CSCE 221 Assignment 3 Cover Page

First Name: Adrian Last Name: Gamez-Rodriguez UIN:126009409

User Name: adriangamez E-mail address: adriangamez@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/

Types of Sources				
People	TA in Lab			
Web Pages	Piazza	Stackoverflow		
Printed Material	Lecture Slides	Textbook		
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

Adrian Gamez-Rodriguez Date: 3/7/2019

Part 1 (Doubly Linked List & Template Doubly Linked List)

The purpose of Part 1of this assignment is to create a doubly linked list data structure that can hold integers and then template it to so it can hold any data type. Implementing this data structure allows us to utilize and recall many valuable concepts in C++, such as pointers, classes, structs, and OOP characteristics.

As it might be obvious the data structure, we are making is a double linked list data that is made of nodes and pointers. These simple concepts can create a data structure that can hold information and can hold a pointer to the previous or next node and essentially create a list. It's important to note that the struct Node can be considered a data structure in it of itself because they are the building blocks of the doubly linked list.

Implementing these data structures are in C++ involved using pointers and classes to define what a DListNode is and what the member variables are. In this case the DListNode member variables are the pointers to the next and previous nodes and the data that it holds. Using the DListNode, we create another class called DoublyLinkedList that has member variables that are objects of the DListNode struct and we call them Header and Trailer. These objects tell us where the list starts and ends. Along with the member variables there are member functions that allow us to manipulate the data structure and the data that is holds. They also define operations allowed on the data structure.

Some of the algorithms used to define the member functions of the DoublyLinkedList class have a time complexity of O(n) while others have O(1). I'll cover that once I go over every function and what they do.

The class DoublyLinkedList has quite a few member functions. Here is what they are; constructor, copy constructor, move constructor, destructor, assignment operator, move

assignment, getFirst(), getAfterLast(), isEmpty(), first(), last(), insertFirst(), removeFirst(), insertLast(), removeLast(), insertAfter(), insertBefore(), removeAfter(), removebefore(), DoublyLinkedListLength, and operator <<.

What the constructor does is construct the header and trailer nodes and points them to each other because the list is empty when creating an instance of the class. The move constructor, “steals” data representation of the old object and assigns it to the new object. The destructor destroys an object and makes sure that no memory leaks occur. Assignment operator assigns the new obj with a deep copy of the old object. The move assignment does a similar thing to the move constructor as it just steals the old objects representation and makes sure to deallocate the old object. GetFirst(), function just returns a pointer to the first node of the list. GetAfterLast() function, returns a pointer to the trailer node. isEmpty(), function checks if the list is empty by checking if the header and trailer nodes are pointing to each other. First(), returns the first node’s data. Last(), returns the last node’s data. InsertFirst(), inserts new data at the beginning of the list. InsertLast(), inserts new data at the end of the list. RemoveFirst(), removes the first node of the list. RemoveLast(), removes the last node of the list. InsertAfter, inserts new node after the specified node. InsertBefore(), inserts new node before the specified node. RemoveAfter(), removes the node after the specified node. Finally, removeBefore() ,removes the node before the specified node. As you can see these functions use the same basic algorithm of changing pointers to adjust the DoublyLinkedList for each of the functions purpose. The other remaining functions, DoublyLinkedListLength return the size of the list, and the overloaded operator <<, outputs the list to see. These functions maintain the DoublyLinkedList objects and decide what operations on the object is possible. Also note that one exception is used throughout the program which is EmptyDLinkedListException, which helps prevent runtime errors. The

implementation of the DoublyLinkedList required OOP concepts like encapsulation of the data as nodes and then encapsulating those nodes to create the Doubly Linked List data structure.

Time Complexity of Functions used:

Function	Time Complexity
Copy constructor(), Move constructor(), Destructor(), Assignment operator(), Move Assignment(), << operator(), DoublyLinkedListLength(),	**O(n)**
getFirst(), getAfterLast(), isEmpty(), first(), last(), insertFirst(), removeFirst(), insertLast(), removeLast(), insertAfter(), insertBefore(), removeAfter(), removeBefore(),	**O(1)**

Note that these time complexities also apply to the templated version of the DoublyLinkedList class, because it's essentially the same algorithms just different data types that the class can hold.

