PIC 10A Section 2 - Homework #10 (due Sunday, March 13, by 11:59 pm)

You should upload each .cpp (and .h file) separately and submit them to CCLE before the due date/time! Your work will otherwise not be considered for grading. Do not submit a zipped up folder or any other type of file besides .h and .cpp.

Be sure you upload files with the precise name that you used in your editor environment otherwise there may be linker and other errors when your homeworks are compiled on a different machine. Also be sure your code compiles and runs on Visual Studio 2022.

WORD RUN

This assignment is focused on building familiarity with streams and data structures. Hopefully it will be fun, too!

In this assignment, you are to write a game that parallels Wordle, but adds a component of time. You will submit 3 files in the end: **WordRun.h** (providing declarations of constructors/member functions), **WordRun.cpp** (providing definitions), and **Game.cpp** (providing the main routine for the game).

Here is what the program will do:

The user is asked for their name and either told their previous high score or notified they haven't played before. They are then asked if they want to play. Assuming they choose to play, a four-letter word will randomly be chosen from a file (details later) and a collection of valid four-letter words is initialized. After instructions are displayed, the user will be asked to guess the secret word. If their word is among the words of collection, they will see a pattern displayed beneath their guess where '-' indicates the letter below which it appears is not part of the secret word, '~' indicates the letter below which it appears is in the secret word but not at that location, and '*' indicates the letter below which it appears is in its rightful place. If the word is not part of the collection, they are told so. They proceed to guess until they eventually guess the word. At this point, they are awarded points out of a maximum of 300 points (scoring explained later). These results are saved and the program finished after displaying their results.

A few possible runs of the program are below to illustrate the sequence of events before

more precise stipulations are given (other cases need to be considered and are explained in the instructions). Note that the file of four-letter words provided may leave something to be desired but it was the best that could be found quickly.

Figure 1: Not played before and chooses not to play.

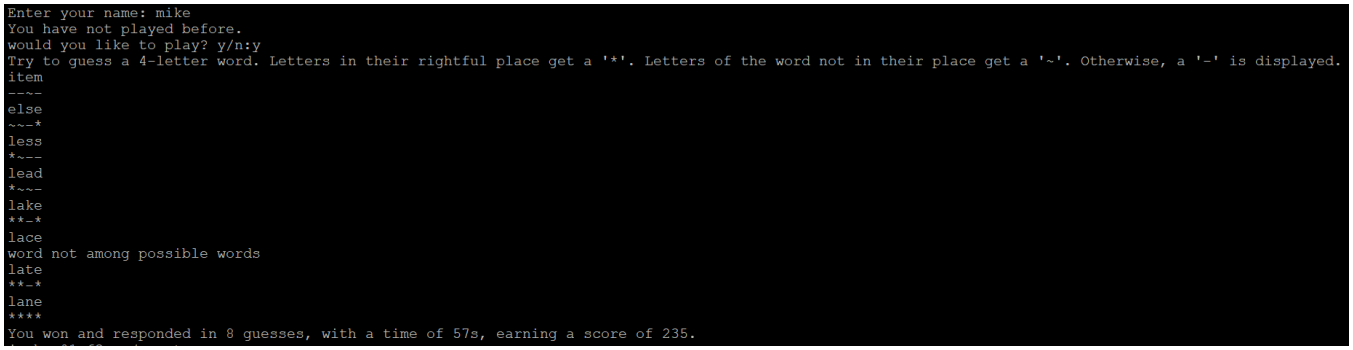

Figure 2: Not played before, does play, and eventually finishes game.

