CS 215 – Fall 2018
Project 4
Due: Wed Dec 12 midnight – no late submissions accepted

Learning Objectives:
- completion of the implementation of a Linked List

General Description:
Starting with your completed code for Lab 9, along with an extended main.cpp provided on the course website, complete the Buddy List App by adding functionality for:
- delete (drop) a buddy from the list
- sort the list by last/first name
- read the list from a file (this method is given on the course website)

Specifications:
main.cpp:
- Download the new version from the course website, under the Proj 4 item.
- Put your name and section in the comment box at the top.

buddy.h:
- no changes

buddyList.h:
- add the following prototypes in the **public** section:

```
int  drop(string fname, string lname);
void sort();
void read();
```

- add the following prototypes in the **private** section:

```
buddy* search(string fname, string lname);
buddy* maxByName();
void   remove(buddy * r);
```

buddyList.cpp:

`buddy* search(string fname, string lname)`
Given a first and last name, this method returns a pointer to a node (buddy) in the list with a matching first and last name, or NULL when not found (NULL when list empty).

`buddy* maxByName()`
Search the list for the node (buddy) with the alphabetically lowest name ("Zues" is lower than "Apollo"). The comparison is based on last and first name, so:
if `p->max` is greater than `q->max`, then `p` points to the node with the lower name.
if `p->max` is equal to `q->max` AND `p->first` is greater than `q->first`,
then `p` points to the node with the lower name.
Return a pointer to the node found (NULL when list empty).

`void remove(buddy * r)`
Given a pointer to a node in the list, removes the node from the list, but does NOT deallocate it. See course notes in **Linked Lists** for LList::remove(node * r) for details.

```
int  drop(string fname, string lname)
```
Given a the first and last name of a buddy to drop:

- search() for the name; if not found, return -1 for not found
- remove() the found node
- deallocate the found node (do not leave garbage!)
- return 0 indicating success

```
void read()
```
Reads buddy data from a text file called "buddyList.txt" (found on the course we site), using add() to add the buddies to the list. This code is given on the course website.

```
void sort()
```
Sorts the list from lowest to highest by last name/first name.

Since the Bubble Sort we've used all semester relies heavily on array *indices*, and since Linked Lists do not have indices, Bubble Sort does not work well for Linked Lists.

Instead, implement a **Selection Sort**.

```
create a temporary list (empty)
repeat until the original list is empty:
   o  select the "max" from the original list
   o  remove the "max" from the original list
   o  add the "max" to the temporary list
make the temporary list the original list
```

This example demonstrates this process. For each step, the **max** is highlighted.

Orig	Temp
56	
23	
82	
18	
34	

Orig	Temp
56	82
23	
18	
24	

Orig	Temp
23	56
18	82
24	

Orig	Temp
23	24
18	56
	82

Orig	Temp
18	23
	24
	56
	82

Orig	Temp
	18
	23
	24
	56
	82

Orig	Temp
18	
23	
24	
56	
82	

In this method, you can declare and use a temporary `buddyList` object. Methods operating on "this" original list are directly accessed (along with members); those of the temporary list are accessed inside the temporary object. Example:

```
void buddyList::fun() {
   buddyList temp;
   print();             // prints "this", the original list, using "this"'s head
   temp.print();        // prints the list inside the temp object, using its head
   head = NULL;         // sets the head of "this", the original list to NULL
   temp.head = NULL;    // sets the head inside the temp object to NULL
 }
```

Since these are Linked Lists, it is VERY important that, at the end, when the sorted list is inside the temp object and "this" list is empty (head==NULL) to "transfer" the list from the temp back to the original and "disconnect" the temporary's head; otherwise, the entire list will be destroyed by temp's destructor upon exit from this method.

```
head = temp.head;   // "this" head points to the list pointed to by temp's head
temp.head = NULL;   // disconnect temp's head from the linked list
```

Submission:
Zip all .h and .cpp files into a .zip file and submit the .zip file in Canvas.

Due Date:
Wed Dec 12, midnight
NO LATE SUBMISSIONS ACCEPTED!!