CSCE 221 Cover Page
Homework Assignment

First Name: Victoria Last Name: Rivera Casanova UIN: 927007294

User Name: vrc E-mail address: vrc@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	n/a				
Web pages (provide URL)	stack overflow				
Printed material					
Other Sources	Slides/Notes from class				

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 6/8/2020

The Programming Assignment Report Instructions CSCE 221

1. The description of an assignment problem.

   In this assignment, we were tasked with creating programs that utilized Linked Lists and Doubly Linked Lists to display, store and organize data. The assignment builds on each of its parts and at the end we put it all together to create a library database, where the user can search for a book and enter a new book into the database.

   I have included the runtime analysis of each function here on the report if you do not find it in the code comments.

2. The description of data structures and algorithms used to solve the problem.

   (a) Provide definitions of data structures by using Abstract Data Types (ADTs)

   We used several versions of data structures to organize the input data of this assignment.

   Doubly Linked List: For Part 1, we created a simple DLLinked list and used it to store the input data from the given "main.cpp" file. The doubly linked list is a linear in nature, however, you can access the next or previous node of the current node you are at. The links or nodes are allocated on the heap and its declaration on the stack.

   Vector: Vectors we used in Part 0 and Part 2 to store input data from the user. They work like arrays, but they can have their size altered using a built-in function (unlike arrays where you have to call a resize function).

(b) Write about the ADTs implementation in C++.

The **doubly-linked** list implementation had 4 functions to begin with associated to its class (insert_before(), insert_after()… etc) that were used to insert new nodes and remove nodes from the overall list. In Part 1, we made the DLList class more usable and reliable by adding out own constructors, destructors and functions. We added functions to remove the last, first or any node in the middle. The same thing was done for insert functions. They also contained header and trailer nodes that marked the start and end of the list. This implementation is different that other DLList implementation where the first node's previous and the last node's next points to nullptr (another variation I had seen before).

In Part 1, we made the DLList class a template class and that means it can take in any variable T (string, int, user defined objects, etc). This would go on to help us in Part 2, where we made the DLList of type Record (from Part 0).

The **vector** ADT was utilized in Part 2 to create our "library". Each position of the vector corresponded to a letter on the alphabet that contained a DLList with only "Records" of names that began with that particular letter. This worked to keep the database organized and efficient.

(c) Describe algorithms used to solve the problem.

In Part 2 we used sorting and searching algorithms. We used sorting to sort the given input into its correct position in the vector and, also, to sort the Record into the correct position within the DLList.

Searching was utilized when the user entered in a book title. We had to create an algorithm that would look for the book inside its correct vector spot and return the found book, a list of books with the same name, or that it did not find the book. These loops had to go into the DLList a few times to get the correct results. We had to iterate through both the vector and the DLList to be at the correct locations in each ADT for our program to work.

(d) Analyze the algorithms according to assignment requirements. (page 20 w/ q's from instructions)

I wrote comments beside each function about the Big-Oh runtime and what was controlling the iterations. But I will also write down a few here:

Part 0: The main algorithm here was the overloaded input >> operator function. I had the while loop (!isf.eof()) in the main and each time that function was called was based on the number of Record inputs inside the text file. Inside the function, it checks for the blank line and takes in the 5 lines of input, creates a new Record and outputs that into the awaiting declared Record variable back in the main.

Inside the main, the while loop, while loop (!isf.eof()), runs O(n) times where n is the number of Records inside the text file. This is because the ">>" operator reads in 5 lines + the blank line in one single function call.

SimpleDLList: This is the beginning of Part 1. For this directory we only had to implement 4 functions and they each have O(1) runtime because it is either removing or inserting a given node that is passed into it as a parameter. What takes up more runtime is the for loops inside the main function that are calling our implemented functions. Those run at O(n) time though.

I also want to note that the implementation of the DLList here is different than in the further sections. This implementation does not have a define header or trailer.

DLList and Templated DLList: The implementation for these functions that we implemented are about the same as the SimpleDLList because they run at O(1) time. The only exception being the constructors and the destructor. The for loops in the main DLList-main.cpp and TDLList-main.cpp have a O(n) as we have seen consistently in this assignment.

The constructor and destructor have a O(n), where n is the number of nodes inside the DLList. This is because those nodes have to each be deep copied into the given variable and in the case of the destructor those nodes each have to be destroyed.

The move constructor have a O(1) because they only do a shallow copy; only moving the header and the trailer of the old list. And the move assignment runs at O(n), where n is the number of nodes in the parameter's list.

Part 2: This section gets more complex because it has sorting and searching algorithms to take into account. All the above regarding the Templated DLList functions apply to Part 2 because we use those functions to implement Part 2 instructions.