To Run the DoublyLinkedList program simply go to the folder labeled DoublyLinkedList (should be in **Part1** folder) and type the command, ***make all*** then to run type the command, ***./run-dll***. No input files or input from the user are required. The output should appear on screen.

To Run the TemplateDoublyLinkedList program go to the folder labeled **Part1**, look for the folder, **TemplateDoublyLinkedList** go into that folder. Then type the command ***make all***, next to run it type the command ***./run-tdll***. No input files or input from the user are required and the output should appear on screen.

Some Test Cases (int DoublyLinkedList):

Valid tests:

```
//tests for insertAfter, insertBefore
cout<<"Inserting After Tests\n";
dll2.insertAfter(*(dll2.getFirst()->next->next), 36);
cout<<"list2: after inserting 36 after the third node, "<<dll2<<endl;
dll3.insertAfter(*(dll3.getFirst()->next->next->next->next), 65);
cout<<"list3: after inserting 65 after the fifth node, "<<dll3<<endl;
cout<<endl;

//Test for  insertBefore
cout<<"Inserting Before Tests\n";
dll2.insertBefore(*(dll2.getFirst()->next->next->next->next->next), 25);
cout<<"list2: after inserting 25 before the sixth node, "<<dll2<<endl;
dll3.insertBefore(*(dll3.getFirst()->next->next->next->next->next->next), 23);
cout<<"list3: after inserting 23 before the seventh node "<<dll3<<endl;


// Tests for removeAfter
cout<<"Remove After Tests\n";
dll2.removeAfter(*(dll2.getFirst()->next->next));
cout<<"list2: after removing the node after the third node, "<<dll2<<endl;
dll3.removeAfter(*(dll3.getFirst()->next->next->next->next));
cout<<"list3: after removing the node after the fifth node, "<<dll3<<endl;

//Tests for removeBefore
cout<<"Remove Before Tests \n";
dll2.removeBefore(*(dll2.getFirst()->next->next));
cout<<"list2: after removing the node before the third node, "<<dll2<<endl;
dll3.removeBefore(*(dll3.getFirst()->next->next->next->next));
cout<<"list3: after removing the node before the fifth node, "<<dll3<<endl;


// Tests for DoublyLinkedListLength
cout<<endl;
cout<<"List length of list1: "<<DoublyLinkedListLength(dll);
cout<<endl;
cout<<"List length of list2: "<<DoublyLinkedListLength(dll2);
cout<<endl;
cout<<"List length of list3: "<<DoublyLinkedListLength(dll3);
```

```
[adriangamez]@linux2 ~/CSCE221/Assignment3/Part1/DoublyLinkedList> (21:15:51 03/07/19)
:: make all
g++ -std=c++11 -c DoublyLinkedList.cpp
g++ -std=c++11 -c Main.cpp
g++ -std=c++11 DoublyLinkedList.o Main.o -o run-dll

[adriangamez]@linux2 ~/CSCE221/Assignment3/Part1/DoublyLinkedList> (21:15:53 03/07/19)
:: ./run-dll
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

Inserting After
list2: after inserting 36 after the third node, 100 90 80 36 70 60 50 40 30 20 10 10 20 30 40 50 60 70 80 90 100
list3: after inserting 65 after the fifth node, 100 90 80 70 60 65 50 40 30 20 10 10 20 30 40 50 60 70 80 90 100

Inserting Before
list2: after inserting 25 before the sixth node, 100 90 80 36 70 25 60 50 40 30 20 10 10 20 30 40 50 60 70 80 90 100
list3: after inserting 23 before the seventh node 100 90 80 70 60 65 23 50 40 30 20 10 10 20 30 40 50 60 70 80 90 100

Remove After
list2: after removing the node after the third node, 100 90 80 70 25 60 50 40 30 20 10 10 20 30 40 50 60 70 80 90 100

list3: after removing the node after the fifth node, 100 90 80 70 60 23 50 40 30 20 10 10 20 30 40 50 60 70 80 90 100

Remove Before
list2: after removing the node before the third node, 100 80 70 25 60 50 40 30 20 10 10 20 30 40 50 60 70 80 90 100
list3: after removing the node before the fifth node, 100 90 80 60 23 50 40 30 20 10 10 20 30 40 50 60 70 80 90 100

List length of list1: 0
List length of list2: 20
List length of list3: 20
```

