> to allow for the presence of spaces in the phrase)
* Clear output window so phrase is not seen, using the provided function clearScreen()
* Setup unsolved phrase so game has a separate string with dashes in place of characters

(call setupUnsolved function)

* Display the unsolved phrase
* Play Game until phrase is completely guessed, or all 7 incorrect guesses are used up
  + Get a valid letter guess from user
    - call getGuess function - see below for meaning of “valid guess”
    - After initial phrase has been set, user input should **ONLY** occur in getGuess
  + Update a string consisting of *all characters guessed so far*
  + Take action on the guess:
    - Correct: update unsolved phrase using the updateUnsolved function
    - Incorrect: increment the wrong guess count
  + Clear the screen again, using clearScreen()
  + Output game status:
    - updated unsolved phrase - phrase with dashes for still-unguessed letters
    - list of characters guessed so far
    - number of wrong guesses left
* Output game over message (You lost! or Congratulations!!)

**Additional Requirements and Hints**

* **Implement each function** and get feedback/credit for each **before doing main game**
* **All** user input statements should be followed by outputting a blank line.
* **All user inputs and corresponding prompts** after initial phrase **occur only in body of** getGuess
* All phrases and valid guesses will use lowercase alphabetic characters   
  *(i.e. in our testing, we will only use lower case phrases; and we will enter only lower case characters as* ***valid*** *guesses; we will, of course, input non-alpha characters as* ***invalid*** *guesses)*
* The player gets **7** wrong guesses
* Utilize a single puzzle variable to hold the current state of the game, updating the value to the return value from setupUnsolved initially, and updateUnsolved subsequently.
* **Use the string member function at()**: Remember, at() can be used on either side of the assignment operator (=), to either acquire the value at a specified position of the string, or to set a new value into the specified position of the string.
* The function at() “throws an exception” when you attempt to access a non-existent location.
* **bool isalpha(char)**: In order to check if a guess is valid - i.e. if it is an alphabetic character - you can use this built-in predicate function that takes a single character as an argument and reteruns true if the character is an alphabetic character. Requires: #include <cctype>
* The functions setupUnsolved and updateUnsolved are “complementary” to each other. One replaces letters with dashes, the other replaces those dashes with letters.

**Functions**

The following 3 functions are **required, exactly as declared**.  
You may, if you wish, define other functions in addition to these.

You must define, implement and correctly invoke these 3 functions exactly as declared here.  
You may wish to copy & paste the following into your source code, and comment out the function declarations you are not yet ready to implement.

* When defining the functions, don't forget to remove the semicolon after the function header.
* Each function only performs the defined task, nothing more, nothing less. If a task is not defined in the function comments, then the task's implementation must be in another function or inside the main function's algorithm.

/// @brief Puts dashes in place of alphabetic characters in the phrase.

/// @param phrase the phrase to be solved

/// @return the phrase with all alphabetic characters converted to dashes

string setupUnsolved(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 updateUnsolved(string phrase, string unsolved, char guess);

/// @brief Gets input until a valid input occurs, returns the valid input.

///

/// A guess is taken as input as a character. Do not use getline.

/// The guess is valid if:

/// 1) the guess 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 getGuess(string prevGuesses);

**Example Run** (typed input is **bold & underlined**, c represents a clearScreen invocation [follow the algorithm])

Enter phrase: **fizz, bizz, buzz.**

c

Phrase: ----, ----, ----.

Enter a guess: **f**

c

Phrase: f---, ----, ----.

Guessed so far: f

Wrong guesses left: 7

Enter a guess: **z**

c

Phrase: f-zz, --zz, --zz.

Guessed so far: fz

Wrong guesses left: 7

Enter a guess: **i**

c

Phrase: fizz, -izz, --zz.

Guessed so far: fzi

Wrong guesses left: 7

Enter a guess: **z**

Invalid guess! Please re-enter a guess: **b**

c

Phrase: fizz, bizz, b-zz.

Guessed so far: fzib

Wrong guesses left: 7

Enter a guess: **;**

Invalid guess! Please re-enter a guess: **i**

Invalid guess! Please re-enter a guess: **.**

Invalid guess! Please re-enter a guess: **9**

Invalid guess! Please re-enter a guess: **b**

Invalid guess! Please re-enter a guess: **u**

c

Phrase: fizz, bizz, buzz.

Guessed so far: fzibu

Wrong guesses left: 7

Congratulations!!