I implemented mine where some of my DLLists were already filled. I used the same code from Part 0 to read in the file. This was also useful for testing. Below I am going to list my main functions and take about their BigO and the conditions needed for those functions to run:

Note: Besides the books in the text file, I designed my program where a book with the same title cannot be repeated. Even though as you will see on the test there are 2 versions of Harry Potter and the Chamber of Secrets. This was allowed to enter because the ISBN number were different. If they were the same, only one record would have been sorted.

bool sortRecord(int index, Record& book): This, I believe, is the function that is called most often throughout my program because all the scenarios end up with sortRecord() being called. It's worst case runtime is O(n), where n is the number of nodes inside the designated doubly linked this. This occurs when the given Record, "book", is alphabetically last in the entire list. In order for it to be placed in the correct spot, it has to traverse the entire length of the DLList. It's best case is O(1) where it is alphabetically before the first node or the list is empty and it is inserted_first().

If the book entered by the user isn't in the DLList, then this function will be called to sort the new entry. This function is called in the beginning of the program to sort the inputted Records from the text file (this was an optional thing, but it was useful and I left it in).

sortRecord() has multiple if else comparisons to get the correct position to input the new Record. These run O(1) for iteration of the main while loop of the function.

void enterNewBook(string name, DLList<Record>& list): This function is called when the input title of the book is not found. It has a run time of O(1) because it really only takes in input and calls the sortRecord() function to enter the new book into the database.

void chooseOne(int startingIndex, int numbooks, DLList<Record>& list): The choose function is an interesting one because it does 2 things. It prints out the books with the same name as the input title and then prints out the one selected by the user. It traverses the list one time to print out all the options for the user to see. The worst case of this function would be O(n), where the entire list is a list of books of the same name as the inputted title. The more average case would be Ω(1), or a constant, where there are only 2 or 3 books to display. There is a second for loop that prints out the chosen book. It traverses the DLList again and its worst case is also O(n), where the last node is the chosen node. So, it has to traverse to the end.

The while loop in the middle runs O(n) times, where n is the number of wrong inputs. It won't let you enter a number outside the given conditions. If there are 3 books to choose from, you must pick 1, 2, or 3. Anything else is invalid and it will make you try again.

The overall runtime for this function is still O(n).

void bookSearch(string name): bookSearch is run one time at the beginning to search for the title inputted

(a) Provide a list and description of classes or interfaces used by a program such as classes used to implement the data structures or exceptions.

I used the following classes and libraries for my programs: (taken from Part 2)

```
# include <iostream>
# include <fstream>
# include <vector>
# include <string>
# include <string.h>
using namespace std;
```

Most of these are standard. I used the "string.h" to use strcmp() in order to sort the Records lexicographically, or by the ASCII values of each letter. This was one of the harder things to implement. That is why it is assumed the user inputs the title using correct grammar and syntax.

We used the Record, TemplatedDDList class to implement part 2. I also include "Library.h", where I placed all the unique functions of Part 2.

We also used templated functions and classes in this program. The TemplatedDLList was of typename T and making this change from the DLList implementation allowed us to make lists of our user defined Record class.

(b) Include in the report the class declarations from a header file (.h) and their implementation from a sourcefile (.cpp).

```
// V. Rivera Casanova
// CSCE 221
// Leyk 199

#include "Record.h"
#include "TemplatedDLList.h"
#include "Library.h"

# include <iostream>
# include <fstream>
# include <vector>
# include <string>
# include <string.h>
using namespace std;
```

This is the declarations in Part 2. I'm including this one because it brings all the other Parts of the assignment together.

We have to include the h file and write the function definitions, such as the overloaded operators, outside the class scope for linking reasons. That is why I did not include, for example, a Library.cpp to go with Library.h file.

```
vector<DLList<Record>> library;
```

I also want to note that I used a global variable library in my main for Part 2. I thought this would be easier for accessibility reasons because so many of the functions need access to that data structure. And it would be a bit less complex, than keeping track of reference and pointers to this main structure of the program.

(c) Provide features of the C++ programming paradigms like Inheritance or Polymorphism in case of object-oriented programming, or Templates in the case of generic programming used in your implementation.

We didn't use inheritance or polymorphism in this program, but we did use template. I will give examples below:

```
template<typename T>
class DLList {
  private:
    int num;
    DLListNode<T> header, trailer;
// We have to include the template declaration at the beginning of the
class and that the beginning of each function signature.

// copy constructor
template<typename T>
DLList<T>::DLList(const DLList<T>& dll)
```

This was used in part 1 and part 2 to be able to make the DLList of type Record. (idk why the red lines are above; please ignore them)

4. A user guide description how to navigate your program with the instructions how to:

   (a) compile the program: specify the directory and file names, etc.

   Navigate to /pa1/221-A1-20b-Handout/221-A1-20b-code/. Here is the directories of each part of the homework:

   book_keeping: corresponds to Part 0.

   I used the command **g++ -std=c++17 -o book_keeping** main.cpp and **./book_keeping** to run this program.

   SimpleDLList: contains the first part of Part 1. I ran this using the given make file

   DLList-class and Templated-DLList-class: These two folders respectively contain the rest of Part 1 implementation. The command to run that I used for these two is are the ones provided also **./run-dll** and **./run-**tdll used with the makefile.

   application_of_dll: contains the implementation for Part 2. I don't know how to make a makefile, but since I am running it on Ubuntu, the linux like terminal, the command I used to run my program is **g++ -std=c++17 -Wall -Werror -Wpedantic library_main.cpp** and the**n ./a.out**

   (b) run the program: specify the name of an executable file.

