CS215, Introduction to Program Design, Abstraction, and Problem Solving
Fall 2020

Programming Assignment #4

SPECIAL NOTE: this is a double-value assignment. It will count towards your final semester grade twice as much as a single-value assignment such as Programming Assignment #1.

Design, write, execute, and test a C++ program that will demonstrate the use of several linked list methods by allowing the user to work with an ongoing, "current" list by repeatedly choosing actions from the following menu:

```
f  Reset the current list from the keyboard in forward order
b  Reset the current list from the keyboard in backward order
e  Reset the current list to be empty
i  Insert a given element at a given position in the current list
d  Delete the element at a given position in the current list
c  Combine the current list with a given list
r  Reverse the elements in the current list
h  Help the user by displaying these menu choices
q  Quit the program
```

Here are some requirements for your program:

- For the purposes of this program you should assume that *elements* are *ints*.
- Your program should be able to handle lists with any number of *elements*.
- When your program starts executing, it should automatically start with a current list of 0 *elements*; in other words, the current list should start off as an empty list.
- Before displaying the menu each time, the program should display the current list. If the current list is empty, then the phrase "(empty)" should be displayed.
- Whenever the user chooses to reset the current list from the keyboard, the user should then be allowed to enter in elements from the keyboard, signaling the end of the input by entering the "sentinel" element of -1.
- Whenever the user chooses to insert a given element at a given position in the list, the position should be given as a positive integer, where 1 signifies the first place on the list, 2 signifies the 2nd place, etc. If that position does not or would not exist on the list, then a warning message should be displayed and the list should be left unchanged.
- Whenever the user chooses to delete a given element at a given position in the list, the position should be given as a positive integer, where 1 signifies the first place on the list, 2 signifies the

2nd place, etc. If that position does not exist on the list, then a warning message should be displayed and the list should be left unchanged.

- All inputs given by the user to your program should undergo data validation to ensure that they are correct in data type and range before being used.

- You should write this program by extending the *LList* class that we recently studied in our CS215 lecture. At a minimum, your program should include the vast majority of those *LList* methods and associated parameters. In addition, you should also design, write, and use the following two new methods:

  void LList::InsertAtPosition(element thing, int position)
  inserts a new *listnode* containing *element thing* at the given *position* onto the native object *LList*, or displays a warning message and leaves the native object *LList* unchanged if that *position* does not or would not exist

  void LList::DeleteAtPosition(int position)
  deletes the existing *listnode* at the given *position* on the native object *LList*, or displays a warning message and leaves the native object *LList* unchanged if that *position* does not exist

- In order to make your main function relatively modular and easy to read, you should also include additional methods and/or functions as necessary.

- Your program should ensure that all *LList* objects stay valid during their entire lifetimes.

- Other than objects of type LList, you should not use arrays or any other form of repetitive data structures in this assignment.

Here are some hints to make your coding easier:

- You can assume that there will always be sufficient memory to allocate storage for new listnodes in your linked lists.

- You should code and test each of the above methods separately. When they are working properly, then you should combine them into the overall program.

- You should be very careful using pointers! Many programming errors can be traced to the incorrect use of pointers that are undefined (for example, that are uninitialized or are dangling). Be sure that each pointer you are using to access a *listnode* has directly or indirectly been initialized by using the *new* operator, and that it isn't *NULL*, and isn't dangling due to premature use of the *delete* operator. If you seem to be having trouble getting your pointers to work, a good debugging technique is to draw some sample *LList* objects by hand, and then trace through your code step-by-step looking for places where your pointers seem to go astray.

Be sure to follow the overall requirements for programming assignments found in the document called "Programming Requirements" posted on the CS215 Blackboard Course Shell. You'll need to develop and run your own set of proposed testcases that are sufficient in quantity, variety, and thoroughness to

show that your program works correctly in all situations. (Note: the instructor will not be providing required testcases for this assignment.)

To complete this programming assignment, submit your Listing and Executions as a single PDF document attached to the appropriate submission link on the CS215 Blackboard Course Shell by the "Listing and Executions" due date stated there. (NOTE: You should create the PDF document for your Listing and Executions by using the CSTools “listexec” command on the Grace Linux computer system.)

On the following page are samples of what your program executions should look like when completed.

Good luck!

```
Linked List Demo Program, version 1.0
(c) 2020, (your name)

Current list: (empty)

Command (h for help): h

Valid commands are:
   f  Reset the current list from the keyboard in forward order
   b  Reset the current list from the keyboard in backward order
   e  Reset the current list to be empty
   i  Insert a given element at a given position in the current list
   d  Delete the element at a given position in the current list
   c  Combine the current list with a given list
   r  Reverse the elements in the current list
   h  Help the user by displaying these menu choices
   q  Quit the program

Current list: (empty)

Command (h for help): f

Resetting the current list from the keyboard in a forward order.

Enter a series of elements, -1 to stop: 6 4 2 9 7 3 -1

Finished resetting.

Current list: 6 4 2 9 7 3

Command (h for help): b

Resetting the current list from the keyboard in a backward order.

Enter a series of elements, -1 to stop: 9 5 7 1 0 8 -1

Finished resetting.

Current list: 8 0 1 7 5 9

Command (h for help): e

Resetting the current list to be empty.

Current list: (empty)

Command (h for help): f

Resetting the current list from the keyboard in a forward order.

Enter a series of elements, -1 to stop: 2 7 6 3 9 5 4 -1

Finished resetting.

Current list: 2 7 6 3 9 5 4

Command (h for help): i

Inserting a given element at a given position on the current list.

Element to insert: 8

Position to insert at: 4
```

```
Finished inserting.
Current list: 2 7 6 8 3 9 5 4
Command (h for help): i
Inserting a given element at a given position on the current list.
Element to insert: 1
Position to insert at: 9
Finished inserting.
Current list: 2 7 6 8 3 9 5 4 1
Command (h for help): i
Inserting a given element at a given position on the current list.
Element to insert: 0
Position to insert at: 11
Sorry, position 11 does not or would not exist on the current list.
Finished inserting.
Current list: 2 7 6 8 3 9 5 4 1
Command (h for help): d
Deleting the element at a given position on the current list.
Position to delete at: 3
Finished deleting.
Current list: 2 7 8 3 9 5 4 1
Command (h for help): d
Deleting the element at a given position on the current list.
Position to delete at: 9
Sorry, position 9 does not exist on the current list.
Finished deleting.
Current list: 2 7 8 3 9 5 4 1
Command (h for help): c
Combining the current list with a given list.
Enter a series of elements, -1 to stop: 5 2 8 6 -1
New list: 5 2 8 6
Finished combining.
```

```
Current list: 2 7 8 3 9 5 4 1 5 2 8 6

Command (h for help): r

Reversing the current list.

Current list: 6 8 2 5 1 4 5 9 3 8 7 2

Command (h for help): q

Finishing Linked List Demo Program, version 1.0
```