CS302

Assignment 1: Selection Sort On A Linked List

Description

Sorting. The simplest complicated concept of all of computer science. You will be given a set of integers, and your taks will be to implement selection sort on the list. The pseudo code for selection sort is can be found at *https://en.wikipedia.org/wiki/Selection_sort*.

The catch here is we are going to store the list of elements into a doubly linked list, and when we swap elements, we will swap the node locations, not the data store in the node.

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 headInsert(const type&);
  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 (if it1 and it2 are adjacent, or one or both point to the head/tail...)

Contents of main

In main, you declare an `LL<int>` object, then read in an input filename and then using an ifstream variable read a number from the file and populate the linked list, then you perform selection sort, by finding the max element and swap it to the end, and the find the max again (excluding the last node), and swap that to the second to last position in the linked list and so on. Then output the sorted linked list.

Specifications

- Comment your code and functions
- 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

```
$ g++ Assignment01.cpp
$ ./a.out i1.txt

1
16
17
49
80
81
82
87
98
98
```

Submission

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