Program 3 CS 237 Kutztown

Topic: Inheritance and Polymorphism
Points: 25
Due: See Deliverables section. Submit late at your own risk.

Overview: For this project, you will write a small software system that again counts occurrences of words, as in Project 1. But its implementation will follow a strictly object oriented design that employs inheritance and polymorphism.

You will expand the inheritance hierarchy provided in the WordDataList subclass example to include two WordList subclasses, one employing an STL sequential container, and the other employing a circular linked list with iterator.

Using the provided WordList example inheritance hierarchy from class, available on acad and the webpage for this project, add new WordList subclasses named WordSTLSeq and WordCList, with both implementing WordList's pure virtual functions. The WordSTLSeq subclass of WordList will have a templated sequential container to hold its data and implements the printRecursively() member function as well; it uses a C++ iterator as the pointer. WordCList's data member will be a CLinkedList<WordData> instance of the templated circular-linked list object that you created in Project 2. The inherited virtual member functions you implement for printing will use the list iterator wrapper class you wrote to access the linked list, printing the contents of each node using the existing stream insertion operator for an ostream and a WordData object.

```
1: Object Array Iterative
2: Object Array Recursive
3: Object Array Pointer Only
4: Circular List Iterator
5: Circular List Iterator Recursive
6: STL __________ Iterative
7: STL __________ Recursive
8: Exit
```

Write a menu-driven test driver with options **exactly** as in the box. (The blanks will be filled in with your choice of STL sequential container.) Update the provided WordList application example as follows:

- Update the menu.
- Each time the user makes a selection, the WordList pointer **TheList** is to be instantiated to point to an object of the appropriate type.
- Use polymorphism to call the correct implemented version of the WordList print functions. These calls are already in the switch statement and may only require adding cases.
- Each time the requested action is completed, reclaim the memory pointed at by **TheList**.

Note: The call of *parseIntoList*() in app.cpp does not require updating. It should polymorphically call the correct version according to the object pointed at by **TheList**.

Once your program is complete, back it up and add code to time *parseIntoList()* and the print functions. At the bottom of each table, print both times, well labeled, to read the data and to print, both in microseconds. Use the Microseconds example to guide you in timing routines.

Requirements, Notes and Suggestions:

- The WordList hierarchy example is provided and can be executed as is. Use it as a guide for your other subclasses.
  - Programs that don't adhere to the manner in which this demo runs will receive a grade of zero. No reprieves; no 2nd chances.
- You must choose a sequential container from the STL for that subclass. In your readme you **must** name the containers (at least 3) that should be considered and why you made your choice.
  - This is a point of emphasis in this project. Failure to consider appropriate STL objects and provide a clear, cogent, **well-organized,** and complete narrative in the readme that justifies your choice will be penalized up to 5 points.
- The STL container must be maintained sorted in ascending order by tokens.
  - It has no limit on tokens.
- Your program's menu will have 8 options. The WordData list has the pointer option, the others just iterative and recursive (the pure virtual functions you implement in the subclasses). Option 8 is to exit.
- The program is still to be runnable from the command line. If there is a command line argument, your program will NOT prompt for a file name and will instead assume the command line argument is a file name. It will then read the file and process it in the same manner as if the input had come from the user, running each menu option and printing to the screen, labeling each output appropriately, including the menu option text. After all possibilities are exhausted, the program terminates. Automatic 5-point penalty if this doesn't work.
- Consider the use of helper or worker functions to make recursion work.
- **WORK INCREMENTALLY!!! ONE SUBCLASS/MENU ITEM AT A TIME.**
- Timing program execution is imprecise on a computer that isn't dedicated to the experiments.
  - Make sure that the timings of your program are realistic, as your 2nd homework will use the results of timing within your program.
- You must carefully and completely document your code and place id blocks in all files, including those provided from external sources (e.g. the course instructor).
  - Lazy, chintzy, or otherwise substandard documentation is not acceptable.
  - Programs with significant documentation issues will be penalized at least three letter grades (7.5 points).
- You will use the Doxygen tool to create an API for your project and post it on the web.
  - Mainpage and all \briefs are to be included to create a professional interface.
  - There is a robust example of Doxygen prep and appearance here: https://faculty.kutztown.edu/spiegel/CSc237/Examples/DoxygenDemo/PolygonList/
  - **Parameter types are to be provided**. Include within `\param` tag.
  - We will discuss in class. There are primers on my links page.

Deliverables: Deadline TBA. Submit late at your own risk.

⇒ D2L Project 3 dropbox

- Readme containing Doxygen link, reasoning for your selection of STL data structure and any other material deemed appropriate, including bug report.
  - Again, submissions that don't follow these specs will lose 5 points.

⇒ Acad:

- All h and cpp files, and a makefile. You may want to use the makefile provided with the WordDataList example as a starting point. A correct makefile is part of the assignment. It is to produce an executable named ***app*** (that name exactly) by default. Penalties for non-compliance.