CSCE 221 Assignment 3 Cover Page

First Name Jonathan Last Name Westerfield UIN 224005649

User Name jgwesterfield E-mail address jgwesterfield@tamu.edu

Please list all sources in the table below including web pages which you used to solve or implement the current homework. If you fail to cite sources you can get a lower number of points or even zero, read more on Aggie Honor System Office website:

Type of sources	Peer Teacher			
People	Lauren Kleckner			
Web pages (provide URL)				
Printed material				
Other Sources				

I certify that I have listed all the sources that I used to develop the solutions/codes to the submitted work.
On my honor as an Aggie, I have neither given nor received any unauthorized help on this academic work.

Your Name Jonathan Westerfield Date 7/27/17

CSCE 221 Assignment 3
Summer 2017

due to eCampus by July 29th, and demonstration of Part 1 in labs on July 24th.

Objective

This is an individual assignment which has two parts.

1. Part 1: C++ implementation of DoublyLinkedList for and generic types based on the provided supplementary code.
2. Part 2: C++ implementation of data structure that can store *comparable elements.*

Part 1: Implementation of DoublyLinkedList

1. Untar supplementary code . Use the 7-zip software to extract the files in Windows, or use the following command in Linux.

2. for integers
   (a) Most of the code is extracted from the lecture slides. An exception structure is defined to complete the program.
   (b) You need to complete the functions which are declared in the header file .
   (c) Type the following commands to compile the program

   (d) The main program includes examples of creating doubly linked lists, and demonstrates how to use them. Type the following command to run the executable file:

   (e) Test the doubly linked list functions in .

3. Implement a templated version of the class and test the functions for correctness. Follow the instructions below:
   (a) Templates should be declared and defined in a header file. Move the content of and to
   (b) Replace by in the so the list nodes store generic objects instead of integers. Later on, when a object is created, say, in the main function, can be specified as a , or a user-defined class.
   (c) To create a templated class with a generic type , you must replace a declaration/return type by (except for the variable).
      i. To use the generic type , you must change each type declaration.
      ii. Use the generic type anywhere throughout the class .
   (d) Add the keyword before a class declaration.
   (e) In each member function signature, replace by

(f) If a member function is defined outside the class declaration, change the function signature, that is, replace by

(g) To use the generic type anywhere throughout the class and , you must declare (add) before classes and member functions defined outside the class declaration.

4. Compile and run the generic version in a similar way as for type. Type the following commands to compile the program.

5. The main program includes examples of creating doubly linked lists of , and demonstrates how to use them. Type the following command to run the executable.

Part 2: Implementation of MinQueue data structure based on DoublyLinkedList

The data structure should store *the comparable elements* that support the queue operations: , ,, and in addition the operation that returns (but not deletes) the smallest value currently stored in the queue.

Use the adapter design pattern for implementation of that work together with the class defined in the Part 1. The runtime worst case of all operations except should be *constant,* $O(1)$.

The implementation details of the operations, justification of their running time, and tests for correctness should be provided in the part 2 of the report.

What to submit to eCampus?

- Create a directory for the Part 1 that includes: source code for and generic types, typed report with description of the linked list implementation, complexity analysis of code expressed in terms of big-O, and the test cases done for correctness.

