Lab 13 - Linked List Class

Goal: write a C++ class to implement a linked list

---

(0) We could define a linked list of elephants, where each node is:

```
// node.h

typedef
struct anElephant
{
    short                  age;
    string                 name;
    struct anElephant  *next;

} ELEPHANT;
```

See `node.h`

---

(1) Make a (test) driver (console project), `main.cpp`, to do the following:

- (do NOT worry about a class at this point)
- `#include "node.h"`
- declare a (local variable) **`head`** pointer and set up a buffer node
  (see the drawing on the board for help)

- declare three (local but dynamically allocated) temporary elephant
  nodes in main(); use `new()` directly here
- assign each node an elephant name, e.g., "Elmer", “Edna”, “Eloise”
- link the nodes together
- write a loop to traverse the list (starting at head)
  and printing each name as you go

CALL ME OVER WHEN THIS IS WORKING

---

(2) Start to implement a class to handle a List. See my `eList.h` file in the starter kit and also the API documentation on the last page of this lab handout. Obviously, begin by making a new file called `eList.cpp`. Begin to implement your <u>link list class</u> in these files. In this step (#2), just write the public constructor (CTOR).

Since you may need the functionality of creating a “new node” for a list more than just in this CTOR, write a method called `create()` which will allocate memory for a new node, initialize it, and return the address of its location on the HEAP. Since you might know the elephant’s name prior to create(), **overload** your create() function to work in either case: (i) you don’t know the name and (ii) you do know the name (before creation). Thus, the prototypes for `create()` will be:

```
ELEPHANT*  create(void);
ELEPHANT*  create(string newName);
```

Modify your main driver to use the CTOR: `ElephantList L;`

CALL ME OVER WHEN THIS IS WORKING

---

(3) Implement `PrintAll()` ... then test in `main()`

```
ElephantList L;
     :
     :
L.PrintAll(); // can't really test this until you do #4 below
```

---

(4) Implement `InsertFront()` (see PRE/POST below)

```
ElephantList L;

  // in main(), make a temporary node (tempNode) filled with
  // data like in part1 above; then insert it onto your list
  // (see below) and print out the list

  L.InsertFront(tempNode);
  L.PrintAll();

// now add a couple of more nodes and then PrintAll()again ...
```

---

(6) Implement the DTOR; ***careful*** .....

---

(7) Implement `Insert()` and `Delete()` (see PRE/POST on next page)

Here are the specifications for some of the List class functions

```
//=========================================================
 ~ElephantList();
//.................................................................
// POST: all data nodes on List are freed(deleted)


//.................................................................
bool IsEmpty() const;
//.................................................................
// POST: Return true if list is empty; false otherwise
// Note: This list always keeps a leading empty buffer node; this
//       node is not part of the list; rather, buffer->next points
//       to the actual initial element on the linked list


//.................................................................
void InsertFront( /* in */ ELEPHANT* newNode );
//.................................................................
// PRE:  Assigned(newNode)
// POST: newNode inserted at the very front of the list


//.................................................................
void Insert( /* in */ ELEPHANT* newNode );
//.................................................................
// PRE:  Assigned(newNode)
// POST: newNode inserted alphabetically by name


//.................................................................
void Delete(/* in */ string thisName);
//.................................................................
// PRE:  not Empty() && Assigned(thisName)
// POST: find node with this name and delete the node if found


//.................................................................
void PrintAll();
//.................................................................
// PRE:  none
// POST: entire list printed to stdout (buffer node is not printed)


//.................................................................
hmmm, rather than PrintAll(), how about this ...
friend ostream& operator<<(ostream& out, const ElephantList& L);
```