Assignment 7: Linked List Recursion

Collaboration Policy

You may not use code from any source (another student, a book, online, etc.) within your solution to this assignment. In fact, you may not even look at another student's solution or partial solution to this assignment. You also may not allow another student to look at any part of your solution to this exercise. You should get help on this assignment by asking the instructor or or by posting questions on Canvas (you still must not post assignment code publically on Canvas.) See the full Course Collaboration Policy here: Collaboration Policy

IntList Recursion Assignment Specifications:

You will add some additional recursive functions to your IntList class as well as make sure the Big 3 are defined.

IntNode class

I am providing the IntNode class you are **required** to use. Place this class definition within the IntList.h file exactly as is. Make sure you place it above the definition of your IntList class. Notice that you will not code an implementation file for the IntNode class. The IntNode constructor has been defined inline (within the class declaration). Do not write any other functions for the IntNode class. Use as is.

```
struct IntNode
{
    int data;
    IntNode *next;
    IntNode( int data ) : data(data), next(0) {}
};
```

IntList class

Encapsulated (Private) Data Fields

- head: IntNode *
- tail: IntNode *

Public Interface (Public Member Functions)

- IntList()
- IntList(const IntList &list)
- ~IntList()
- void display() const
- void push_front(int value)
- void push_back(int value)
- void pop_front()
- void select_sort()
- void insert_sorted(int value)
- void remove_duplicates()
- IntListIterator begin()
- IntListIterator end()
- int front() const
- int back() const
- int length() const;
- int sum() const;
- void reverseDisplay() const;
- IntList & operator=(const IntList &list)

<u>Constructor and Destructor</u>

IntList() - the default constructor

Initialize an empty list.

IntList(const IntList &list) - the overloaded copy constructor

Initialize a new list with the contents of an existing list.

~IntList()

This function should deallocate all remaining dynamically allocated memory (all remaining IntNodes).

<u>Accessors</u>

void display() const

This function displays to a single line all of the int values stored in the list, each separated by a space. It should **NOT** output a newline or space at the end.

intListIterator begin()

This function returns an iterator at the beginning of the linked list. Returns an iterator pointing to head.

intListIterator end()

This function returns an iterator one element past the last element of the linked list. Returns an iterator pointing to NULL.

int front() const

This function returns the data in the head of the linked list.

int back() const

This function returns the data in the tail of the linked list.

int length() const

This function recursively determines the length of the list.

int sum() const

This function recursively determines the sum of all of the elements in the list.

void reverseDisplay() const

This function recursively displays the contents of the list in reverse order.

Mutators

void push_front(int value)

This function inserts a data value (within a new node) at the front end of the list.

void push_back(int value)

This function inserts a data value (within a new node) at the back end of the list.

void pop_front()

This function removes the value (actually removes the node that contains the value) at the front end of the list. Do nothing if the list is already empty. In other words, do not call the exit function in this function as we did with the IntVector's pop_front.

void select_sort( )

This function sorts the list into ascending order using the **selection sort** algorithm.

void insert_sorted(int value)

This function assumes the values in the list are in sorted (ascending) order and inserts the data into the appropriate position in the list (so that the values will still be in ascending order after insertion). **DO NOT** call select_sort within this function.

void remove_duplicates()

This function removes all values (actually removes the nodes that contain the value) that are duplicates of a value that already exists in the list. Always remove the later duplicate, not the first instance of the duplicate. **DO NOT** call select_sort within this function. This function does **NOT** assume the data is sorted.

IntList & operator=(const IntList &list)

This function copies over all of the nodes in an existing list to another already existing list.

IntListIterator class

Encapsulated (Private) Data Fields

- current: IntNode *

Public Interface (Public Member Functions)

- IntListIterator()
- IntListIterator(IntNode *ptr)
- int operator*()
- intListIterator operator++()
- bool operator==(const intListIterator& right) const;
- bool operator!=(const intListIterator& right) const;

Constructors

IntListIterator() - the default constructor

Initialize the iterator. Basically just need to set the pointer to NULL.

IntListIterator(intNode *ptr) - the overloaded copy constructor

Initialize the iterator with parameter passed int. Need to set the pointer equal to whatever pointer is passed in.

Accessors

int operator*()

This function overloads the dereferencing operator*. It should return the info contained in the node.

intListIterator operator++()

This function overloads the pre-increment operator++. It should return an iterator that is pointing to the next node.

bool operator==(const intListIterator& right) const;

This function overloads the equality operator. Should return true if this iterator is equal to the iterator specified by right, otherwise it returns false.

bool operator!=(const intListIterator& right) const;

This function overloads the not equal to operator. Should return true if this iterator is not equal to the iterator specified by right, otherwise it returns false.

Private Helper Functions

You may define any **private** helper functions you deem useful, provided they do not affect the efficiency of the problem they are used to solve. Be careful making any function that must traverse the list to get to the node it will be working on. A private helper function that does this will almost always cause the function it is helping to become less efficient. You may lose points for that. For example, **DO NOT** make a function that returns the size of the list.

You **MAY NOT** define any other data fields, public or private, for this assignment.

What to Submit

Note: Your files should compile with the command **`g++ -std=c++11 *.cpp`**
In Canvas, submit the following files (**case sensitive**):

- main.cpp (test harness)
- intNode.h
- intList.h
- intList.cpp
- intListIterator.h
- intListIterator.cpp