Part I

Report for DoublyLinkedList

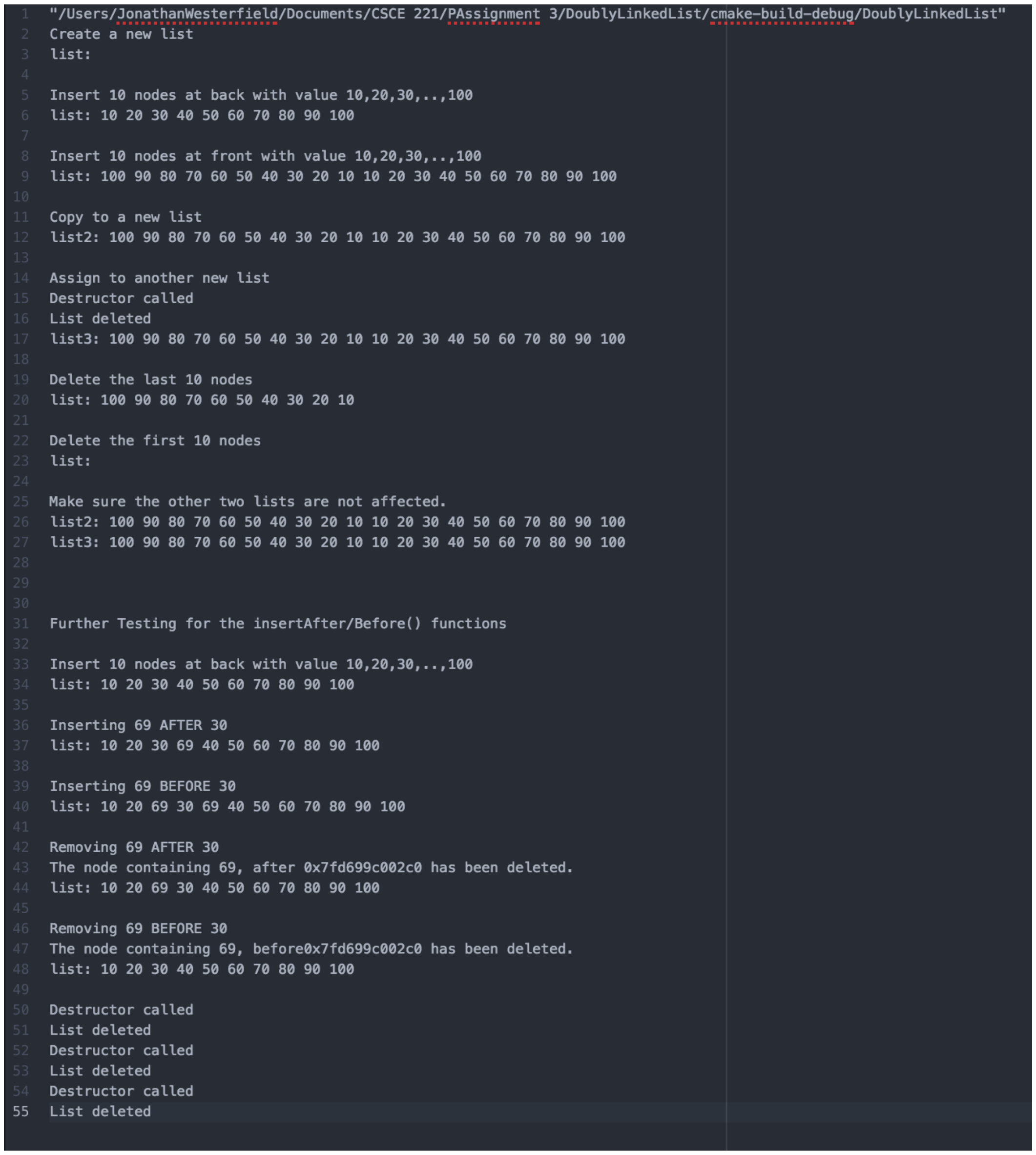
1. To compile the DoublyLinkedList:
   (a) navigate to the folder Submission/Part 1/DoublyLinkedList
   (b) Use the command <g++ -std=c++11 *.cpp -o main>
   (c) Then type <./main> into the terminal to run

2. To compile TemplateDoublyLinkedList:
   (a) navigate to the folder Submission/Part 1/TemplateDoublyLinkedList
   (b) Use the command <g++ -std=c++11 *.cpp -o main>
   (c) Then type <./main> into the terminal to run