   For the folders with the given makefiles, the executable files are DLList.o , DLList-main.o , TDLList-main.o

   For my directory application_of_dll the executable is a.out because of the run command that I used that I described above.

5. Specifications and description of input and output formats and files

   (a) The type of files: keyboard, text files, etc (if applicable).

   During this assignment we took in input from the keyboard (user), from a text file (.txt). We see this Parts 0 – 2. It is a big part of our project and it is also useful in debugging.

   The output was output to the console/terminal.

(b) A file input format: when a program requires a sequence of input items, specify the number of items per line or a line termination. Provide a sample of a required input format.

I used cin and getline() mainly to take in input from the user. getline() for reading in from the file, or for reading in a string from the user that contained spaces. cin does not take in input with spaces; I used that mainly for ints. Examples are below:

```
cout << "Enter a title of a book to search: ";
getline(cin, title);
--------------------------------------------------------------------------
while (chosen > ind || chosen < 1) { // O(n)
        cout << "Number entered is invalid!" << endl;
        cout << "Enter which version of the book you want (1, 2, ...): "
;
        cin >> chosen;
    }
```

Here is the loop inside the chooseOne() function that uses cin. It is only allowed to take in an int as input, which the instructions specified.

(c) Discuss possible cases when your program could crash because of incorrect input (a wrong file name, strings instead of a number, or such cases when the program expects 10 items to read and it finds only 9.)

The program can crash when the input is not what it is expecting, such as when the input stream is expecting an int and it gets a string. Thi scan happen at multiple locations throughout the program. During the file i/o section of this assignment, the program tends to crash if the first line if the file is not a blank line (that is how it was given to us) or if each Record information is not separated by a blank line. It will also crash if each record file has less than or greater than 5 lines for each record. The conditions were made to look for 5 consecutive lines of input separated by a lined blank line.

This program is also case-sensitive so “Harry” goes before “harry” in alphabetical order for Part 2, this is because I compare them based on ASCII codes and the lower cases letters have a higher value on that table. This makes sense for this program because book titles are usually capitalized, and we assume the user is using these correct conventions when writing the input.

6. Provide types of exceptions and their purpose in your program.

   (a) logical exceptions (such as deletion of an item from an empty container, etc.).

   For the cases where we try to remove an item from a DLList, we first check to see if the list is empty. Because if we don't our program will crash. In our definition for a DLList, we said a list is empty if the header points to the trailer. To check this, we have a user define exception, that is thrown when the list is empty, and you are trying to remove a node. We could give it a unique output message as well. This is used in Part 1.

   In part 2 since we are not removing nodes, there is no need to use it. I used the is_empty() function to check if the list was empty before inserting a node. If it was empty, I would insert the node first, if not, then the other conditions would run.

   (b) runtime exception (such as division by 0, etc.)

   Runtime exceptions can come up in multiple locations. One spot is when you are iterating through a DLList and you don't have a condition for when you get to the trailer. There is a function in the header file tha we can use in out loop conditions to avoif this error. bool *after_last_node() returns a pointer to the trailer and we make sure our node->next does not match this pointer.

   I could also run into a runtime exception in the chooseOne() function when trying to output the multiple records. In fact, I did because I mixed up my variables, so I changed my variable names to something easier to track.

7. Test your program for correctness using valid, invalid, and random inputs (e.g., insertion of an item at the beginning, at the end, or at a random place into a sorted vector). Include evidence of your testing, such as an output file or screen shots with an input and the corresponding output.

   I tested for Part 0, so I will exclude this part from this section.

SimpleDLList: I used the makefile to run this program.

```
rc@DESKTOP-241LE5C:~/csce221/pa1/221-A1-20b-Handout/221-A1-20b-code$ ls
LList-class  SimpleDLList  Templated-DLList-class  application_of_dll  book_keeping
rc@DESKTOP-241LE5C:~/csce221/pa1/221-A1-20b-Handout/221-A1-20b-code$
```

```
Makefile  SimpleDLList.cpp  simple
vrc@DESKTOP-241LE5C:~/csce221/pa1/221-A1-20b-Handout/221-A1-20b-code/SimpleDLList$ code SimpleDLList.cpp
vrc@DESKTOP-241LE5C:~/csce221/pa1/221-A1-20b-Handout/221-A1-20b-code/SimpleDLList$ make clean
rm *.o simple
rm: cannot remove '*.o': No such file or directory
Makefile:7: recipe for target 'clean' failed
make: *** [clean] Error 1
vrc@DESKTOP-241LE5C:~/csce221/pa1/221-A1-20b-Handout/221-A1-20b-code/SimpleDLList$ make
g++ -std=c++11 SimpleDLList.cpp -o simple
vrc@DESKTOP-241LE5C:~/csce221/pa1/221-A1-20b-Handout/221-A1-20b-code/SimpleDLList$ ./simple
Create a new list
list:

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

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

Delete the last 5 nodes
list: 100, 90, 80, 70, 60, 50, 40, 30, 20, 10, 10, 20, 30, 40, 50,

Delete the first 5 nodes
list: 50, 40, 30, 20, 10, 10, 20, 30, 40, 50,
vrc@DESKTOP-241LE5C:~/csce221/pa1/221-A1-20b-Handout/221-A1-20b-code/SimpleDLList$
```