Testing removeFirst(), removeLast(), removeAfter and removeBefore, when the list is empty: (should throw error)

```
try
{
    DoublyLinkedList dll4;
  dll4.removeFirst();
}
catch(exception &error){cerr<<"Error: "<<error.what()<<endl;}
try
{
    DoublyLinkedList dll4;
  dll4.removeLast();
}
catch(exception &error){cerr<<"Error: "<<error.what()<<endl;}
try
{
    DoublyLinkedList dll4;
  dll4.removeAfter(*(dll4.getFirst()));
}
catch(exception &error){cerr<<"Error: "<<error.what()<<endl;}
try
{
    DoublyLinkedList dll4;
  dll4.removeBefore(*(dll4.getFirst()));
}
catch(exception &error){cerr<<"Error: "<<error.what()<<endl;}
return 0;
```

```
[adriangamez]@linux2 ~/CSCE221/Assignment3/Part1/DoublyLinkedList> (22:10:04 03/07/19)
:: ./run-dll
Error: DoublyLinkedList is Empty
Error: DoublyLinkedList is Empty
Error: DoublyLinkedList is Empty
Error: DoublyLinkedList is Empty
```

Test Cases for TemplateDoublyLinkedList:

Valid Test Cases:

```
// add tests for insertAfter,
cout<<"Inserting After\n";
dll2.insertAfter(*(dll2.getFirst()->next->next), "hel
cout<<"list2: after inserting 'hello' after the third
dll3.insertAfter(*(dll3.getFirst()->next->next->next-
cout<<"list3: after inserting 'hello again' after the
cout<<endl;

//Test for  insertBefore
cout<<"Inserting Before\n";
dll2.insertBefore(*(dll2.getFirst()->next->next->next
cout<<"list2: after inserting 'bye' before the sixth
dll3.insertBefore(*(dll3.getFirst()->next->next->next-
cout<<"list3: after inserting 'ohhh' before the seven

// Tests for removeAfter
cout<<"Remove After\n";
dll2.removeAfter(*(dll2.getFirst()->next->next));
cout<<"list2: after removing the node after the third
dll3.removeAfter(*(dll3.getFirst()->next->next->next-
cout<<"list3: after removing the node after the fifth

//Tests for removeBefore
cout<<"Remove Before \n";
dll2.removeBefore(*(dll2.getFirst()->next->next));
cout<<"list2: after removing the node before the thir
dll3.removeBefore(*(dll3.getFirst()->next->next->next
cout<<"list3: after removing the node before the fift
cout<<endl;

 // Tests for DoublyLinkedListLength
cout<<"List length of list1: "<<DoublyLinkedListLength
cout<<endl;
cout<<"List length of list2: "<<DoublyLinkedListLength
cout<<endl;
cout<<"List length of list3: "<<DoublyLinkedListLength
```

```
[adriangamez]@linux2 ~/CSCE221/Assignment3/Part1/TemplateDoublyLinkedList> (22:20:24 03/07/19)
:: ./run-tdll
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
Inserting After
list2: after inserting 'hello' after the third node, 100 90 80 hello 70 60 50 40 30 20 10 10 20 30 40 50 60 70 80 90
100
list3: after inserting 'hello again' after the fifth node, 100 90 80 70 60 hello again 50 40 30 20 10 10 20 30 40 50
60 70 80 90 100

Inserting Before
list2: after inserting 'bye' before the sixth node, 100 90 80 hello 70 bye 60 50 40 30 20 10 10 20 30 40 50 60 70 80
90 100
list3: after inserting 'ohhh' before the seventh node 100 90 80 70 60 hello again ohhh 50 40 30 20 10 10 20 30 40 50
60 70 80 90 100
Remove After
list2: after removing the node after the third node, 100 90 80 70 bye 60 50 40 30 20 10 10 20 30 40 50 60 70 80 90 10
0
list3: after removing the node after the fifth node, 100 90 80 70 60 ohhh 50 40 30 20 10 10 20 30 40 50 60 70 80 90 1
00
Remove Before
list2: after removing the node before the third node, 100 80 70 bye 60 50 40 30 20 10 10 20 30 40 50 60 70 80 90 100
list3: after removing the node before the fifth node, 100 90 80 60 ohhh 50 40 30 20 10 10 20 30 40 50 60 70 80 90 100


List length of list1: 0
List length of list2: 20
List length of list3: 20
```