3. Implementation

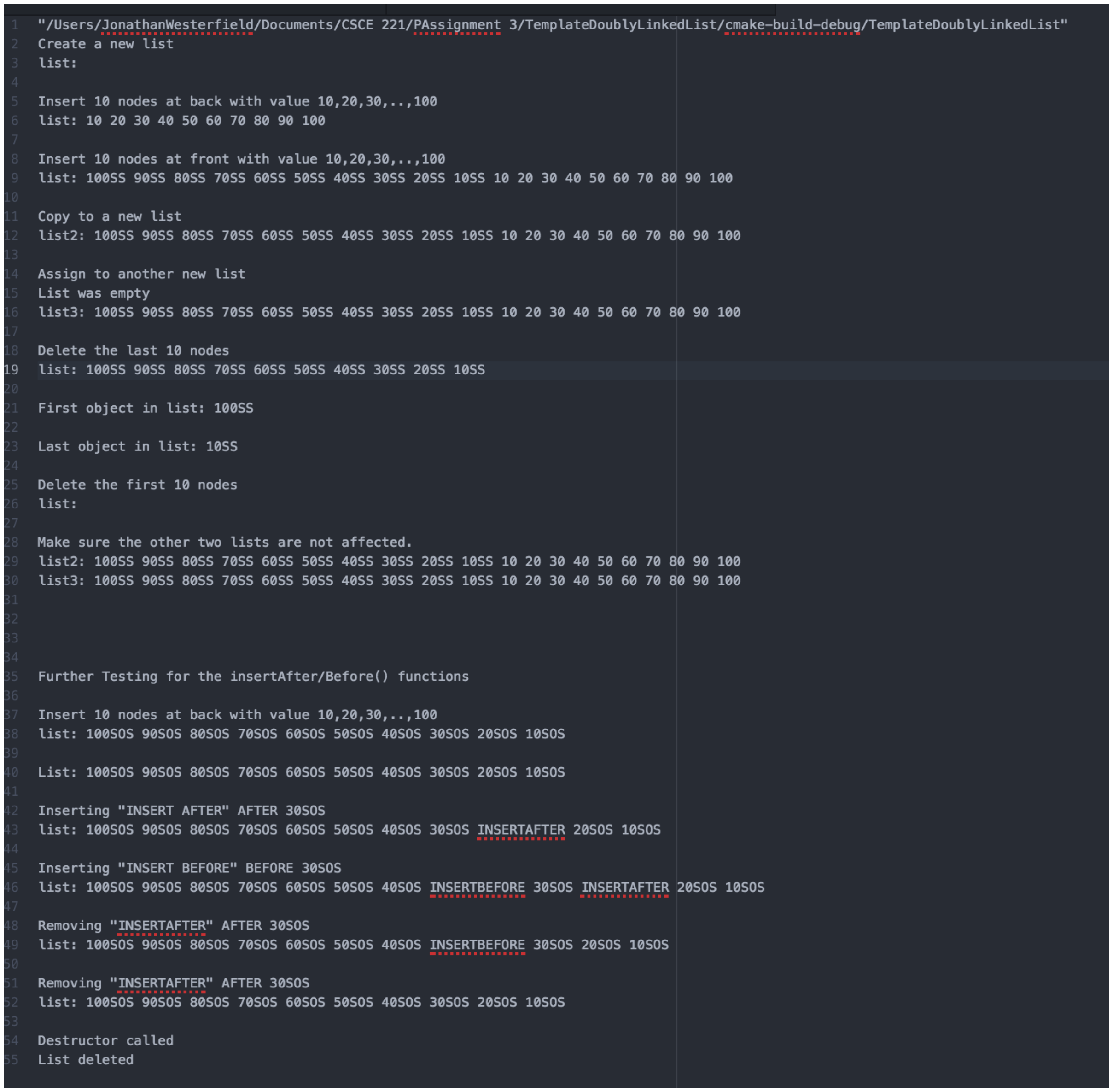
(a) The doubly linked list class is a linear doubly linked list with each node having a previous and next pointer that points to the previous and next node respectively. The TemplateDoublyLinkedList class is implemented exactly the same as the regular DoublyLinkedList class but is implemented using templates. This allows any data type to be stored in the linked list instead of just integer data types. The assignment and ouptut operator are both $O(n)$ because they must parse through the whole list in order to output every element. The copy constructor and the class destructor are also both $O(n)$ because they must parse through the entire list to copy the list's nodes. The insertBefore/After(), removeBefore/After(), insertFirst/Last(), and first/last(), functions are all $O(n)$ since they only have a set number of instructions to follow. The test cases for both the DoublyLinkedList and TemplateDoublyLinkedList class are below and based off of the test main class provided to us.