DLList (with own tests for insert_after, insert_before, remove_after, remove_before) I added nodes before the last node and after the first node to test the insert functions.

```
vrc@DESKTOP-241LE5C:~/csce221/pa1/221-A1-20b-Handout/221-A1-20b-code/DLList-class$ make clean
rm *.o run-dll
vrc@DESKTOP-241LE5C:~/csce221/pa1/221-A1-20b-Handout/221-A1-20b-code/DLList-class$ make
c++ -g -std=c++11 -c DLList.cpp
c++ -g -std=c++11 -c DLList-main.cpp
c++ -g -std=c++11 DLList.o DLList-main.o -o run-dll
vrc@DESKTOP-241LE5C:~/csce221/pa1/221-A1-20b-Handout/221-A1-20b-code/DLList-class$ ./run-dll
Create a new list
list:


Insert 10 nodes at back with value 10,20,30,..,100
list:
Node: 0 -- 10
Node: 1 -- 20
Node: 2 -- 30
Node: 3 -- 40
Node: 4 -- 50
Node: 5 -- 60
Node: 6 -- 70
Node: 7 -- 80
Node: 8 -- 90
Node: 9 -- 100


Insert 10 nodes at front with value 10,20,30,..,100
list:
Node: 0 -- 100
Node: 1 -- 90
Node: 2 -- 80
Node: 3 -- 70
Node: 4 -- 60
Node: 5 -- 50
Node: 6 -- 40
Node: 7 -- 30
Node: 8 -- 20
Node: 9 -- 10
Node: 10 -- 10
Node: 11 -- 20
Node: 12 -- 30
Node: 13 -- 40
Node: 14 -- 50
Node: 15 -- 60
Node: 16 -- 70
Node: 17 -- 80
Node: 18 -- 90
Node: 19 -- 100


Copy to a new list
list2:
Node: 0 -- 100
```

```
Copy to a new list
list2:
Node: 0 -- 100
Node: 1 -- 90
Node: 2 -- 80
Node: 3 -- 70
Node: 4 -- 60
Node: 5 -- 50
Node: 6 -- 40
Node: 7 -- 30
Node: 8 -- 20
Node: 9 -- 10
Node: 10 -- 10
Node: 11 -- 20
Node: 12 -- 30
Node: 13 -- 40
Node: 14 -- 50
Node: 15 -- 60
Node: 16 -- 70
Node: 17 -- 80
Node: 18 -- 90
Node: 19 -- 100

Assign to another new list
list3:
Node: 0 -- 100
Node: 1 -- 90
Node: 2 -- 80
Node: 3 -- 70
Node: 4 -- 60
Node: 5 -- 50
Node: 6 -- 40
Node: 7 -- 30
Node: 8 -- 20
Node: 9 -- 10
Node: 10 -- 10
Node: 11 -- 20
Node: 12 -- 30
Node: 13 -- 40
Node: 14 -- 50
Node: 15 -- 60
Node: 16 -- 70
Node: 17 -- 80
Node: 18 -- 90
Node: 19 -- 100

Delete the last 10 nodes
list:
Node: 0 -- 100
Node: 1 -- 90
Node: 2 -- 80
Node: 3 -- 70
Node: 4 -- 60
Node: 5 -- 50
Node: 6 -- 40
Node: 7 -- 30
Node: 8 -- 20
Node: 9 -- 10

Delete the first 10 nodes
list:
```

1

```
Delete the first 10 nodes
list:


Make sure the other two lists are not affected.
list2:
Node: 0 -- 100
Node: 1 -- 90
Node: 2 -- 80
Node: 3 -- 70
Node: 4 -- 60
Node: 5 -- 50
Node: 6 -- 40
Node: 7 -- 30
Node: 8 -- 20
Node: 9 -- 10
Node: 10 -- 10
Node: 11 -- 20
Node: 12 -- 30
Node: 13 -- 40
Node: 14 -- 50
Node: 15 -- 60
Node: 16 -- 70
Node: 17 -- 80
Node: 18 -- 90
Node: 19 -- 100

list3:
Node: 0 -- 100
Node: 1 -- 90
Node: 2 -- 80
Node: 3 -- 70
Node: 4 -- 60
Node: 5 -- 50
Node: 6 -- 40
Node: 7 -- 30
Node: 8 -- 20
Node: 9 -- 10
Node: 10 -- 10
Node: 11 -- 20
Node: 12 -- 30
Node: 13 -- 40
Node: 14 -- 50
Node: 15 -- 60
Node: 16 -- 70
Node: 17 -- 80
Node: 18 -- 90
Node: 19 -- 100

My Own Tests::
list2:
```