Some Invalid cases:

```
cout<<"Some invalid cases that call removeFirst(), removeLast(), removeAfter(), and removeBefore():"<<endl;
try
{
    DoublyLinkedList<string> dll4;
    dll4.removeFirst();
}
catch(exception &error){cerr<<"Error: "<<error.what()<<endl;}
try
{
    DoublyLinkedList<string> dll4;
    dll4.removeLast();
}
catch(exception &error){cerr<<"Error: "<<error.what()<<endl;}
try
{
    DoublyLinkedList<string> dll4;
    dll4.removeAfter(*(dll4.getFirst()));
}
catch(exception &error){cerr<<"Error: "<<error.what()<<endl;}
try
{
    DoublyLinkedList<string> dll4;
    dll4.removeBefore(*(dll4.getFirst()));
}
catch(exception &error){cerr<<"Error: "<<error.what()<<endl;}
```

```
[adriangamez]@linux2 ~/CSCE221/Assignment3/Part1/TemplateDoublyLinkedList> (22:31:13 03/07/19)
:: ./run-tdll
Some invalid cases that call removeFirst(), removeLast(), removeAfter(), and removeBefore():
Error: DoublyLinkedList is Empty
Error: DoublyLinkedList is Empty
Error: DoublyLinkedList is Empty
Error: DoublyLinkedList is Empty
```

Part 2 (TemplateMinQueue)

Part 2 of the assignment involved creating another data structure that is Minimum Priority Queue by using an adaptive design that utilizes the already constructed class DoublyLinkedList. More specifically it uses the templated version of the class. The data structure MinQueue also has its own member variables and member functions. The purpose of this MinQueue is to store comparable elements.

To implement such a data structure, I encapsulated the class of the templated doubly linked list and used that as a member variable in which I store all data. One constraint to this data structure is that it can only store comparable elements in its queue. As per the assignment requirements we only had to implement five member functions which are, enqueue(x), dequeue(), size(), isEmpty(), and min(). The functions are self-explanatory, enqueue(x) queues

an object into the queue, dequeue() removes the minimum obj in the queue, size() returns the size of the queue, and min() returns the minimum obj but does not remove it.

A brief C++ implementation of the data structure is templating the MinQueue class, alongside with declaring a private member of the type DoublyLinkedList<T> and calling it minQ, defining a constructor, and declaring the five functions required and also an overloaded operator that outputs the queue, a helper function that returns the DoublyLinkedList<T> for the << operator to print out. Note that one exception is used, it is the same exception that was utilized in part 1. As I mentioned before this assignment involves OOP concepts such as encapsulation.