(b) Doubly Linked List Output

i. 

```
"/Users/JonathanWesterfield/Documents/CSCE 221/PAssignment 3/DoublyLinkedList/cmake-build-debug/DoublyLinkedList"
Create a new list
list:

Insert 10 nodes at back with value 10,20,30,..,100
list: 10 20 30 40 50 60 70 80 90 100

Insert 10 nodes at front with value 10,20,30,..,100
list: 100 90 80 70 60 50 40 30 20 10 10 20 30 40 50 60 70 80 90 100

Copy to a new list
list2: 100 90 80 70 60 50 40 30 20 10 10 20 30 40 50 60 70 80 90 100

Assign to another new list
Destructor called
List deleted
list3: 100 90 80 70 60 50 40 30 20 10 10 20 30 40 50 60 70 80 90 100

Delete the last 10 nodes
list: 100 90 80 70 60 50 40 30 20 10

Delete the first 10 nodes
list:

Make sure the other two lists are not affected.
list2: 100 90 80 70 60 50 40 30 20 10 10 20 30 40 50 60 70 80 90 100
list3: 100 90 80 70 60 50 40 30 20 10 10 20 30 40 50 60 70 80 90 100



Further Testing for the insertAfter/Before() functions

Insert 10 nodes at back with value 10,20,30,..,100
list: 10 20 30 40 50 60 70 80 90 100

Inserting 69 AFTER 30
list: 10 20 30 69 40 50 60 70 80 90 100

Inserting 69 BEFORE 30
list: 10 20 69 30 69 40 50 60 70 80 90 100

Removing 69 AFTER 30
The node containing 69, after 0x7fd699c002c0 has been deleted.
list: 10 20 69 30 40 50 60 70 80 90 100

Removing 69 BEFORE 30
The node containing 69, before0x7fd699c002c0 has been deleted.
list: 10 20 30 40 50 60 70 80 90 100

Destructor called
List deleted
Destructor called
List deleted
Destructor called
List deleted
```

(c) Template Doubly Linked List Output



```
"/Users/JonathanWesterfield/Documents/CSCE 221/PAssignment 3/TemplateDoublyLinkedList/cmake-build-debug/TemplateDoublyLinkedList"
Create a new list
list:

Insert 10 nodes at back with value 10,20,30,..,100
list: 10 20 30 40 50 60 70 80 90 100

Insert 10 nodes at front with value 10,20,30,..,100
list: 100SS 90SS 80SS 70SS 60SS 50SS 40SS 30SS 20SS 10SS 10 20 30 40 50 60 70 80 90 100

Copy to a new list
list2: 100SS 90SS 80SS 70SS 60SS 50SS 40SS 30SS 20SS 10SS 10 20 30 40 50 60 70 80 90 100

Assign to another new list
List was empty
list3: 100SS 90SS 80SS 70SS 60SS 50SS 40SS 30SS 20SS 10SS 10 20 30 40 50 60 70 80 90 100

Delete the last 10 nodes
list: 100SS 90SS 80SS 70SS 60SS 50SS 40SS 30SS 20SS 10SS

First object in list: 100SS

Last object in list: 10SS

Delete the first 10 nodes
list:

Make sure the other two lists are not affected.
list2: 100SS 90SS 80SS 70SS 60SS 50SS 40SS 30SS 20SS 10SS 10 20 30 40 50 60 70 80 90 100
list3: 100SS 90SS 80SS 70SS 60SS 50SS 40SS 30SS 20SS 10SS 10 20 30 40 50 60 70 80 90 100




Further Testing for the insertAfter/Before() functions

Insert 10 nodes at back with value 10,20,30,..,100
list: 100SOS 90SOS 80SOS 70SOS 60SOS 50SOS 40SOS 30SOS 20SOS 10SOS

List: 100SOS 90SOS 80SOS 70SOS 60SOS 50SOS 40SOS 30SOS 20SOS 10SOS

Inserting "INSERT AFTER" AFTER 30SOS
list: 100SOS 90SOS 80SOS 70SOS 60SOS 50SOS 40SOS 30SOS INSERTAFTER 20SOS 10SOS

Inserting "INSERT BEFORE" BEFORE 30SOS
list: 100SOS 90SOS 80SOS 70SOS 60SOS 50SOS 40SOS INSERTBEFORE 30SOS INSERTAFTER 20SOS 10SOS

Removing "INSERTAFTER" AFTER 30SOS
list: 100SOS 90SOS 80SOS 70SOS 60SOS 50SOS 40SOS INSERTBEFORE 30SOS 20SOS 10SOS

Removing "INSERTAFTER" AFTER 30SOS
list: 100SOS 90SOS 80SOS 70SOS 60SOS 50SOS 40SOS 30SOS 20SOS 10SOS

Destructor called
List deleted
```

i.

