ENGR:2730 Computers in Engineering
Bell & Christensen, Spring 2022, Homework #3

Assignment:

W O R D L E

(100 points) Deadline: Milestones 1 and 2 are due Fri., Feb. 18, 2022 at 11:59pm
Early turn in bonus for Milestones 1 and 2 is Wed., Feb. 16, 2022 at 11:59 pm

(100 points) Deadline: Milestone 3 and 4 is Fri., Feb. 25, 2022 at 11:59pm
Early turn in bonus for Milestones 3 and 4 is Wed., Feb 23, 2022 at 11:59 pm

(100 points) Deadline: Milestone 5 and 6 is Fri., Mar. 4, 2022 at 11:59pm
Early turn in bonus for Milestones 5 and 6 is Wed., Mar. 2, 2022 at 11:59 pm

---

This homework was adapted from the web-based word game *Wordle* developed by Josh Wardle: https://www.powerlanguage.co.uk/wordle/.

Players have six attempts to guess a five-letter secret word. Feedback is given for each guess that indicates whether each letter of the guess is (1) correct and in the correct spot, (2) correct but in the wrong spot, or (3) the letter is not in the secret word.

Homework Goal: Practice writing/using C++ classes, splitting up programs into multiple files, and problem solving.

Homework Details:

The Wordle program is broken down into two parts:

- The main program that has the responsibility for creating objects, unit testing, and responding to user input.
- Wordle—This class encapsulates all the information for a single Wordle puzzle.

Milestone 1: Partially implement, document, and test the `Wordle` class

The **`Wordle`** class must be implemented in the files **Wordle.h** and **Wordle.cpp** and have the following functionality:

1. Add a private string called **wordListFilename** to the Wordle class. This string will hold the filename of a file that contains a list of all valid 5-character words.
2. Implement getters and setters for **wordListFilename**. Note that you must implement these methods even if they are not used in your final program. Be sure to follow the Principle of Least Priviledge.
3. Implement an **explicit**, one-argument constructor for the Wordle class that takes a string. The argument should have a default value of the empty string, i.e., "". The constructor should call the **wordListFilename** setter.
4. Add a private C++ vector of strings called **wordList** to the **Wordle** class. The **wordList** object should be initialized as an empty vector. *Note that the constructor of a C++ vector does this automatically. There is nothing that you need to do to initialize wordList to be the empty vector.*

5. Implement a **loadWordList()** method that loads the list of words from **wordListFilename** into the **wordList** vector of strings. **The format of the file is one word per line with either a newline or not after the very last word.** This function should (0) open the file specified by **wordListFilename**; (1) read the list of words to count the number of words; (2) use the vector's resize method to allocate the required number of strings in **wordList** to save the words, (3) use **fin.clear();** to clear the end of file flag, (4) use **fin.seekg(0, fin.beg);** to return the file reading to the start of the file; and then (5) read in and store each word in **wordList**. If a file of **wordListFilename** cannot be opened, an error message should be printed to the console (see example below).
6. Implement a public **printWordList()** method that prints all the words in the **wordList** vector. A warning message should instead be printed if there are currently no words in the **wordList** vector.
7. Write unit tests for each method in the main function. That is, you should hard code multiple tests in main that test each special case of your constructor, setters and getters, loadWordList, and printWordList. Each unit test should print the expected output followed by the real output. **Your unit tests must appear in main in your final homework solution. Do not delete or comment them out. The only exception to this rule is that you should comment out the printWordList() after you confirm that your program can successfully read and load the words into your Wordle object.**
8. Document your main program and the **Wordle** class.

Example code and output:

```
Wordle test;
test.loadWordList();
test.printWordList();

Wordle wordle("wordList.txt");
wordle.loadWordList();
wordle.printWordList();
```

```
Error: Word list filename has not been specified.
Warning: Word list is empty.
cigar
rebut
sissy
…
zygon
zymes
zymic
```

The above code and output are to help you get started. You must make your own tests to fully test your class. Note, your output messages can be different than above but should convey similar information.