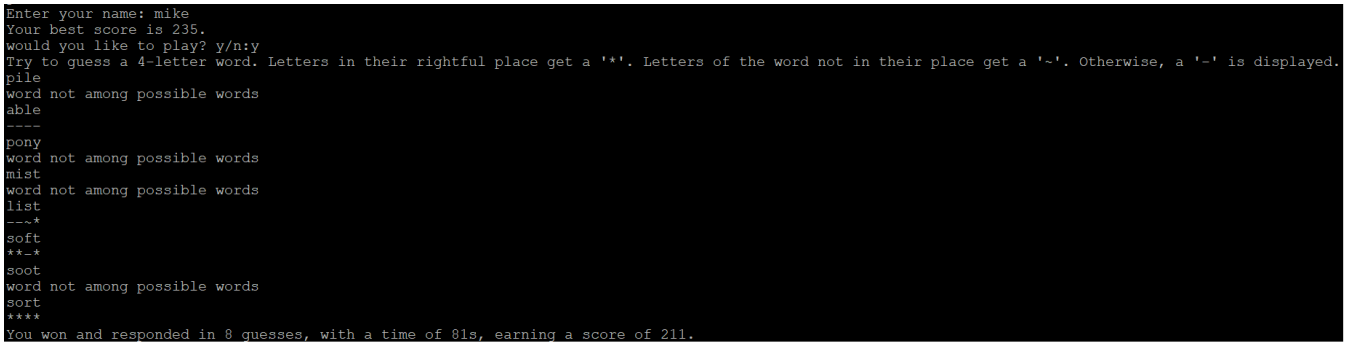

Figure 3: Has played before, does play, not a new best.

Here is the logical flow (before **class**-specific items):

1. The user is first prompted with "Enter your name: " for their name (can include spaces).
2. Based on their name, they are either told "You have not played before." or "Your best score is [N]." where [N] denotes their best score.
3. They are asked, "Would you like to play? y/n:" and the user can enter 'y' for "yes" or 'n' for "no."

4. If they select ‘n’ then the program ends with a message “goodbye”.

5. If they select ‘y’ then instructions are displayed, “Try to guess a 4-letter word. Letters in their rightful place get a ’*’. Letters of the word not in their place get a ’ ’. Otherwise, a ’-’ is displayed.” and the game begins.

6. As the game takes place, the time the user takes (in seconds) and their number of guesses used are tracked.

7. They keep guessing and on the following line see either 4 characters (‘-’, ‘*’, or ‘~’) to give them hints if their word is among a dictionary of four-letter words or they are given the message “wrong size” (if their word is not four-letters) or “word not among possible words” (if their word is four letters but not in the collection).

8. A ‘-’ hints the letter above is not in the secret word, a ‘~’ indicates the letter is in the word but not at that location, and a ‘*’ indicates the letter is in its rightful place.

9. At the end of the game (when the user guesses the word), they are given the message, “You won and responded in [G] guesses, with a time of [T] s, earning a score of [S].” Here, [G] is their number of guesses, [T] is their number of seconds taken, and [S] = TimeScore + GuessScore, where TimeScore = max(0, 150-[T]) and GuessScore = max(0, 150-[G]).

10. The results of each game are *appended to the end* of a file for that user.

To handle the game, you should write a class **WordRun**. The **WordRun** class should:

- store the list of four-letter words in an appropriate data structure that *allows for rapid lookup of words (generally speaking, that will not be a vector!)*, **words**;
- store the chosen secret word as an std::string, **chosen**;
- store the characters within **chosen** in a suitable structure, **chars**;
- store the number of guesses used, **guesses**;
- store the start time and duration of play as two **time_t** variables, **start** and **duration**;
- store a **bool** for whether the user has guessed the word, **won**; and
- store a membeer variable for the score, **score**;
- have a constructor to suitably initialize all the member variables assuming the four-letter words are stored in a plain text file in the same directory as the program called “word4.txt”;
- have a **setWords** function, called from the constructor, to store the words;

- have a **chooseWord** function, called from the constructor, to choose a word and set its characters;
- have a **guess** function that can be fed a std::string and proceeds to handle the hint-giving and checking of the input;
- have a **setScore** that is called when the user has won the game (should probably be called from within **guess** at some point);
- have an accessor function **hasWon** returning if the user has won;
- have an accessor function **getScore** returning their overall score (or -1 if it the game has not been won yet);
- have an accessor function **getTime** returning the number of seconds they took (or -1 if the game has not been won yet); and
- have an accessor function **getGuesses** returning the number of guesses they took (or -1 if the game has not been won yet).

The user data should be stored in a text file of the form "[Name].txt" where [Name] is the user's name. Most operating systems do allow spaces in filenames. The text file for each user must **store all scores that user has ever had!**

Hints:

1. First, make sure you can read all the words from "word4.txt" into a suitable structure.
2. If you can do that, decide on a way to select a word at random (you won't have random access if you do this correctly).
3. With the first two items resolved, you can write part of the class constructor and implementation (ignoring all the guessing, etc.).
4. Now, try to write the **guess** function but focus on just getting the basics right: how do you know if a letter at any given **char** of the user's guess is in the secret word? How about knowing when it is in its correct place? Add in the edge cases of the word not being in the collection or the guess being too many or too few characters.
5. To handle timing, remember the <**ctime**> header and the **time** function.
6. Don't forget to increase the guess count each time a guess is made.