CS302

Assignment 2: Selection Sort

Description

For this assignment, you will read in a list of random numbers from a file and build a doubly-linked list and perform selection sort on the list, you will need to implement the following linked list class along with its iterator

```
template <class Type>
class LL
{
  struct node
  {
    Type item;
    node * next;
    node * prev;
  };

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

  LL();
  LL(const LL&);
```

```
  const LL& operator=(const LL&);
  ~LL();
  iterator begin() const;
  iterator end() const;
  void headRemove();
  void tailRemove();
  bool isEmpty() const;
  void headInsert(const Type&);
  void tailInsert(const Type&);
  void update(const iterator&, const Type&);
private:
  node * head;
  node * tail;
};
```

Each member of the LL class contains/performs the following

- `struct node` - will be each member of the linked list, it contains an item, a pointer to the right node, and a pointer to the left node
- `node * head` - is a pointer that points to the first node in the linked list
- `node * tail` - is a pointer that points to the last node in the linked list
- `LL()` - is the default constructor that sets an empty linked list
- `LL(const LL&)` - is the copy constructor, performs a deep copy of the LL object passed in
- `const LL& operator=(const LL&)` - is the assignment operator, performs a deep copy of the LL object passed in (the object on the right hand side of the assignment operator)
- `~LL()` - is the destructor, deallocates the linked list
- `iterator begin)_ const` - returns an iterator object that points to the first node in the list
- `iterator end() const` - returns an iterator object that points to the last node in the list
- `void headRemove()` - removes the front node in the linked list
- `void tailRemove()` - removes the end node in the linked list
- `bool isEmpty() const` - returns `true` if the list is empty and `false` otherwise
- `void headInsert(const Type&)` - inserts a new node to the front of the list, sets this node's `item` field to the value passed into the function
- `void tailInsert(const Type&)` - inserts a new node to the end of the list, sets this node's `item` field to the value passed into the function
- `void update(const iterator&, const Type&)` - reassigns a value of the node that the iterator object points to with the value passed in (the second parameter)

Each member of the iterator class contains/performs the following

- `node * current` - is a pointer that points to a node in the linked list
- `iterator()` - default constructor that sets `current` to `NULL`
- `iterator(node*)` - a constructor that sets `current` to the pointer passed into the function
- `Type operator*()` - overloads a unary operator, returns the `item` field of the `node` object that `cuurent` points to

- `iterator operator++(int)` - overloads a unary operator, moves the `current` pointer over to the next node of the linked list
- `iterator operator--(int)` - overloads a unary operator, moves the `current` pointer over to the previous node of the linked list
- `bool operator==(const iterator&) const` - overloads the binary equals operator, returns `true` if the `iterator` on the left side of the operator points to the same node as the `iterator` on the right side, and returns `false` otherwise
- `bool operator!=(const iterator&) const` - overloads the binary equals operator, returns `false` if the `iterator` on the left side of the operator points to the same node as the `iterator` on the right side, and returns `false` otherwise

The `iterator` object will basically serves as a pointer, you will use it to navigate through the linked list, suppose you create a linked list object of type `int`, and you inserted content to the front or the back and wish to traverse and print out its contents, the following code can be used

```
LL<int> list;
LL<int>::iterator it;

//Assuming you inserted nodes into the list at some point

//this loop starts the iterator by setting it to the front of the linked list
//the loop will keep running as long as the iterator hasn't reached the last node
//it will move to the next node each time
for (it = list.begin(); it != list.end(); it++)
{
  cout << *it << endl; //outputs the value in the node that it points to
}
//since the loop doesn't output the last node since it stops right when it hits the
//last node, we output it here
cout << *it << endl;
```

Contents of main

You will be given an input file with a set of integers, you will read through the file and insert each item into the linked list object using the head or tail insert functions. Then you will implement a known sorting algorithm, selection sort. The way this algorithm works is it first finds the max element in the list and then swaps the item with the last element in the list. Then you find the max element in the array (not including the last element), then you swap this max element to the second to last element, and so on until the list is finally sorted. This is not the most efficient sorting algorithm but it works, you would need to have several `iterator` objects, once to traverse the list, one to point to the current max element, and one at the end (to denote the element to be swapped with once the max is found)

Specifications

- Comment your code and your functions
- Do not add extra class members or remove class members and do not modify the member functions of the class
- Do not use global variables
- Make sure your program is memory leak free

Example Output

Once your compile and run, your program should output a sorted list, it would take too many pages of the pdf to output that but you know if your program works: if it outputs a sorted list, then it worked, if not then it didn't work 😄

Submission

Upload your source files `LL.h` and `main.cpp` to the moodle site by the deadline