P4 Linked List

- Due May 15, 2018 by 12:20pm
- Points 100
- Available May 3, 2018 at 12am - Jun 15, 2018 at 11:59pm about 1 month

This assignment was locked Jun 15, 2018 at 11:59pm.

More practice with Class creation, Pointers, and an introduction to Linked List concepts.

MYSting : refers to your string class (**Note:** If you weren't able to finish program 3 and update your MYString, then for this program you could use the normal C++ string class. If you are going to use the standard C++ string then you must talk with me so we can discuss the changes you will need to make).
DLL : refers to the doubly linked list class that you will be writing.

Program Description: Your assignment is to write a linked list class. Your linked list class will store your strings from your MYString class that you wrote for program 3. You will need to change your comparison operators (==, <, >) so that they will make a true alphabetical comparison instead of an ASCII comparison.

An alphabetic comparison is what we would normally think of if we were to put words in order. To do this, you can use either one of two c functions tolower(char ch) or toupper(char ch) on the characters as you are comparing them. Both of these functions are found in the cctype library. For better or worse, now "The" would be equal to "the", and so which ever word is inserted into the linked list first would be allowed to enter.

Your linked list will be a doubly linked list, meaning that it will have links to both the next and previous nodes. Your linked list will also be an ordered list, which means that as you insert new strings into the list, they will need to be inserted in the correct position (from smallest to largest). Also, you are only inserting a string into the list if it currently is not in the list, so our list will be made up of unique words.

Programing Suggestion: these are some guidelines on how to start writing your program. write and test it a little bit at a time.

- First add and test the new functions for the MYString class
- Write the Node class
- Start the DLL class with a constructor and the << function (so you can test the contents of your list)
- Write and test a simple insert function without worrying about inserting in the right place (for example pushBack which always insert at the back of the list). This function can be used by your copy constructor and = operator
- Add a new insert function so that it now inserts in the right place

- And so on.....

Node Class:	
Member Data: + data : MYString + next : Node * + prev : Node *	
Member Functions : return type	**Description**
Node( )	constructor
Node(MYString str)	constructor with initialization data

DoubleLinkedList Class:	
Member Data: - head : Node * - tail : Node * - it : **mutable** Node * { used as the "**it**erator" to move through the list by using next( ), resetIteration( ), and hasMore( ) } - count : int	
Member Functions : return type	**Description**
DoubleLinkedList( )	default constructor
DoubleLinkedList(const DoubleLinkedList& dll)	copy constructor
= operator(const DoubleLinkedList& dll)	assignment operator
~ DoubleLinkedList( )	destructor
<< operator	output the data to the ostream (separate each string with a blank space)
insert(const MYString& str) : bool	insert the string argument into the list in the proper place (smallest to largest). Do not insert the string if the DLL already has a string of that value. Return true if the string was inserted, otherwise return false

remove(const MYString & str) : bool	if the string argument is found in the DLL remove it from the DLL object. Return true if a string was removed, otherwise return false.
getCount( ) : int	returns the number of strings that are stored in the DLL.

These following functions are needed for your main to be able to get the MYStrings from the list so that you could remove them from the other list as required. They can make moving though your List very easy....and could be used in the << operator, = op, and copy constructor.

Make the **it** member variable **mutable** (place key word mutable before the data type in the variable declaration), which means it can change, and not effect the const status of an instance or member function.

```
Example using the iterator member functions:
// assuming DoubleLinkedList list has been created and filled with
data
list.resetIteration( );     // puts iterator at head of list
while( list.hasMore( ) ){   // is there another word in the list
  cout << list.next( );     // get the word and advance the iterator
}
```

resetIteration( )	Internal to your object, points a member pointer to the first node if there is one, otherwise points to nullptr
next( ) : MYString	returns the string where your pointer is pointing and then moves it to the next thing (node or nullptr)
hasMore( ) : bool	returns true if the **it** member pointer is pointing to a node, otherwise it returns false

MYString Class: *Changes*

Programming Note: Write and test one or two functions at a time
All of these functions are doing alphabetic comparisons (comparing your strings one char at a time based on their alphabetic value by temporarily getting and comparing their character in lower (or upper) case). "cat" == "Cat" would be true

Member Functions that potentially need to be changed	**Description**
= = operator : bool	checks to see if the two strings are equal (again “abc” would be equal to “AbC”)
< operator : bool	checks to see if the lvalue is less than the rvalue
> operator : bool	checks to see if the lvalue is greater than the rvalue

! = operator : bool	**OPTIONAL:** check to see if the two string are not equal
>> operator	Change your >> operator so that it will remove **one** trailing punctuation mark (this is the only punctuation that you should remove. Example "plan." would become "plan"). In cctype library there is a function called ispunct(char ch) that returns 0 if ch is not a punctuation or non-zero if it is a punctuation.

Main Requirements: The requirement for your main:

create 4 of objects of your DLL class (list1, list2, modList1, modList2) modList stands for modified list
read all of the data from file1 and insert it into list1
read all of the data from file2 and insert it into list2
cout the size of the 4 lists with explaining text (what size goes with which list)

modList1 = list1
modList2 = list2
cout the size of the 4 lists with explaining text

remove from modList1 all of the strings stored in list2
remove from modList2 all of the strings stored in list1
cout the size of the 4 lists with explaining text

call changer function passing as an argument modList1
call changer function passing as an argument modList2
cout the size of the 4 lists with explaining text

cout the createdCount from the MYString with describing text
cout the currentCount from the MYString with describing text

output modList1 to outfile1.txt with a space between each string
output modList2 to outfile2.txt with a space between each string

Changer Function Requirements:

void changer(DLL list); // prototype....yes it is passing by value, so it can test your copy constructor
it will insert "ZIP" and "ZAP" into the list variable passed into this function
cout the size of the list after the insertions with explaining text {something like: Inside changer function: size of list is 2345 }

Some numbers to verify your code:

As a guide, when you get done reading from infile1.txt, your list1 variable should contain 1104 MYStrings. Your modList1 after removing all the strings from list2, should have 759 MYString still left.

Turn in: A paper copy of your program, (main, list.h, list.cpp, mystring.h and the part of the mystring.cpp that has the additional 3 functions. Your output from the program should follow all of the program listings. For the output, you should have the text that is sent to the screen, and the text that is written to outfile1.txt and outfile2.txt **(in that order)**

Ways to lose points:

- if your file does not contain the program header with a program description
- your .h file should have a class description about what the class does
- your code should also be consistently indented as talked about in class, and shown in the book
- you can't use global variables unless it is a const
- you should use good variable names (descriptive, and start with lower case letter)
- proper placement of { and } (a } should not be placed at the end of a line of code)
- no staple to keep your papers together (folding a corner or using a paper clip are not good enough)
- **Remember the order of your files is main, .h files, .cpp files, and then the output (console window, outfile1.txt, and then outfile2.txt).**

Comments: Comments are a way of documenting a program (explaining who did what and how). All programs for the rest of the course are required to have the following program header documentation (above main, and in any interface files (.h)) and inline documentation to explain any tricky pieces of code.

```
////
// Name: Susy Programmer
// Section: A, B, or S
// Program Name: Hello World
//
// Description: A brief description of the program.  What does the
//  program do (not how it does it: for example, it uses loops)?  Does
//  the program get input?  What kind?  What information is output
//  from the program and to where (screen or file)
////

#include <…>

.......the rest of the program
```