This is the assignment for unit VII, "files." You are expected to complete the assignment in the C++ language and submit one ".cpp" file. You must complete and submit the assignment on or before the due date of **November 19**. *Remember: Assignments will NOT be accepted more than 7 days late.*
This assignment is worth **4 points**. There is **1 point** of extra credit.

Submission instructions:

- Submit your assignment to me via email: `sklar@sci.brooklyn.cuny.edu`
- Your email subject line should be: **CISC 1110 Lab VII submission**
- Attach your C++ (**anagrams.cpp**) file to your email.
- Make sure your name is in the body of the email message.
- Make sure your name is also included in the header comments at the top of your C++ file.

Overview

This assignment emulates a simple game of Anagrams. The computer will pick a word from a data file of 7-letter words (i.e., words that are 7 letters long). (The data file is provided by me and is posted on the class web site.) The computer will display the word to the user. The user then enters words that can be made out of the computer's word. Each word that the user enters is saved in a data file. The game counts how many words the user enters and displays that number at the end.

A sample run is shown below:

```
number of words in file = 20551
how many words can you find in bugbite?
enter a word: bug
okay!
more (y/n)? y
enter a word: bite
okay!
more (y/n)? y
enter a word: bugs
invalid word!
more (y/n)? y
enter a word: tie
okay!
more (y/n)? n
you found 3 words!
```

step 1 (1.5 points)

1. Create a new C++ program called `anagrams.cpp`.
2. In the `main()`, initialize the random number generator.
3. Create a function that returns a string. The function should do the following:
   - Open the **words7.dat** data file (provided on the class web page). The file contains a number on the first line, which is equal to the total number of words in the file. The rest of the file contains a long list of 7-letter words. Each word appears on a line by itself.

- Read the number of words from the file. Select a random number between 0 and the number of words in the file. Read words from the file until you have reached the random number chosen. For example, if the random number chosen is 10, then your program should read 10 words from the file and stop. Close the data file.
- The function should return the last word read (i.e., the 10-th word in our example).

4. In the `main()`, call the function you created above, and then display to the user the word that was chosen.
5. Compile and run your program to make sure it works correctly.

step 2 (1.5 points)

1. Modify your file by adding another function. This new function should be called from inside the `main()`, after displaying the word returned by the function created above. The new function should take one string argument (call-by-value) and return an integer. The new function should do the following:
   - Open a data file for output. Call it something like "mywords.dat".
   - Prompt the user to enter a word. Read the user's word. Write the user's word to the output data file.
   - Ask the user if s/he wants to enter another word. If yes, then repeat the process, starting where you prompt the user to enter a word. If not, then close the data file.
   - Keep a count of each word that the user enters. The function should return the value of that count.
2. Compile and run your program to make sure it works correctly. Check that your output data file (e.g., "mywords.dat") contains all the words that you entered.

step 3 (1 point)

1. Modify your file by adding another function.
   This new function should be called from inside the function you wrote in step 2, above.
   This function should take two string arguments (call-by-value) and return a boolean (`bool`).
   The two string arguments are: (1) the computer's word (from which the user is making words), and (2) a word that the user entered.
   The new function should compare the letters in the user's word with the computer's word. If there are letters in the user's word that are not in the computer's word, then this function should return `false`. Otherwise, the user's word is okay and the function should return `true`.
2. Modify the function you created in step 2, above, so that the word the user enters is only counted and written to the output data file if the function written in this step deems that it is a valid word.
3. Compile and run your program to make sure it works correctly. Check that your output data file (e.g., "mywords.dat") contains only the valid words that you entered.

extra credit step (1 point)

1. Modify the function you created in step 3, above, so that it accounts for words that have multiple instances of the same letter. For example, if you are NOT checking for duplicate letters and the computer's word is "ALLERGY", then the user could enter "RARE" and that would be okay. However, if you are checking for duplicate letters (i.e., which is what this extra credit part is supposed to do), then "RARE" would be deemed invalid (because there is only one 'R' in the computer's word).
   Note that if the computer's word has duplicate letters, then it is okay for the user to enter a word that has the same duplicate letters. In the example above, where the computer's word is "ALLERGY", the user could enter "ALL" and that would be okay.
2. Compile and run your program to make sure it works correctly.