- Create a directory for the Part 2 that includes: Queue source code, typed report with description of the Queue class implementation, complexity analysis of code expressed in terms of big-O, and the test cases done for correctness.

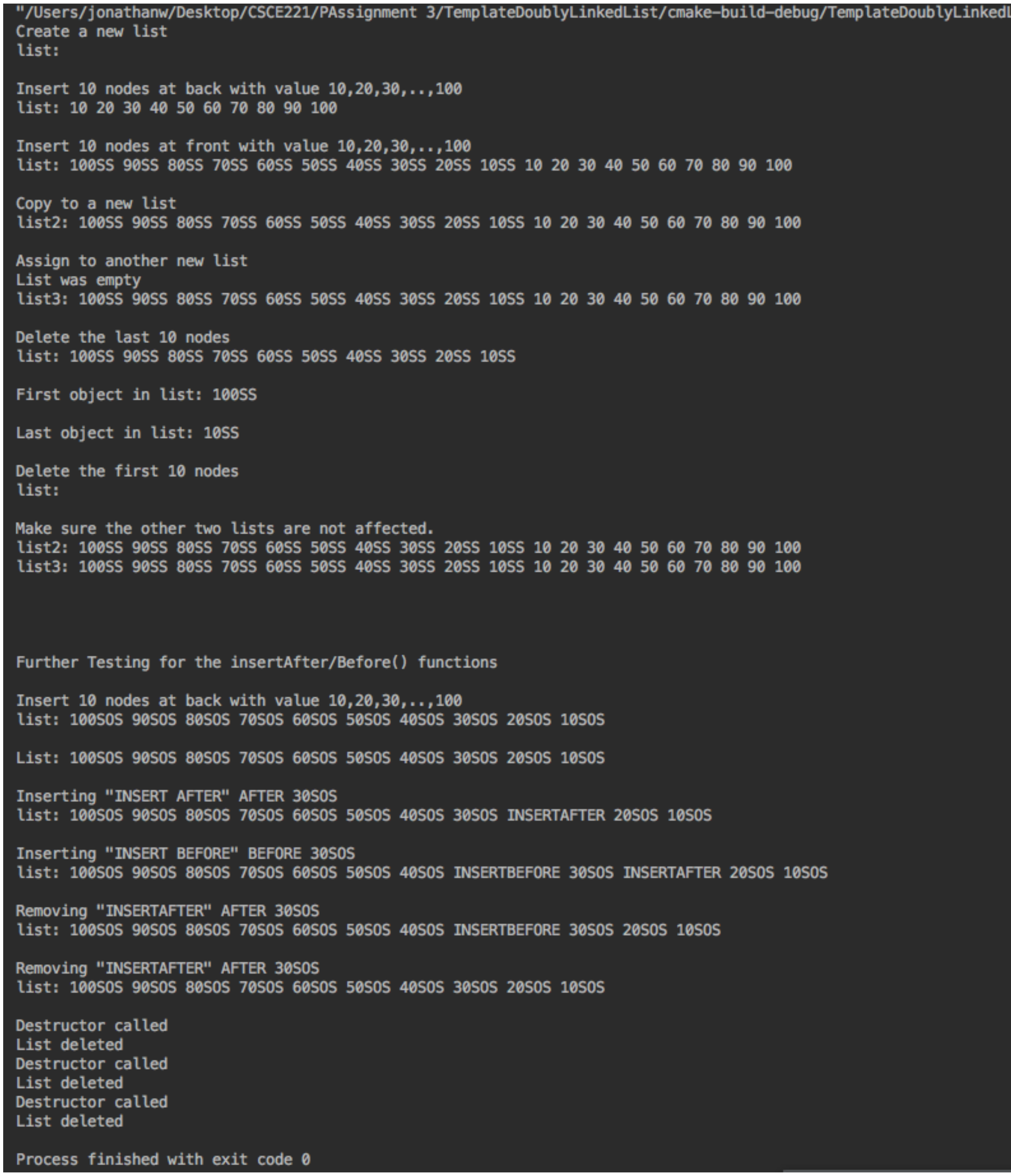
Part II

MinQueue

1. To compile:
   (a) navigate to the folder Submission/Part 2/MinQueue
   (b) Use the command <g++ -std=c++11 *.cpp -o main>
   (c) Then type <./main> into the terminal to run
2. Implementation

(a) The MinQueue class was implemented using the TemplatedDoublyLinkedList class. To support the use the templates in the TemplatedDoublyLinkedList class, the MinQueue class also uses templates to support any data type that gets fed into it. The enqueue(), dequeue(), size(), and isEmpty() functions are all $O(1)$. This is because all of these functions pretty much take 1 to 4 operations to finish the function. The min function is $O(n)$ because it has to parse through all the nodes in the list to compare each and find the min.

(b) MinQueue output

i. 

```
"/Users/jonathanw/Desktop/CSCE221/PAssignment 3/TemplateDoublyLinkedList/cmake-build-debug/TemplateDoublyLinkedL
Create a new list
list:

Insert 10 nodes at back with value 10,20,30,..,100
list: 10 20 30 40 50 60 70 80 90 100

Insert 10 nodes at front with value 10,20,30,..,100
list: 100SS 90SS 80SS 70SS 60SS 50SS 40SS 30SS 20SS 10SS 10 20 30 40 50 60 70 80 90 100

Copy to a new list
list2: 100SS 90SS 80SS 70SS 60SS 50SS 40SS 30SS 20SS 10SS 10 20 30 40 50 60 70 80 90 100