2

```
My Own Tests::
list2:
Node: 0 -- 100
Node: 1 -- 500
Node: 2 -- 90
Node: 3 -- 80
Node: 4 -- 70
Node: 5 -- 60
Node: 6 -- 50
Node: 7 -- 40
Node: 8 -- 30
Node: 9 -- 20
Node: 10 -- 10
Node: 11 -- 10
Node: 12 -- 20
Node: 13 -- 30
Node: 14 -- 40
Node: 15 -- 50
Node: 16 -- 60
Node: 17 -- 70
Node: 18 -- 80
Node: 19 -- 90
Node: 20 -- 500
Node: 21 -- 100

list3:
Node: 0 -- 100
Node: 1 -- 90
Node: 2 -- 80
Node: 3 -- 70
Node: 4 -- 60
Node: 5 -- 50
Node: 6 -- 40
Node: 7 -- 30
Node: 8 -- 20
Node: 9 -- 10
Node: 10 -- 10
Node: 11 -- 20
Node: 12 -- 30
Node: 13 -- 40
Node: 14 -- 50
Node: 15 -- 60
Node: 16 -- 70
Node: 17 -- 80
Node: 18 -- 90
Node: 19 -- 100

vrc@DESKTOP-241LE5C:~/csce221/pa1/221-A1-20b-Handout/221-A1-20b-code/DLList-class$
```

<-Here you can see I inserted nodes with value 500 and at Node 20.

3

Testing the remove functions:

```
My Own Tests 2::
list2:
Node: 0 -- 100
Node: 1 -- 90
Node: 2 -- 80
Node: 3 -- 70
Node: 4 -- 60
Node: 5 -- 50
Node: 6 -- 40
Node: 7 -- 30
Node: 8 -- 20
Node: 9 -- 10
Node: 10 -- 10
Node: 11 -- 20
Node: 12 -- 30
Node: 13 -- 40
Node: 14 -- 50
Node: 15 -- 60
Node: 16 -- 70
Node: 17 -- 80
Node: 18 -- 90
Node: 19 -- 100

list3:
Node: 0 -- 100
Node: 1 -- 90
Node: 2 -- 80
Node: 3 -- 70
Node: 4 -- 60
Node: 5 -- 50
Node: 6 -- 40
Node: 7 -- 30
Node: 8 -- 20
Node: 9 -- 10
Node: 10 -- 10
Node: 11 -- 20
Node: 12 -- 30
Node: 13 -- 40
Node: 14 -- 50
Node: 15 -- 60
Node: 16 -- 70
Node: 17 -- 80
Node: 18 -- 90
Node: 19 -- 100

500 500
vrc@DESKTOP-241LE5C:~/csce221/pa1/221-A1-20b-Handou
```

Test Code:

```
// more testing...
// add tests for insert_after, insert_before
// add tests for remove_after, remove_before
dll2.insert_after((*dll2.first_node()), 500);
dll2.insert_before((*dll2.after_last_node()->prev), 500);

cout << "My Own Tests::" << endl;
cout << "list2: " << dll2 << endl;
cout << "list3: " << dll3 << endl;

int remove1 = dll2.remove_after((*dll2.first_node()));
int remove2 = dll2.remove_before((*dll2.after_last_node()->prev));


cout << "My Own Tests 2::" << endl;
cout << "list2: " << dll2 << endl;
cout << "list3: " << dll3 << endl;

cout << remove1 << " "<< remove2 << endl;

return 0;
```

TemplatedDLList:

