CSCE 221 Assignment 3 Cover Page

First Name Anders Last Name Wallace UIN 925000221

User Name awallace2 E-mail address awallace2@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: `http://aggiehonor.tamu.edu/`

Type of sources				
People				
Web pages (provide URL)	stackoverflow.com			
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 Anders Wallace Date 10/15/17

CSCE 221 Assignment 3
Fall 2017

Objective

In this assignment we were asked to implement a doubly linked list to handle both integers and generic types, as well as a MinQueue class that used the adapter design pattern in conjunction with the DoublyLinkedList class in order to implement a queue.

Part 1: Implementation of DoublyLinkedList

1. Implementation

   (a) The DoublyLinkedList class was implemented using a doubly linked list design, which is a list of nodes with a header and trailer pointing to the first and last nodes respectively, as well as each node having two pointers, next and prev, which pointed to the nodes in front and behind it. There were several functions which allowed us to insert, delete, and remove various nodes as well as copy constructors, overloaded output and assignment operators, and a destructor. These functions were all implemented for the templated version of the DoublyLinkedList class, which is able to handle generic types.

2. Complexity Analysis

   (a) There were varying runtimes for many of the functions. For functions like the copy constructor, destructor, assignment operator, the functions need to iterate throughout the entire linked list and so their runtimes in terms of Big-O are $O(n)$. However for functions like $insertLast()$, $first()$, and $last()$ their complexities are one operation, and so their runtime in terms of Big-O are $O(1)$. Below is a list of the various runtimes of each function:

   $O(n)$: Copy Constructor, Assignment Operator, insertAfter, insertBefore, removeAfter, removeBefore, DoublyLinkedListLength, removeAll, Overloaded Output Operator

   $O(1)$: getFirst, getAfterLast, isEmpty, insertFirst, insertLast, removeFirst, removeLast, last

3. Test Cases

   (a) Below are screenshots of the test cases for DoublyLinkedList and TemplateDoublyLinkedList respectively

```
[awallace2]@build ~/CSCE_221/PA3> (22:02:56 10/16/17)
:: ./a.out
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
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
Insert after test: 100 15 90 80 70 60 50 40 30 20 10 10 20 30 40 50 60 70 80 90
100
Insert before test: 16 100 15 90 80 70 60 50 40 30 20 10 10 20 30 40 50 60 70 80
 90 100
Remove after test: 16 15 90 80 70 60 50 40 30 20 10 10 20 30 40 50 60 70 80 90 1
00
List length test: 21

[awallace2]@build ~/CSCE_221/PA3> (22:03:03 10/16/17)
::
```

```
[awallace2]@build ~/CSCE_221/PA3/Template_Stuff> (22:05:33 10/16/17)
:: ./a.out
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
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
Insert after test: 100 hello 90 80 70 60 50 40 30 20 10 10 20 30 40 50 60 70 80 90
Insert before test: goodbye 100 hello 90 80 70 60 50 40 30 20 10 10 20 30 40 50 60
Remove after test: goodbye hello 90 80 70 60 50 40 30 20 10 10 20 30 40 50 60 70 80
List length test: 21

[awallace2]@build ~/CSCE_221/PA3/Template_Stuff> (22:05:37 10/16/17)
::
```

Part 2: Implementation of MinQueue data structure based on DoublyLinkedList

1. Implementation

   (a) The MinQueue class was implemented using an adapter design pattern with the previously written DoublyLinkedList class in order to create a properly working queue ADT. The goal was to limit the DoublyLinkedList class to FIFO (first in first out) in order to match the scope of a queue. We then created functions like enqueue, dequeue (to insert and remove elements respectively), and others like isEmpty, size, and min in order to check if there are elements and how many elements/what the smallest element is. Under the hood the MinQueue class was a doubly linked list, but we limited the functions that could be done to it to queue operations in order to get the first in first out aspect of a data structure like a queue.

2. Complexity Analysis

   (a) The functions of the MinQueue class both fell into runtime worst cases of $O(1)$ and $O(n)$. However, because many functions were able to use the methods of the DoublyLinkedList class that were constant $O(1)$, functions like enque and deque ran in constant time. Meanwhile, the methods size and min had to run in linear time, because they required to iterate throughout the entire queue in order to obtain the size and smallest element. Below is a compilation of the worst case runtime of all the methods:

$O(1)$: isEmpty, enqueue, dequeue

$O(n)$: Constructor, Destructor, size, min

3. Test Cases

   (a) Below is a screenshot of a main function testing the various features of the MinQueue class