How the min() function is implemented results in a time complexity of O(n), because it searches for the minimum object while traversing the entire queue, thus depending on the size of the queue n, however the function does not delete the node. The isEmpty() function is O(1), because it calls the function isEmpty() from the DoublyLinkedList<T> class, which just checks whether the header and trailer point to each other. The function dequeue(), is O(n), because it must search for the node that contains the minimum and then removes that node. The function enqueue(x), is O(1), because it just inserts the new node at the end of the queue. Note that by implementing it this way the queue will not be sorted.

Time Complexity of functions:

Function	**Time Complexity**
Min() **Dequeue()**	**O(n)**
isEmpty() **enqueue(x)**	**O(1)**

size()	

To run the MinQueue program go to folder named **Part2**, then go to the folder named, **MinQueue**. Once there type the command ***make all,*** then type the command ***./run-minQueue.*** Again, no input files or input from the user is required, and the output should display on the screen.

Some Test cases (MinQueue):

Valid cases with type int:

```
//Creating MinQueue object
MinQueue<int> minQ1;
cout<<"After creating a MinQueue object with type int\nminQ1: ";
cout<<minQ1;
cout<<endl;
cout<<"Is minQ1 empty? :";
cout<<minQ1.isEmpty();
cout<<endl;
cout<<endl;

//Testing Enqueue with random numbers to enqueue.
cout<<"Inputing 10 random intergers of range 0-100 into the queue:\n";
for(int i= 0; i<10;i++)
{
    int random = rand()%100;
    minQ1.enqueue(random);
}
cout<<"minQ1: ";
cout<<minQ1;
cout<<endl;
cout<<endl;

//Testing size function
cout<<"The size of the min priority queue is:
cout<<minQ1.size();
cout<<endl;
cout<<endl;


//Testing the min() function
cout<<"The minimum of minQ1 is: ";
cout<<minQ1.min();
cout<<endl;
cout<<endl;


//Testing the dequeue function
cout<<"The minQ1 after removing the minimum n
minQ1.dequeue();
cout<<minQ1;
cout<<endl;
cout<<"Size of queue: ";
cout<<minQ1.size();
cout<<endl;
cout<<endl;

//Testing the isEmpty() function
cout<<"Is the minQ1 empty?: ";
cout<<minQ1.isEmpty();
cout<<endl;
```

```
[adriangamez]@linux2 ~/CSCE221/Assignment3/Part2/MinQueue> (22:41:37 03/07/19)
:: make all
c++ -std=c++11 MinQueueMain.o -o minQueue

[adriangamez]@linux2 ~/CSCE221/Assignment3/Part2/MinQueue> (22:42:31 03/07/19)
:: ./minQueue
After creating a MinQueue object with type int
minQ1:
Is minQ1 empty? :True

Inputing 10 random intergers of range 0-100 into the queue:
minQ1: 32 84 72 67 91 72 36 13 0 44

The size of the min priority queue is: 10

The minimum of minQ1 is: 0

The minQ1 after removing the minimum node :32 84 72 67 91 72 36 13 44
Size of queue: 9

Is the minQ1 empty?: False
```

Some invalid test cases when calling dequeue() and min() when MinQueue is empty:

```
//Some invalid test cases

cout<<"Create new object called minQ2\n";
MinQueue<int> minQ2;
cout<<"minQ2: ";
cout<<minQ2;
cout<<endl;
cout<<endl;
cout<<"Testing invalid cases, calling dequeue and min when list is empty:\n";
try
{
    minQ2.min();
}catch(exception &error){cerr<<"Error: "<<error.what();}
cout<<endl;
try
{
    minQ2.dequeue();
}catch(exception &error){cerr<<"Error: "<<error.what();}

return 0;
```

```
[adriangamez]@linux2 ~/CSCE221/Assignment3/Part2/MinQueue> (23:00:11 03/07/19)
:: ./minQueue
Create new object called minQ2
minQ2:

Testing invalid cases, calling dequeue and min when list is empty:
Error: DoublyLinkedList is Empty
Error: DoublyLinkedList is Empty
[adriangamez]@linux2 ~/CSCE221/Assignment3/Part2/MinQueue> (23:00:17 03/07/19)
```