Project 2 CS 136 Kutztown

Purpose: Create and implement an object type and use it in an application. Overload operators.
Points: Project: 25.
Deadline: As stated in class. Submit late at your own risk.
Description: For your next project, you will rework Project 1 to employ an object in an application. It constructs the same list of tokens as was compiled in Project 1.

Each token will now be represented in a full-blown object instantiated from a class, replacing the data only structs from Project 1. Using the suggested name *WordRec*, here are the specs:

WordRec: Data attributes: word or token (string) and number of appearances in the file (int)
Implement these member functions in the **WordRec** class to perform the actions given (operators noted):

- Constructor (with default parameter values) initializes word to "" and counter to 1.
- Set and get for each data member.
- Increment the counter (++; both pre and post)
- Return whether a WordRec object's word attribute is alphanumerically less than that of another WordRec object passed in (operator <)
- Return whether a WordRec object's word attribute is equal to that of another WordRec object passed in (operator ==)

The following non-member associated operators are found in the **WordRec** class' files (these are **NOT** friends):

- Print a WordRec object's data attributes in a well-formatted manner (operator <<)
  - << can call *printf*
- Read a string from an ifstream into a WordRec object (and set its counter to 1) (operator >>)

This class will be stored in files `WordRec.h` and `WordRec.cpp`. `WordRec.h` is provided (more below).

The application, **p2.cpp**, is provided and repeats the Project 1 task of reading the data into an array of WordRec objects and printing it. It sorts it while processing duplicates as was done in creating the drawings for this project.You are to complete it by adding code, to permit the user to request output of the data in several modes, chosen via a menu matching the one in the box.

```
Choice:
A)ll Words with Number of Appearances
P)rint Words That Appeared N Times
F)ind a Word
Q)uit
```

Functions (provided):

- Open the file
  - Same as Project 1's openFile()
- Read the file into the array of words
  - In a loop, call the >> operator to input the data.
- Sort/Remove duplicates.
  - This is a single operation, with the sort function calling helper functions like *isDup*().
- Process User Requests
  - In a loop prompt and input the user's choice. You must use the character choices (case insensitive) in the menu in the box. If the user enters:
    - 'A', print all words and their multiplicities, well formatted, in a loop using the << operator you wrote for the WordRec class; you should be able to condense and convert the print function from project 1 for this purpose.
    - 'P', input an int and call a function (also in `p2.cpp`) that prints qualifying words.
    - 'F', perform the search in yet another function in the application, reporting the result.
    - 'Q', terminate the program.

Project 2 CS 136 Kutztown

Notes:

- main() is mostly function calls, along with the declaration of the array of **WordRec**. Each task must be in its own module (function). This includes handling the user's choice in the application. Modular development will be a point of emphasis in grading.
- Proper object-oriented design is a must. Implement your **WordRec** class properly, using sets and gets to access the data, even in other member functions.
- A global constant is declared to permit variable sizing of the WordRec array as you develop the program.
  - The capacity is 25 elements. Set it to a smaller list for development and reset to 25 when you submit.
- Note that the sort (written in p2.cpp) does not directly compare word members. It uses the overloaded < operator to compare entire WordRec objects by their string member (the token they hold).
  - Similarly, the application's *find* operation will make use of *==(const WordRec &,const string &)* from above.
- Work incrementally. Implement the menu and its items in the application, one at a time. Use multiple test files and a small array at first.
- Your prompts and function names must be descriptive.
- Input of menu choices **must** be case insensitive. Letter grade penalty for non-compliance. Look up *toupper*() .
  - You will be penalized 5 points if your application doesn't loop.
- Your output must be well labeled and easy to read. It must have category header(s) when user input results in output of tabular data. When a user specifies a multiplicity, do not print how many times each word appeared (it will be the same for all of them; put that info in the header). No category headers should be printed if no data is output; print a 'not found' message.
- All << operators must take a constant reference to the object. Note that >> operators can't have a constant reference; instead, they have a reference (Why?)
- p2.cpp and a makefile is provided for you in the Projects/Project2 directory of the public area on acad and on the web. It creates an executable named **p2**. Don't forget: Invoke your program using **`./p2 .`**
- `WordRec.h` is provided in Projects/Project2 as well. You MUST use this class header. Do NOT add any functions to this file. Complete the documentation for each function by adding pertinent info, including describing the function as a mutator, facilitator, or inspector, and the parameters by type, i.e. import, export, or import/export. You must complete this. Submissions with this work missing will not be graded.
- Complete id blocks and documentation in the application `p2.cpp`, except that application functions are not classified as mutator, facilitator, or inspector.
  - Implementation file `WordRec.cpp` is to have a complete id block on top and brief descriptions on all functions (the descriptions may be copied from the header file)
- To grade your program, the instructor will expect to type *make* to compile the file, and *p2* to run it. Make sure that it will run **on acad** under these conditions.
- Your program must be well written, properly indented, and commented. If you aren't sure, ask!

Deliverables: All code files: Application **p2.cpp,** all lower case. **WordRec** class (.h and .cpp files; named with capital W and R to match specs in makefile). The makefile. 2 point penalty for non-compliance on exact file names, 1 more for not turning in the makefile. You may use different names for files if your makefile works. Name of application MUST be `p2`.