Milestone 2: Select a random secret word

Do not start Milestone 2 until you have completely finished Milestone 1.

1. Add a private string called **secretWord** to your **Wordle** class. This string holds the secret word to guess.
2. Implement a getter method for **secretWord**. Be sure to follow the Principle of Least Priviledge.

3. Implement a setter method for **secretWord.** We don't want to allow just any word to be **secretWord**, so you will need to validate that the provided word (passed into the setter as an argument) is a valid word in the **worldList** vector. The following code can be used to determine if the input word is in the **wordList**.

   ```
   // Check if word is in word list
   if (find(wordList.begin(),wordList.end(),word) == wordList.end()){
       // word is not in wordlist. Do something.
   } else {
       // word is in wordList. Do something.
   }
   ```

4. Implement a **setRandomSecretWord()** method in your **Wordle** class. For this task, you will need to modify the Wordle constructor to also seed a random number generator using "srand (time(nullptr));". Then, use rand() in your **setRandomSecretWord()** method to select and set the class secret word variable.
5. Write unit tests to test each method above in the main function. Do not delete or comment your previous tests.
6. Document your main program and the **Wordle** class.

Milestone 3: Record guesses

Do not start Milestone 3 until you have completely finished Milestone 2.

1. Add the following private variables to your Wordle class.
```
int guessNum = 0;                // number of guesses
const int MAX_GUESS_NUM = 6;     // maximum number of guesses

vector<string> guess;            // list of previous guesses
vector<string> feedback;         // list of feedback strings
```
2. Implement a getter for **guessNum**. Do not implement a setter for **guessNum** because it will be the class's responsibility to make sure **guessNum** remains valid.
3. Implement a **makeGuess** method. This function should have a single input parameter which is a word stored in a C++ string. This function returns true if the input word matches the secret word or false otherwise. This function must also check if the input word is a valid 5-character word and in the word list (see Milestone 2.3 to see how to check if a guess is in the wordList). If the word is valid and the guessNum is less than MAX_GUESS_NUM, then (1) the word should be added to the guess vector, (2) the sting "?????" should be added to the feedback vector, and (3) the guessNum incremented. The correct feedback for the guess will be implemented in a later Milestone; don't do this yet. Your function should print a message if the input word matches the secret word. Your function should print a game over message and the correct secret word if it is the last guess and the input word did not match the secret word.
4. Implement a **print()** function that prints each guess in **guess** vector and its associated feedback in the **feedback** vector to the screen.
5. Write unit tests to test each method above in the main function. Do not delete or comment your previous tests.
6. Document your main program and the **Wordle** class.

Example code and output:
```
Wordle wordle("wordList.txt");
wordle.loadWordList();
wordle.setSecretWord("blue");
wordle.setSecretWord("llama");
cout << "Setting secret word to \"llama\". The secret word is " <<
wordle.getSecretWord() << ".\n" << endl;
wordle.makeGuess("blue");
wordle.makeGuess("thars");
wordle.print();
wordle.makeGuess("vezir");
wordle.makeGuess("vezir");
wordle.makeGuess("vezir");
wordle.makeGuess("stall");
wordle.print();
wordle.makeGuess("llama");
wordle.print();
```

```
Warning: blue is not in word list.
Unable to set blue as the secret word.

Setting secret word to "llama". The secret word is llama.

Incorrect guess: blue is not in word list.
Please make different guess.
```

```
1: thars ?????

1: thars ?????
2: vezir ?????
3: vezir ?????
4: vezir ?????
5: stall ?????

Congratulations!! You guessed the secrete word.
1: thars ?????
2: vezir ?????
3: vezir ?????
4: vezir ?????
5: stall ?????
6: llama ?????
```

Milestone 4: User input for your Wordle program

Do not start Milestone 4 until you have completely finished Milestone 3.

The next step in the process is to write the **main** program that uses the Wordle class to implement the Wordle game. Do not delete or comment the unit tests that you made in the previous Milestones. Immediately after these unit tests, you should implement the following.

