CSCE 221 Cover Page
Programming Assignment #3

First Name: Devin Last Name: Tuchsen UIN: 121000270

User Name: devint1 E-mail address: devint1@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 in the Aggie Honor System Office http://aggiehonor.tamu.edu/

Type of sources		
People	Jonathan Willing	Chris Lenart
Web pages (provide URL)	https://piazza.com/class#spring2013/csce221/ http://courses.cs.tamu.edu/teresa/csce221/csce221-index.html	cplusplus.com stackoverflow.com
Printed material		
Other Sources		

I certify that I have listed all the sources that I used to develop the solutions/code 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 (signature) Date Assignment

3 Report

Program Description This project contains a number of program used to test a number of different data structures: linked list, linked stack, and linked queue. Each of these data structures have their own program for testing, but they are all implemented in the "reverse" program. This program uses a linked queue and a linked stack to reverse the lines in a given "input.txt" file.

Purpose of the Assignment The purpose of this assignment was to explore a number of data structures discussed in class, namely: linked list, linked stack, and linked queue. Implementing the various data structures served as a reinforcement of their functionality, and the testing involved in the various programs served as debugging and testing practice.

Data Structures Description A number of data structures were implemented in this assignment:

LinkedList This data structure consists of two pointers to `ListNode`: `head` and `tail`. Each `ListNode` contains two other members: `obj`, which is a string object used to store data; and `next`, a pointer to another `ListNode`. `head` serves as the first element in the linked list, while `tail` serves as the last element in the linked list. This forms a chain of elements, each node having a link to the next node.

LinkedStack This data structure uses a linked list and forces LIFO behavior by limiting operations to pop, which removes the last element and returns it; top, which returns the first element; and push, which places an element onto the stack.

LinkedQueue This data structure uses a linked list and forces FIFO behavior by limiting operations to dequeue, which removes the first element and returns it; enqueue, which places an element at the end of the list; and first, which returns the first element.

Algorithm Description The main algorithms in this assignment were in the linked list implementation. In the copy constructor, new memory is recursively allocated for each node while assigning it a value in a given linked list node, thus creating a copy of the given linked list. `size` iterates over each node in the list and returns a count of all nodes, and is $O(n)$. `insertFirst` and `insertLast` add elements to the list, both of which are $O(1)$. `removeFirst` removes elements from the beginning of the list, and is also $O(1)$. `removeAll` and the overloaded `<<` operator are both $O(n)$.

Program Organization and Description of Classes There are three main classes in this assignment: `linkedList`, `linkedStack`, and `linkedQueue`. Each of these represent a data structure discussed in class. There are also files with these same names ending in "main" that perform the testing of the program. The class `listNode` defined in LinkedList.h represents a single node in a linked list. There are also some other exception classes defined in RuntimeException.h and overridden in the different data structure classes.

How to Compile and Run

1. In a Unix terminal, change the current directory by typing `cd SOURCE_DIR`, where `SOURCE_DIR` is the path to the source directory.
2. Type `make COMMAND`, where `COMMAND` can be any of the following:

   `test_all`: Compiles all sources.
   `linkedlist`: Compiles the linked list test program.
   `linkedstack`: Compiles the linked stack test program.
   `linkedqueue`: Compiles the linked queue test program.
   `reverse`: Compiles the reverse program.
   `clean`: Remove all binaries and debugging symbols.
   NOTE: Omitting `COMMAND` will default to `reverse`.

3. Run the executable by typing `./EXECUTABLE`, where `EXECUTABLE` can be `linkedlist`, `linkedstack`, `linkedqueue`, or `reverse`.

4. For the reverse program, the file "input.txt" may be changed.

Input/Output Specifications For all programs but reverse, the output is pre-specified and there is no user input. For the file "input.txt," the string elements are expected to be separated by a newline.

Logical Exceptions In the event that an element is attempted to be removed from an empty list, an exception will be called. The same will happen if `pop`, `top`, `first`, or `dequeue` are called on an empty list. There are no other known scenarios that will cause exceptions or crashes.

C++ Object Oriented Features There are separate classes for each of the three data structures, as well as linked list nodes and exceptions. Inheritance occurs in the data structure classes when `RuntimeException` is overridden.

Tests The following are output from these programs:

linkedlist

```
ll = {}
length of the list = 0
After inserting 5 elements to ll, ll = {OOO RRR AAA GGG EEE}
length of the list = 5
first element = OOO
Assigning ll to ll_copy, ll_copy = {EEE}
length of the list = 1
first element = EEE
Copying ll to ll_copy2, ll_copy2 = {OOO RRR AAA GGG EEE}
length of the list = 5
first element = OOO
After removing 4 elements from ll, ll = {EEE}
length of the list = 1
After removing all elements, ll_copy = {}
length of the list = 0
list is empty.
The copy of ll, ll_copy2 = {OOO RRR AAA GGG EEE}
length of the list = 5
first element = OOO
```

linkedqueue

```
queue = {1 22 333 4444 55555 123456}
size of queue = 6
first of queue = 1
assigning queue to queue_copy, queue_copy = {1 22 333 4444 55555 123456}
size of queue_copy = 6
first of queue_copy = 1
copy constructor queue = {1 22 333 4444 55555 123456}
size of queue_copy2 = 6
first of queue_copy2 = 1
1
22
333
```

```
4444
55555
123456
dequeueing all elements from queue_copy2, queue_copy2 = {}
original queue:
{1 22 333 4444 55555 123456}
```

linkedstack

```
stack = {ABCDE GGG NNNN AAA RR O}
size of stack = 6
top of stack = ABCDE
assigning stack to stack_copy, stack_copy = {ABCDE GGG NNNN AAA RR O}
size of stack_copy = 6
top of stack_copy = ABCDE
copy constructor stack = {ABCDE GGG NNNN AAA RR O}
size of stack_copy2 = 6
top of stack_copy2 = ABCDE
ABCDE
GGG
NNNN
AAA
RR
O
popping all elements from stack_copy2, stack_copy2 = {}
original stack:
{ABCDE GGG NNNN AAA RR O}
```

reverse

```
Original list:
WHAT
LOLOLOL
HAHAHAHAH
THIS IS A STRING
THIS IS TOO!!!
HoW ExCiTiNg!!
Reversed list:
HoW ExCiTiNg!!
THIS IS TOO!!!
THIS IS A STRING
HAHAHAHAH
LOLOLOL
WHAT
```

input.txt

```
WHAT
LOLOLOL
HAHAHAHAH
THIS IS A STRING
THIS IS TOO!!!
HoW ExCiTiNg!!
```

NOTE: Screenshots are in a separate folder.