Program 1: "Word Processing"
CS-240: Data Structures
Assigned Tuesday September 23, 2014

Goal

In Program 1, you will continue with C++ I/O and text processing, and you will begin to utilize slightly more complex data structures to store information. In particular, you will implement a *singly linked list* container to store, manipulate, and save stories found in text files. Program 1 will provide you experience with:

- a simple singly linked list container that requires pointers
- C++ heap memory management (strategic and extensive use of `new` and `delete`)
- deep copies of stored data, including in copy constructors and assignment operators
- operator implementation and overloading (including assignment operators, unary operators, binary operators, and `friend` functions)
- some simple C++ class and object design.

This program will be difficult for some of you so please begin early and work on it consistently. You can do it and we will help! ~~If~~ When you finish successfully, you will *really* understand linked lists, operator overloading, deep vs. shallow copy, and more, *guaranteed*!

Overview

Write a C++ program that reads and stores information from and to text files that contain "stories," which are held in your program as instances of a new Story class that you write. Programmers (that is, users who write code and link it against your code to use C++ classes and functionality that you provide) should be able to:

(i) combine two stories to create a new one,
(ii) copy one story to create another identical one,
(iii) convert stories to all caps, all lowercase, appropriate capitalization, or Pig Latin (!), and
(iv) pull apart and recombine stories and their components (paragraphs and sentences).

Each instance of Story will contain an ordered *linked list* of Paragraphs, each of which will contain an ordered *linked list* of Sentences, which in turn will contain ordered *linked lists* of Words. Story, Paragraph, Sentence, and Word are all new C++ classes that you will design and implement. Programmers will manipulate instances of your classes almost exclusively using overloaded C++ operators. Details of the operators you should support, along with their semantics, appear below.

Requirements

Overload the following operators to operate on (or with) the four C++ classes that you write, as follows.

a + b
(In the descriptions below, please read "Type"—where Type is either Story, Paragraph, Sentence, or Word—to mean "instance of Type".)

- Story + Story evaluates to a new Story containing the paragraphs of the first Story followed by the paragraphs of the second Story.
- Story + Paragraph evaluates to a new Story with an additional paragraph at the end
- Paragraph + Story evaluates to a new Story with an additional paragraph at the beginning
- Paragraph + Paragraph evaluates to a new Paragraph with Sentences of the first followed by Sentences of the second
- Paragraph + Sentence evaluates to a new Paragraph with an additional Sentence at the end
- Sentence + Paragraph evaluates to a new Paragraph with an additional Sentence at the beginning
- Sentence + Sentence evaluates to a new Paragraph containing the two sentences
- Sentence + Word evaluates to a new Sentence with the Word added to the end
- Word + Sentence evaluates to a new Sentence with the Word added to the beginning
- Word + Word evaluates to a new Sentence containing the two words.

a++

- If **a** is a Word, makes it all caps.
- If **a** is a Sentence, makes the whole sentence all caps.
- If **a** is a Paragraph, makes the whole paragraph all caps.
- If **a** is a Story, makes the whole story all caps.

a--

- If **a** is a Word, makes it all lowercase.
- If **a** is a Sentence, makes the whole sentence all lowercase.
- If **a** is a Paragraph, makes the whole paragraph all lowercase.
- If **a** is a Story, makes the whole story all lowercase.

a + 1

- If **a** is a Word, capitilizes the first letter.
- If **a** is a Sentence, capitalizes the first letter of the sentence.
- If **a** is a Paragraph or Story, capitalizes the first words of all contained sentences.
- **a + 2** (or a + <any other integer>) should not alter the Word, Sentence, Paragraph, or Story in any way. Simply ignore it.

++a

- If **a** is a Word, converts it to *Pig Latin*. (See below for the very simple definition of Pig Latin.)
- If **a** is a Sentence, converts the whole sentence to Pig Latin.
- If **a** is a Paragraph, converts the whole paragraph to Pig Latin.
- If **a** is a Story, converts the whole story to Pig Latin.

--a

- If **a** is a Word in Pig Latin, converts it from Pig Latin to plain English.
- If **a** is a Sentence, Paragraph, or Story, converts all contained Words that are in Pig Latin back to English.