1. Print a list of instructions for playing the game.
2. Instantiate a Wordle object. Load the valid word list into the object.
3. Set a ***known*** secret word and print it to make testing easier.
4. Implement a while loop to let the user enter up to six guesses. The loop should use the Wordle object to make guesses, use the getGuessNum() function to know how many valid guesses have been made (Note: the Wordle class only increments guessNum when the user enters a valid word). After each guess, the program should print the list of guesses and associated feedback for the guess (Note: true/correct feedback should not be implemented yet). The while loop should terminate when the user inputs the secret word. The main program should give the user feedback whether they won or lost the game before terminating.
5. Document your main program.

Milestone 5: Feedback about guess

Do not start Milestone 5 until you have completely finished Milestone 4.

1. Modify your **makeGuess** method to give feedback of correct letters in the correct position and correct letters in the wrong position. This feedback will be a five-character string such that: '.' denotes that the letter entered in that position does not appear in the secret word, '!' denotes the letter is correct and was entered in the correct position, and '+' denotes that the character appears in the word but is in the incorrect position. Implement your own logic or the logic below.

   1. Assume all characters in the feedback string are initially '.'
   2. Make another variable that is a copy of the secret word
   3. Compare the guess character-by-character with copy of the secret word. Replace '.' with '!' in the feedback string if the characters match. When a character is matched, replace the character in the copy with a '-' so it will not be matched again.
   4. <u>For each</u> character in the guess that does not have an '!', compare it to <u>all the characters in the copy</u>. If it matches a character in the copy, replace the character in the copy with a '-' so it is not matched again and replace the '.' at that position with a '+'.

2. Write unit tests to test the method above in the main function. Do not delete or comment your previous tests. Insert these unit tests above the code that you inserted in Milestone 4.
3. Document your main program and the **Wordle** class.

Milestone 6: Feedback about possible letters in secret word

Do not start Milestone 6 until you have completely finished Milestone 5.

For ease-of-play, it is nice for players to know what letters might still be in the secret word. In this Milestone, you will implement functionality to keep track of which letters might still be in the word (i.e., the letters are *available*) or can be discounted as NOT being in the secret word (*unavailable*) based on previous guesses.

1. Add the following private variable to your Wordle class.
   ```
   bool availableLetter[26];
   ```
   This array to holds all “available letters” that still might be in the secret word. Index 0 corresponds to 'a', 1 to 'b', etc. and at each index, a value of true means the character is available.
2. Modify your constructor to initialize this array to all true.
3. Modify your **makeGuess** method to change the value at the proper index to false if a guessed letter is not in the secret word.
4. Implement a **printAvailableLetters()** method to print the letters that might still be in the secret word.
5. Call the **printAvailableLetters()** method in the game’s while loop in your main function.
6. Write unit tests to test the method above in the main function. Do not delete or comment your previous tests. Insert these unit tests above the code that you inserted in Milestone 4.
7. Document your main program and the **Wordle** class.

Congratulations! Once you've gotten this working, you've just finished implementing your very own Wordle application.

Sample game:

```
Guess the WORDLE in 6 tries.

Each guess must be a valid 5-letter word. Hit the enter button to submit.

After each guess, you will get feedback to how close your guess was to the word.
! is the correct letter in the correct spot.
+ is the correct letter in the wrong spot.
. is a letter that does not appear in the word.

enter guess: icons
1: icons  ....+

Available letters to guess: a, b, d, e, f, g, h, j, k, l, m, p, q, r, s, t, u, v,
w, x, y, z,
enter guess: stray
1: icons  ....+
2: stray  !!+.+

Available letters to guess: b, d, e, f, g, h, j, k, l, m, p, q, r, s, t, u, v, w,
x, y, z,
enter guess: styre
Congratulations!! You guessed the secrete word.
1: icons  ....+
2: stray  !!+.+
3: styre  !!!!!
```

Submission Instructions

- Make a CLion project called "hw3" (ALL LOWERCASE) in your SVN homework directory.
- Check your homework into SVN.

Hint: you can see the current version of your submission by opening this link in a web browser:
https://class-svn.engineering.uiowa.edu/cie/projects/spring2022/