COMP 1602, Computer Programming 2
Assignment #3
Date Due: March 10, 2019 @11.55 pm

Background

This assignment is based on the game of Hangman played between two persons. In the game of Hangman, one person (Player 1) thinks of a word while the other person (Player 2) tries to guess the word by suggesting letters. If Player 2 guesses a correct letter, it is placed in its correct position/s in the word. If Player 2 guesses a letter that is not in the word, Player 1 updates the Hangman diagram by drawing one more stage.

The game is over when:

- Player 2 wins by completing the word correctly, or
- Player 1 wins by completing the Hangman diagram (Player 2 made 6 incorrect guesses).

Requirements

You are required to write a computer program to simulate the playing of several games of Hangman. The computer plays the role of Player 1 and the human user plays the role of Player 2.

Features to the Implemented in the Game

1. The user must input his/her name before playing.

2. Read the words from the text file, `words.txt`, and store the words in an array, *words*.
   - Each line in the file has two strings – the word to be guessed and the category of the word (similar to the word categories in the show "Wheel of Fortune").
   - For each word, your program should keep track of whether the word was already used in a game.
   - All the words in the game must be in lowercase, regardless of how they are stored in the file.

3. Your program should ask the user to choose the category of the word. It should then find all the words in this category that haven't been used before (from the *words* array) and randomly select a word from this set to play the game.
   - Just as in "Wheel of Fortune", your program should insert the letters 'R', 'S', 'T', 'L', 'N', and 'E' in the word to be guessed before starting the game (if they are present).
   - Your program should display the initial hangman diagram and a template of the word to be guessed. The underscore character should be used for each letter not yet guessed.

4. Your program should repeatedly prompt the user to guess a letter. If the letter is present in the word, it is displayed along with the letters already guessed. If the letter is not in the word, the user loses a life and the hangman diagram is updated.
   - The user has a maximum of 6 lives. If the six lives are used up before the user guesses the word, then the game is over. When a game is over, the user's name must be displayed together with his/her current score (see below).

- While playing a game, your program must keep track of all the letters entered by the user so far. The set of letters must be displayed each time the user is requested to enter a letter. At the end of a game, the set must be cleared. If the user guesses a letter that has already been guessed or is present in the string '"RSTLNE", he/she must be informed about this but the guess is not counted.
- If the user wishes to guess the entire word rather than a letter, he/she should enter the '#' character when prompted for a letter. When the '#' is entered, the user should be prompted to enter the word. The word entered is compared to the word to be guessed. If it is not the same and the user has less than three lives remaining, the user loses and the game ends. Otherwise, the user loses two of his/her remaining lives.
- If the user guesses all the letters in a word, he/she gains 2 points. If there are two or more letters still to be guessed and the user guesses the entire word correctly, he/she gains 4 points. If the user loses a game, he/she loses 2 points. Your program must keep a running total of the player's score. The amount of lives remaining for the current game along with the score must be displayed on the screen.

5. After the user has either guessed a word correctly or lost his/her six lives, he/she should be prompted to play another game or to stop playing. The option to play another game should repeat the game play from Step 3. The user's current score should also be displayed.

6. If a word has been used in a previous game, it should not be reused.

Programming Guidelines

A *Word* struct must be used to store each word that could be used in one of the games. The *Word* struct is defined the file, `Assignment3.cpp`.

Your program should be decomposed into a set of functions. This section describes the functions that your program should have. The functions with a status of "Given" are already written for you in the file, `Assignment3.cpp`.

Function	**Status**	**Description**
bool isLetter (char c)	Given	This function returns *true* if *c* is a letter and *false*, otherwise.
int readCharacters(char data[])	Given	This function reads all the characters from the `words.txt` file to the *data* array.
int getWord (char words[], int start, char word[])	Given	This function finds the next word from the *words* array and stores it in *word* (as described in the notes).
void drawHangman (int livesLost)	Given	This function draws the hangman depending on how many lives have been lost (passed as a parameter).
int randomNumber (int max)	Given	This function returns a random number between 0 and *max* – 1. For example, if *max* is 5, it could return 0, 1, 2, 3, or 4.

void drawWord (char displayWord[])	To write	This function displays the word being guessed by the player where each letter is separated by a space.
int getWords (char allWords[], Word words[])	To write	This function obtains all the words and their corresponding categories from the *allWords* array of characters. The *Word* structs are stored in the *words* array.

void setWordToUsed (
  Word words[],
  int numWords,
  char wordSelected[])
	To write	This function finds *wordSelected* in the *words* array and sets its *used* field to *true*.

bool findCategoryWord (
  Word words[],
  int numWords,
  char category[],
  char wordSelected [])
	To write	This function finds all the words in the *words* array that are in the *category* specified (if not used before) and stores them in an array. It then randomly chooses a word from this array and stores it in *wordSelected*. The *randomNumber* function can be called to generate a random location in the array.

bool checkLetter (
  char guessWord[],
  char displayWord[],
  char letter)
	To write	This function checks the word to be guessed to determine if *letter* is present. If so, the corresponding letters in the *displayWord* are revealed.

void initWord (
  char guessWord[],
  char displayWord[])
	To write	This function initializes the word to be displayed based on the word to be guessed. It also reveals the letters from "RSTLNE" if they are present.

int lettersMissing (char displayWord[])	To write	This function determines how many letters must still be guessed in the *displayWord*. The count includes repeated letters.
int playGame()	To write	This function plays one game of Hangman with the user. There are three outcomes: the user loses, the user wins by guessing letters, and the user wins by guessing a complete word (where at least two letters were remaining).
int main()	To write	The *main* function reads all the characters into an array and then creates an array of *Word* structs. It then repeatedly plays a game of Hangman until the user decides to quit.

You can use different functions from the ones given in the table above. Your program will be marked for the appropriate use of functions whether or not they come from the table above.

You may find it useful to use the following C-String functions:

```
void strcpy(char a[], char b[])
   // copy the contents of C-String b to C-String a.
```

```
int strlen(char a[])
    // returns the amount of characters in the C-String a.

int strcmp(char a[], char b[])
    // compares two C-Strings a and b, based on "alphabetic" ordering. If they
    // are the same, 0 is returned.
```