[Please note that **a++** is different from **++a** in C++. When ++ follows the variable, this is "postfix ++"; when it precedes the variable, it is "prefix ++". Do a small amount of investigating to see how C++ allows you to overload postfix and prefix ++ and -- as four separate functions with different functionality. Let us know if you have trouble; this should be no more difficult than overloading any four different operators.]

<<

- A Word class should output only the word's characters, in order.
- The Sentence class should output each word. All but the last word should be followed by a single space. The last word should be followed by the punctuation mark associated with the Sentence ('.', '!', or '?'), and then a single space.
- The Paragraph class should output each sentence, should be indented with a single tab character, and should be followed by a single newline character.
- The Story class should output contained paragraphs, with blank lines between them.

`first()` and **`rest()`**

You should support functions `first()` and `rest()` on Story, Paragraph, and Sentence. If `myStory` is an instance of Story, `myPara` is an instance of Paragraph, and `mySent` is an instance of Sentence, then:

- `myStory.first()` returns a Paragraph containing a copy of the first paragraph of `myStory`.
- `myPara.first()` returns a Sentence containing a copy of the first sentence of `myPara`.
- `mySent.first()` returns a Word containing a copy of the first word of `mySent`.
- `myStory.rest()` returns a Story containing a copy of all but the first paragraph of `myStory`.
- `myPara.rest()` returns a Paragraph containing a copy of all but the first sentence of `myPara`.
- `mySent.rest()` returns a Sentence containing a copy of all but the first word of `mySent`.

Finally, you should support a value constructor on the Story class that takes as a parameter the name of a file from which to read a story. For example, the following line should open the file named `story.txt` and construct from it a new instance of the Story class, with contents appropriately divided into objects that contain linked lists of Paragraphs, Sentences, and Words.

```
Story myStory("story.txt");
```

You may assume that files contain only letters (capital or lowercase) and end-of-sentence punctuation ('`.`' or '`!`' or '`?`'), and that all stories stored in files are in English, not Pig Latin. You may assume that input files are well formed, as if generated by using the insertion operator (`<<`) to output an instance of your Story class. (You did plenty of "type checking" of input for Lab 3… let's focus on the fun stuff for Program 1!)

You should also support a `save()` function on the Story class. The following line of code should cause the `myStory` object to be written out to a new file called "`storyout.txt`", which you should create if it does not exist. If it does exist, please overwrite its contents.

```
myStory.save("storyout.txt");
```

Your program should *not* accept any command line arguments nor read any information from `cin`. We will test your program by linking various test programs against your code; those test programs will create and test instances of your classes. Details and sample programs will follow soon.

*You may **not** use the C++ string class **nor** any C-style string manipulation functions.* Do not even `#include` those libraries anywhere in your code! To support "strings," your Word class should contain an array of characters. Look into operations on the `char` C++ type; you should *not* have a giant switch statement with options for each letter of the alphabet; there is an easier way to do everything you need to do with words.

*You may **not** use C++ standard library containers.* Please implement your own linked lists.

Pig Latin

A word is converted to Pig Latin as follows.

- If the word begins with a consonant, then the first *consonant sound* is moved to the back of the word and "ay" is added.
    - Example: "Like this." becomes "Ikelay isthay."
- If the word begins with a vowel, then 'w' and then "ay" are added to the end of the word.
    - In autumn, apples are incredibly awesome.
    - Inway autumnway, applesway areway incrediblyway awesomeway.

Linked Lists

We have not covered the *linked list* container type yet. We will do so in class on Monday September 29th. Please feel free to read the section of the book that describes linked lists, and to do some outside research on them. In particular, the Wikipedia page on linked lists is a useful and accurate resource. For this program, you will need only *singly linked lists*, so you may skip descriptions of other "fancier" linked lists (doubly linked lists, circular lists, etc.).

A great start on this program would be to have all four classes in place, building properly into `.o` files, defining all overloaded operators and necessary constructors and destructors, etc., but with those function bodies simply returning dummy values. Once you have done that, you can begin to add linked list functionality and to support the required operations. This is what I would have assigned as Lab 4, if we had a regular week of classes.

Submission

Please follow the Submission Conventions described in Lab 1, to produce an appropriately named "tar ball" for Program 1. Everything should unpack for us into a single directory named **Lastname_Firstname_Program1**. Submit your code to Blackboard.