Assign to another new list
List was empty
list3: 100SS 90SS 80SS 70SS 60SS 50SS 40SS 30SS 20SS 10SS 10 20 30 40 50 60 70 80 90 100

Delete the last 10 nodes
list: 100SS 90SS 80SS 70SS 60SS 50SS 40SS 30SS 20SS 10SS

First object in list: 100SS

Last object in list: 10SS

Delete the first 10 nodes
list:

Make sure the other two lists are not affected.
list2: 100SS 90SS 80SS 70SS 60SS 50SS 40SS 30SS 20SS 10SS 10 20 30 40 50 60 70 80 90 100
list3: 100SS 90SS 80SS 70SS 60SS 50SS 40SS 30SS 20SS 10SS 10 20 30 40 50 60 70 80 90 100



Further Testing for the insertAfter/Before() functions

Insert 10 nodes at back with value 10,20,30,..,100
list: 100SOS 90SOS 80SOS 70SOS 60SOS 50SOS 40SOS 30SOS 20SOS 10SOS

List: 100SOS 90SOS 80SOS 70SOS 60SOS 50SOS 40SOS 30SOS 20SOS 10SOS

Inserting "INSERT AFTER" AFTER 30SOS
list: 100SOS 90SOS 80SOS 70SOS 60SOS 50SOS 40SOS 30SOS INSERTAFTER 20SOS 10SOS

Inserting "INSERT BEFORE" BEFORE 30SOS
list: 100SOS 90SOS 80SOS 70SOS 60SOS 50SOS 40SOS INSERTBEFORE 30SOS INSERTAFTER 20SOS 10SOS

Removing "INSERTAFTER" AFTER 30SOS
list: 100SOS 90SOS 80SOS 70SOS 60SOS 50SOS 40SOS INSERTBEFORE 30SOS 20SOS 10SOS

Removing "INSERTAFTER" AFTER 30SOS
list: 100SOS 90SOS 80SOS 70SOS 60SOS 50SOS 40SOS 30SOS 20SOS 10SOS

Destructor called
List deleted
Destructor called
List deleted
Destructor called
List deleted

Process finished with exit code 0
```

- Make a tar file that contains the Part 1 and Part 2 directories and submit it to eCampus for grading.