CS302

Assignment 1: Bubble Sort on a doubly linked list

Description

Sorting. One of the celebrated topics in computer science and which we will spend quite a bit of time during the earlier part of the semester. You will implement bubble sort, probably the first sorting algorithm to which you were introduced back in CS135. Except we will use a linked list and use iterators just to give you a very pleasant welcome to CS302. The next section goes over the data type you will need to implement.

LL Class

```
template <typename type>
class LL
{
  struct node
  {
    type data;
    node * prev;
    node * next;
  };

public:
  class iterator
  {
  public:
    friend class LL;
    iterator();
    iterator(node*);
    type operator*() const;
    const iterator& operator++(int);
    const iterator& operator--(int);
    bool operator==(const iterator&) const;
    bool operator!=(const iterator&) const;
  private:
    node * current;
  };

  LL();
  LL(const LL<type>&);
  const LL<type>& operator=(const LL<type>&);
  ~LL();
  void tailInsert(const type&);
  iterator begin() const;
  iterator end() const;
  void swapNodes(iterator&, iterator&);
private:
  node * head;
  node * tail;
};
```

Each member of the iterator class contains/performs

- `node * current` - a pointer that contains an address of a node in the linked list (this pointer denotes which node the iterator points to)
- `LL<type>::iterator::iterator()` - default constructor, sets current with `NULL`
- `LL<type>::iterator::iterator(node * ptr)` - constructor that assigns ptr to current
- `type LL<type>::iterator::operator*() const` - overloads the iterator's derefence operator, returns `current->data`
- `const typename LL<type>::iterator& LL<type>::iterator::operator++(int)` - moves the iterator over one node to the right
- `const typename LL<type>::iterator& LL<type>::iterator::operator--(int)` - moves the iterator over one node to the left
- `bool LL<type>::iterator::operator==(const iterator& rhs) const` - returns `true` if `this` iterator and `rhs` iterator point to the same node, returns `false` otherwise
- `bool LL<type>::iterator::operator!=(const iterator& rhs) const` - returns `true` if `this` iterator and the `rhs` iterator do not point to the same node, returns `false` otherwise

Each member of the LL class contains/performs

- `struct node` - the `LL` will contain a set of nodes to implement the linked list, it will be a doubly linked list
- `node * head` - pointer that points to the front node
- `node * tail` - pointer that points to the end node
- `LL<type>::LL()` - default constructor, sets head and tail to `NULL`
- `LL<type>::LL(const LL<type>& copy)` - deep copy constructor, performs a deep copy of the `copy` linked list into the `this` linked list
- `const LL<type>& LL<type>::operator=(const LL<type>& rhs)` - deep copy assignment operator, performs a deep copy of the `rhs` object into the `this` object, remember to check for a self assignment, deallocate the `this` linked list before performing the actual deep copy, and at the end return `*this`
- `LL<type>::~LL()` - destructor, deallocates the entire linked list
- `void LL<type>::headInsert(const type& item)` - performs a head insert, item will be contained in the data field of the new head node
- `void LL<type>::tailInsert(const type& item)` - performs a tail insert, item will be contained in the data field of the new tail node
- `typename LL<type>::iterator LL<type>::begin() const` - returns an `iterator` object whose current is set to the head pointer
- `typename LL<type>::iterator LL<type>::end() const` - returns an `iterator` object whose current is set to the tail pointer
- `void LL<type>::swapNodes(iterator& it1, iterator& it2)` - swaps the location of the nodes referenced by it1 and it2, do not just swap the data fields of the node within it1.current and it2.current, you need swap the two nodes geographic locations (by setting A LOT of next and prev pointers), make sure you handle the edge cases (for example if it1 or it2 point to the head or tail...)

Contents of main

In main, you will read in a filename from the user, open a filestream and read the contents from a file and store them into a linked list. The contents of the file will be a set of integers (one integer per line), thus you will need to allocate a LL<int> object. You need to use bubble sort to sort the linked list using LL<int>::iterator objects to navigate the linked list, using ++ and – operators accordingly. The following code can give you an idea of how to use iterators

```
LL<int> list;
LL<int>::iterator it;
LL<int>::iterator nil(NULL);

list.tailInsert(1);
list.tailInsert(2);
list.tailInsert(3);
list.tailInsert(4);
list.tailInsert(5);

it = list.begin();

while (it != nil)
{
  std::cout << *it << endl;
  it++;
}
```

The code at the beginning just inserts integers 1-5 into a linked list and the for loop just traverses each node and outputs each node's item field. The following link goes over the array code and explanation of bubble sort https://www.geeksforgeeks.org/bubble-sort/. You would need to write a similar algorithm for linked lists, you would need to use the LL class swapNodes function. You cannot use a counter controlled loop to implement bubble sort on a linked list, since we are not indexing an array, there's no real point to use a counter anywhere.

Specifications

- Comment your code and functions
- No counter controlled loops allowed
- Do not modify the member functions or any of the class variables
- Make your code memory leak free
- Make sure when you're swapping nodes in the linked list, actually swap the node locations and not just the data fields of the two nodes

Sample Run

```
Enter file with list: small.txt
Original List
10 5 9 3 7 12 2 4 11 1

Sorted List
1 2 3 4 5 7 9 10 11 12
```

Submission

Submit Assignment01.cpp and any header files to the codegrade submission by the deadline