Assignment 6: Linked List

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 coming to the instructor's or TA's office hours or by posting questions on Piazza (you still must not post assignment code publically on Piazza.) See the full Course Collaboration Policy here: Collaboration Policy

---

IntList Assignment Specifications:

You will finish the implementation of the singly-linked list we began in lecture and lab.

You are required to come up with a single header file (IntList.h) that declares and implements the IntNode class (just copy it exactly as it is below) as well as declares the IntList Class interface only. You also are required to come up with a separate implementation file (IntList.cpp) that implements the member functions of the IntList class. While developing your IntList class you must write your own test harness (main function). You may lose points if you do not include a thorough test harness in your main function. Never implement more than 1 or 2 member functions without fulling testing them with your own test harness. It is ok to comment out parts of the test you don't want run on any particular test run, but never delete valid tests. Fixing one problem often makes old tests no longer work. You should always test everything over again in the end to make sure you didn't break something that was working previously.

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 Inteface (Public Member Functions)

- IntList()
- ~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()

Constructor and Destructor

IntList() - the default constructor

Initialize an empty list.

~IntList()

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

Accessors

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.

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_back.

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.

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.

Compiling and Testing

You can compile the entire program, generating an executable named a.out, with the following command line:

```
g++ -W -Wall -Werror -ansi -pedantic main.cpp IntList.cpp
```

But when you want to compile the class seperately, without generating an executable, use the command line:

```
g++ -W -Wall -Werror -ansi -pedantic -c IntList.cpp
```

And then to generate the executable once you've already compiled the IntVector class separately, use the command line:

```
g++ -W -Wall -Werror -ansi -pedantic main.cpp IntList.o
```

What to Submit

For this assignment you will turn in the following files (case sensitive) to R'Sub (galah.cs.ucr.edu):

- main.cpp (test harness)
- IntList.h
- IntList.cpp