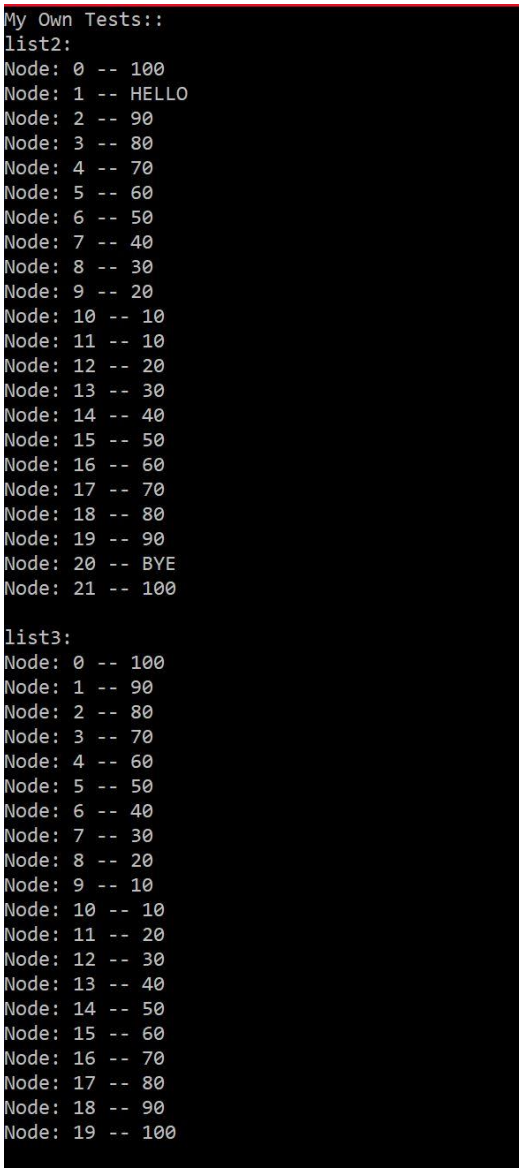
```
vrc@DESKTOP-241LE5C:~/csce221/pa1/221-A1-20b-Handout/221-A1-20b-code/Templated-DLList-class$ make clean
rm *.o run-tdll
rm: cannot remove '*.o': No such file or directory
rm: cannot remove 'run-tdll': No such file or directory
Makefile:10: recipe for target 'clean' failed
make: *** [clean] Error 1
vrc@DESKTOP-241LE5C:~/csce221/pa1/221-A1-20b-Handout/221-A1-20b-code/Templated-DLList-class$ make
c++ -std=c++11 -c TDLList-main.cpp
c++ -std=c++11 TDLList-main.o -o run-tdll
vrc@DESKTOP-241LE5C:~/csce221/pa1/221-A1-20b-Handout/221-A1-20b-code/Templated-DLList-class$ ./run-tdll
Create a new list
list:


Insert 10 nodes at back with value 10,20,30,..,100
list:
Node: 0 -- 10
Node: 1 -- 20
Node: 2 -- 30
Node: 3 -- 40
Node: 4 -- 50
Node: 5 -- 60
Node: 6 -- 70
Node: 7 -- 80
Node: 8 -- 90
Node: 9 -- 100


Insert 10 nodes at front with value 10,20,30,..,100
list:
Node: 0 -- 100
Node: 1 -- 90
Node: 2 -- 80
Node: 3 -- 70
Node: 4 -- 60
Node: 5 -- 50
Node: 6 -- 40
Node: 7 -- 30
Node: 8 -- 20
Node: 9 -- 10
Node: 10 -- 10
Node: 11 -- 20
Node: 12 -- 30
Node: 13 -- 40
Node: 14 -- 50
Node: 15 -- 60
Node: 16 -- 70
Node: 17 -- 80
Node: 18 -- 90
Node: 19 -- 100
```

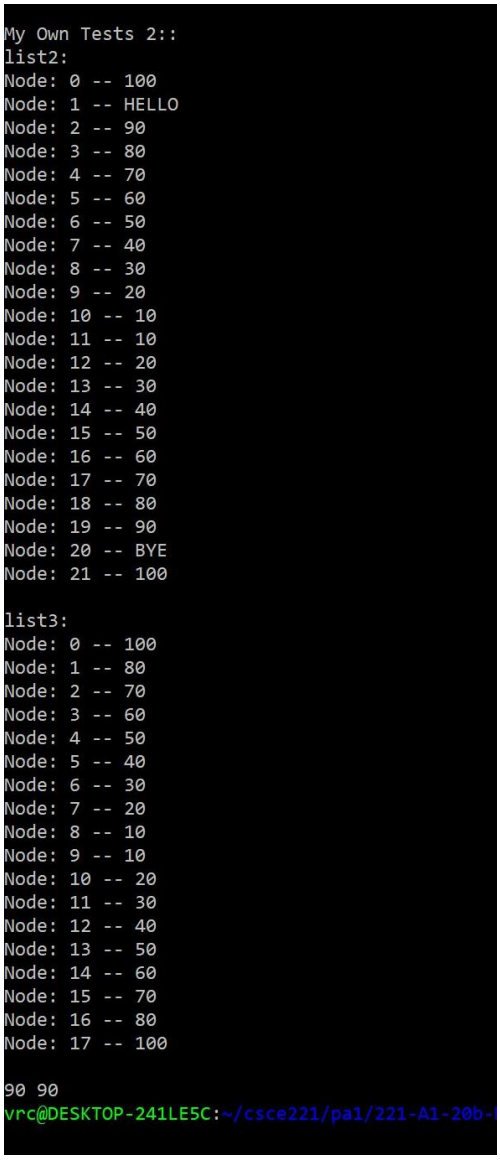
```
Node: 19 -- 100


Copy to a new list
list2:
Node: 0 -- 100
Node: 1 -- 90
Node: 2 -- 80
Node: 3 -- 70
Node: 4 -- 60
Node: 5 -- 50
Node: 6 -- 40
Node: 7 -- 30
Node: 8 -- 20
Node: 9 -- 10
Node: 10 -- 10
Node: 11 -- 20
Node: 12 -- 30
Node: 13 -- 40
Node: 14 -- 50
Node: 15 -- 60
Node: 16 -- 70
Node: 17 -- 80
Node: 18 -- 90
Node: 19 -- 100


Assign to another new list
list3:
Node: 0 -- 100
Node: 1 -- 90
Node: 2 -- 80
Node: 3 -- 70
Node: 4 -- 60
Node: 5 -- 50
Node: 6 -- 40
Node: 7 -- 30
Node: 8 -- 20
Node: 9 -- 10
Node: 10 -- 10
Node: 11 -- 20
Node: 12 -- 30
Node: 13 -- 40
Node: 14 -- 50
Node: 15 -- 60
Node: 16 -- 70
Node: 17 -- 80
Node: 18 -- 90
Node: 19 -- 100


Delete the last 10 nodes
list:
Node: 0 -- 100
Node: 1 -- 90
Node: 2 -- 80
Node: 3 -- 70
Node: 4 -- 60
Node: 5 -- 50
Node: 6 -- 40
Node: 7 -- 30
Node: 8 -- 20
Node: 9 -- 10


Delete the first 10 nodes
list:
```

