CS302

Assignment 1: Cocktail Shaker Sort Using a Linked List

Description

Sorting. Arranging the elements so that the elements increase in ascending order. Such a simple yet complex topic. Of course the problem becomes more complex since we are going to use a doubly linked list and iterators to maintain and manipulate the list.

```
template <typename T>
class LL
{
  //contents of each node
  struct Node
  {
    T data;
    Node* prev;
    Node* next;
  };
//iterator class to allow access of each node in main
public:
  class Iterator
  {
  public:
    friend class LL;
    Iterator();
    Iterator(Node*);
    T 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<T>&);
  const LL<T>& operator=(const LL<T>&);
  ~LL();
  void headInsert(const T&);
  void tailInsert(const T&);
```

```
  Iterator begin() const;
  Iterator end() const;
  void swapNodes(Iterator&, Iterator&);

private:
  Node* head;
  Node* tail;
};
```

Each member of the `Iterator` class performs/contains the following

- `Node * current` - a pointer that contains the address of the node that the Iterator object points to
- `LL<T>::Iterator::Iterator()` - default constructor that sets `current` with `NULL` or `nullptr`
- `LL<T>::Iterator::Iterator(Node* ptr)` - constructor that sets `current = ptr`
- `T LL<T>::Iterator::operator*() const` - overloads the dereference operator, just returns the data field of the node the Iterator object is pointing to
- `const typename LL<T>::Iterator& LL<T>::Iterator::operator++(int)` - postfix ++ operator that moves the Iterator object one node over to the right
- `const typename LL<T>::Iterator& LL<T>::Iterator::operator--(int)` - postfix – operator that moves the Iterator object one node over to the left
- `bool LL<T>::Iterator::operator==(const Iterator& rhs) const` - comparison operator, compares if the `*this` Iterator and the `rhs` Iterator point to the same node, if they do return `true` else return `false`
- `bool LL<T>::Iterator::operator!=(const Iterator& rhs) const` - comparison operator, compares if the `*this` Iterator and the `rhs` Iterator point to a different node, if they point to different nodes return `true` else return `false`

Each member of the `LL` class performs/contains the following

- `struct Node` - a struct object that contains an element in the list and a pointer for its left and right neighbor within the list
- `Node * head` - head pointer, points to the start of the list (the leftmost node)
- `Node * tail` - tail pointer, points to thee end of the list (the rightmost node)
- `LL<T>::LL()` - default constructor, assigns the head and tail with `NULL` or `nullptr`
- `LL<T>::LL(const LL<T>& copy)` - deep copy constructor, deep copies the `copy` object into the `*this` object
- `const LL<T>& LL<T>::operator=(const LL<T>& rhs)` - deep copy assignment operator, deep copies the `rhs` object into the `*this` object, make sure you deallocate the `*this` object first before performing the deep copy, also check for self assignment, then `return *this` at the end
- `LL<T>::~LL()` - destructor, deallocates the entire linked list
- `void LL<T>::headInsert(const T& item)` - insert a new node to the front of the linked list and this node's data field must contain the contents in the item parameter
- `void LL<T>::tailInsert(const T& item)` - insert a new node to the back of the linked list and this node's data field must contain the contents in the item parameter
- `typename LL<T>::Iterator LL<T>::begin() const` - returns an `Iterator` object whose current field contains `this->head`

- `typename LL<T>::Iterator LL<T>::end() const` - returns an `Iterator` object whose current field contains `this->tail`
- `void LL<type>::swapNodes(Iterator& it1, Iterator& it2)` - swap the location of the node `it1.current` with the location `it2.current`, you cannot just swap the data fields, you need to modify prev/next pointers to actually physically move the two nodes in the list, watch the supplemental video for a more detailed explanation

Input

A list of integers (one integer per line), each line is terminated with an end of line character, you would need to have an `LL<int>` object declared and then do a tail insert for each element read from the file

Output

You need to output a sorted list in ascending order, each element needs to be separated by a white space so code grade would be able to compare you answers

Contents Of Main

Once you read in the contents from the file and insert them into a linked list, you need to implement the brick sort algorithm, a detailed explanation can be found here

$https://en.wikipedia.org/wiki/Cocktail_shaker_sort\#:~:text=Cocktail\%20shaker\%20sort\%2C\%20also\%20known\%20$

The catch is that you need to use Iterators to traverse the list and compare and swap elements. You can not use any form of a counter controlled loop (a `for` loop is fine but it cannot be counter based), thus you need to have a few dedicated Iterator objects that act as a sentinel value stop the inner or outer loops (which either ends the algorithm or ends a an even/odd phase)

Specifications

- Must use `LL<int>` and `LL<int>::Iterator` objects to manipulate the list
- No counter controlled loops
- Have your code well documented
- Make sure your code is memory leak free

Sample Run

```
If the list is sorted then it worked, no need to
put an actual output here
```

Submission

Submit the source files to code grade by the deadline

References

- Supplemental Video https://youtu.be/4Vze36ic_Os
- Link to the top image can be found at https://pngimg.com/image/44649