Project 4

Lists, Arrays and Sorting

Due Monday November 7th by 5:00 pm

100 points + (10 extra credit)

For this project you are required to implement a simple `List` data structure. Lists are dynamic data structures that can grow and shrink in size as needed, meaning that unlike the arrays of data we've seen in this course thus far, you do not need to know the number of items to store before creating your list object. Once implemented, you will use your `List` class in a small program that reads and sorts a simple text dataset of book titles.

Part I. List Implementation

Existing code for your `List` implementation will be provided. In order to use this code however, you must conform to a specified class interface in all your class definitions. This means that you **must not change any member function prototype** in the provided header files or change the header file names themselves. The provided source files are as follows:

`book.h` This is the complete header file for the `Book` class; you must complete the class by providing the corresponding implementation.

`list.h` This is a *partial* header definition. You are required to add all corresponding private member variables needed to complete the class definition as well as provide the implementation itself. You may implement the functionality of your List class using either a *Singly Linked List* or a *Dynamic-sized Array*. Both of these approaches are explained in the background information and will be discussed further during class. **No external libraries or STL types may be used.**

`loadfile.h/loadfile.cpp` This code parses a text data set of book titles. You do not need to change anything in this code, only make use of it in your main function to populate your list object using the following function call:

```
load_book_data(<filename>, <list object>)
```

Included with the header files are 2 datasets: *test_dataset.tsv* and *all_ratings.tsv* which contain definitions and ratings for your book objects. For debugging, I recommend using the smaller test data set.

Part II. Sorting

Finally, once you have populated your list with Book items, you must implement a function for sorting books, in descending order, based on their average rating. Your final sorted array should be then be printed to standard output (using std::cout.) along with the list's final size.

Start this project early! While the actual code for implementing lists is quite modest, conceptually understanding how to iterate through lists and

add/remove items can be challenging! A good place to start is to provide function stubs for all class member functions and implement /test your functions one at a time.

Code Requirements:

To receive full credit you must implement all features as outlined above, as well as fulfill the following code specific guidelines:

- Implement the `Book class` object following the provided header.
- Parse a command line argument specifying the input data file.
- Implement a `List` class. You may use either linked lists or dynamic arrays.
- Write a simple function for sorting your list. This can be either a member or no-member function.

Extra Credit:

(*10 points*): Change your list implementation to use *class templates* so that your list class can be used to store objects of any type, not just `Book`. Write some example code to show that your list works.

Grading:

This is a rough outline on how points will be assigned. Remember, if code doesn't compile on the CSG machines or a makefile isn't included additional points will be taken off.

5 points: Implement the `Book` class
25 points: Implement `List` class using either linked lists or dynamic-sized arrays
10 points: Implement a simple sorting algorithm for your `List` type
5 points: Read all arguments by the command line.
5 points: All code is well commented and easy to follow
Extra Credit
+10 points: implement template class list

Example Input/Output:

```
$ ./sort test_dataset.tsv

List size: 99

0887299253  Insight Pocket Guides Paris (Insight Pocket Guides)   1998  10
0155067664  The Writer's Harbrace Handbook      2000  10
1561794597  Focus on the Family Presents Great Stories Remembered 1998  10
...
```

Background Information:

Linked Lists

We will discuss this concept more in class. The Wikipedia article on this subject is pretty good:

http://en.wikipedia.org/wiki/Linked_list

Dynamic-sized Lists

The usual approach for creating a dynamic-sized list is to maintain a dynamically allocated array of pointers to the objects in your list, e.g., for a dynamic-sized list of type `int`, you would allocate an array of type `int*`. This is the approach Python uses for its list implementation. Generally an array of size n is created, and then when more space is needed a new larger array is created. When the size of the list becomes much smaller than the array a new smaller array is created and used for the list. When the new arrays are created the appropriate values are copied from the previous array, and then delete[] is called on the old array.

Sorting

Among the simplest sorting algorithms to implement are *insertion sort* and *bubble sort.* Both of these can be done iteratively using just a few lines of code. The challenge is how to implement these simple methods in the context of your List implementation.

http://en.wikipedia.org/wiki/Insertion_sort

http://en.wikipedia.org/wiki/Bubble_sort

Book-Crossing Dataset

The book data set for this project is based on the public Book-Crossing dataset available at:
http://www.informatik.uni-freiburg.de/~cziegler/BX/