1

```
Make sure the other two lists are not affected.
list2:
Node: 0 -- 100
Node: 1 -- 90
Node: 2 -- 80
Node: 3 -- 70
Node: 4 -- 60
Node: 5 -- 50
Node: 6 -- 40
Node: 7 -- 30
Node: 8 -- 20
Node: 9 -- 10
Node: 10 -- 10
Node: 11 -- 20
Node: 12 -- 30
Node: 13 -- 40
Node: 14 -- 50
Node: 15 -- 60
Node: 16 -- 70
Node: 17 -- 80
Node: 18 -- 90
Node: 19 -- 100

list3:
Node: 0 -- 100
Node: 1 -- 90
Node: 2 -- 80
Node: 3 -- 70
Node: 4 -- 60
Node: 5 -- 50
Node: 6 -- 40
Node: 7 -- 30
Node: 8 -- 20
Node: 9 -- 10
Node: 10 -- 10
Node: 11 -- 20
Node: 12 -- 30
Node: 13 -- 40
Node: 14 -- 50
Node: 15 -- 60
Node: 16 -- 70
Node: 17 -- 80
Node: 18 -- 90
Node: 19 -- 100
```

2


My Own Tests::
list2:
Node: 0 -- 100
Node: 1 -- HELLO
Node: 2 -- 90
Node: 3 -- 80
Node: 4 -- 70
Node: 5 -- 60
Node: 6 -- 50
Node: 7 -- 40
Node: 8 -- 30
Node: 9 -- 20
Node: 10 -- 10
Node: 11 -- 10
Node: 12 -- 20
Node: 13 -- 30
Node: 14 -- 40
Node: 15 -- 50
Node: 16 -- 60
Node: 17 -- 70
Node: 18 -- 80
Node: 19 -- 90
Node: 20 -- BYE
Node: 21 -- 100
list3:
Node: 0 -- 100
Node: 1 -- 90
Node: 2 -- 80
Node: 3 -- 70
Node: 4 -- 60
Node: 5 -- 50
Node: 6 -- 40
Node: 7 -- 30
Node: 8 -- 20
Node: 9 -- 10
Node: 10 -- 10
Node: 11 -- 20
Node: 12 -- 30
Node: 13 -- 40
Node: 14 -- 50
Node: 15 -- 60
Node: 16 -- 70
Node: 17 -- 80
Node: 18 -- 90
Node: 19 -- 100


My Own Tests 2::
list2:
Node: 0 -- 100
Node: 1 -- HELLO
Node: 2 -- 90
Node: 3 -- 80
Node: 4 -- 70
Node: 5 -- 60
Node: 6 -- 50
Node: 7 -- 40
Node: 8 -- 30
Node: 9 -- 20
Node: 10 -- 10
Node: 11 -- 10
Node: 12 -- 20
Node: 13 -- 30
Node: 14 -- 40
Node: 15 -- 50
Node: 16 -- 60
Node: 17 -- 70
Node: 18 -- 80
Node: 19 -- 90
Node: 20 -- BYE
Node: 21 -- 100
list3:
Node: 0 -- 100
Node: 1 -- 80
Node: 2 -- 70
Node: 3 -- 60
Node: 4 -- 50
Node: 5 -- 40
Node: 6 -- 30
Node: 7 -- 20
Node: 8 -- 10
Node: 9 -- 10
Node: 10 -- 20
Node: 11 -- 30
Node: 12 -- 40
Node: 13 -- 50
Node: 14 -- 60
Node: 15 -- 70
Node: 16 -- 80
Node: 17 -- 100
90 90
vrc@DESKTOP-241LE5C:~/csce221/pa1/221-A1-20b-

Test code:

```
// more testing...
// add tests for insert_after, insert_before
// add tests for remove_after, remove_before
dll2.insert_after((*dll2.first_node()), "HELLO");
dll2.insert_before((*dll2.after_last_node()->prev), "BYE");

cout << "My Own Tests::" << endl;
cout << "list2: " << dll2 << endl;
cout << "list3: " << dll3 << endl;

string  remove1 = dll2.remove_after((*dll3.first_node()));
string remove2 = dll2.remove_before((*dll3.after_last_node()->prev));


cout << "My Own Tests 2::" << endl;
cout << "list2: " << dll2 << endl;
cout << "list3: " << dll3 << endl;

cout << remove1 << " "<< remove2 << endl;

return 0;
```

Part 2 Testing: I'm going to do scenarios for each situation:

1. Finding a unique book

```
Enter a title of a book to search: Echo
Entering library database 4

Book found!
Title: Echo
Author: Pam Munoz Ryan
ISBN: 978-0439874021
Year Published: 2015
Edition: 1st edition
vrc@DESKTOP-241LE5C:~/csce221/pa1/221-A1-20b-Handout/221-A1-20b-code/application_of_dll$
```

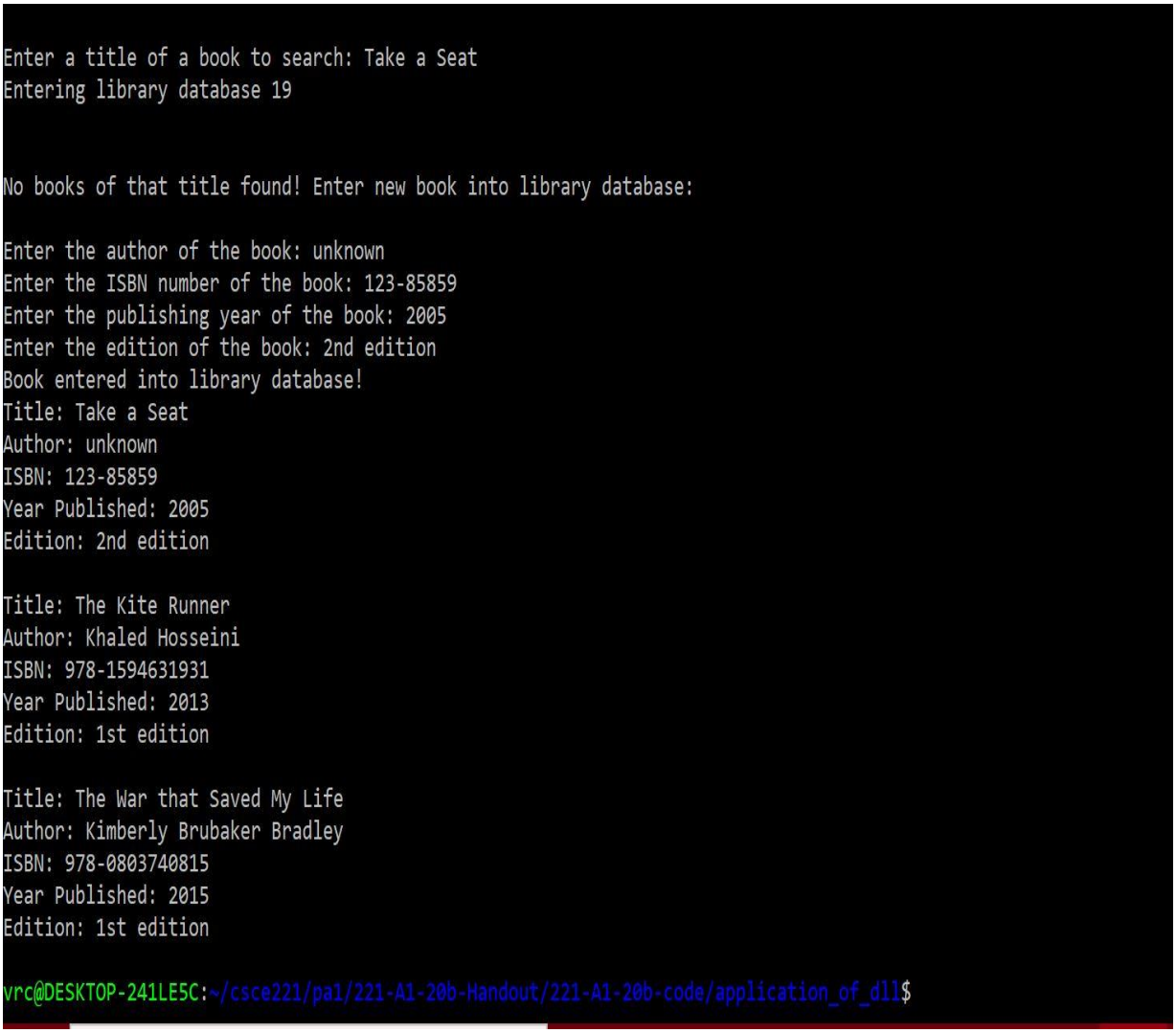
2. Finding multiple books with same title and choosing one

```
Enter a title of a book to search: Harry Potter and the Chamber of Secrets
Entering library database 7




Multiple entries with same name found.
Title: Harry Potter and the Chamber of Secrets
Author: J. K. Rowling
ISBN: 978-0439064871
Year Published: 2000
Edition: 2nd edition

Title: Harry Potter and the Chamber of Secrets
Author: J. K. Rowling
ISBN: 978-0439064873
Year Published: 2000
Edition: 1st edition

Enter which version of the book you want (1, 2, ...): 1
Title: Harry Potter and the Chamber of Secrets
Author: J. K. Rowling
ISBN: 978-0439064871
Year Published: 2000
Edition: 2nd edition

vrc@DESKTOP-241LE5C:~/csce221/pa1/221-A1-20b-Handout/221-A1-20b-code/application_of_dll$
```

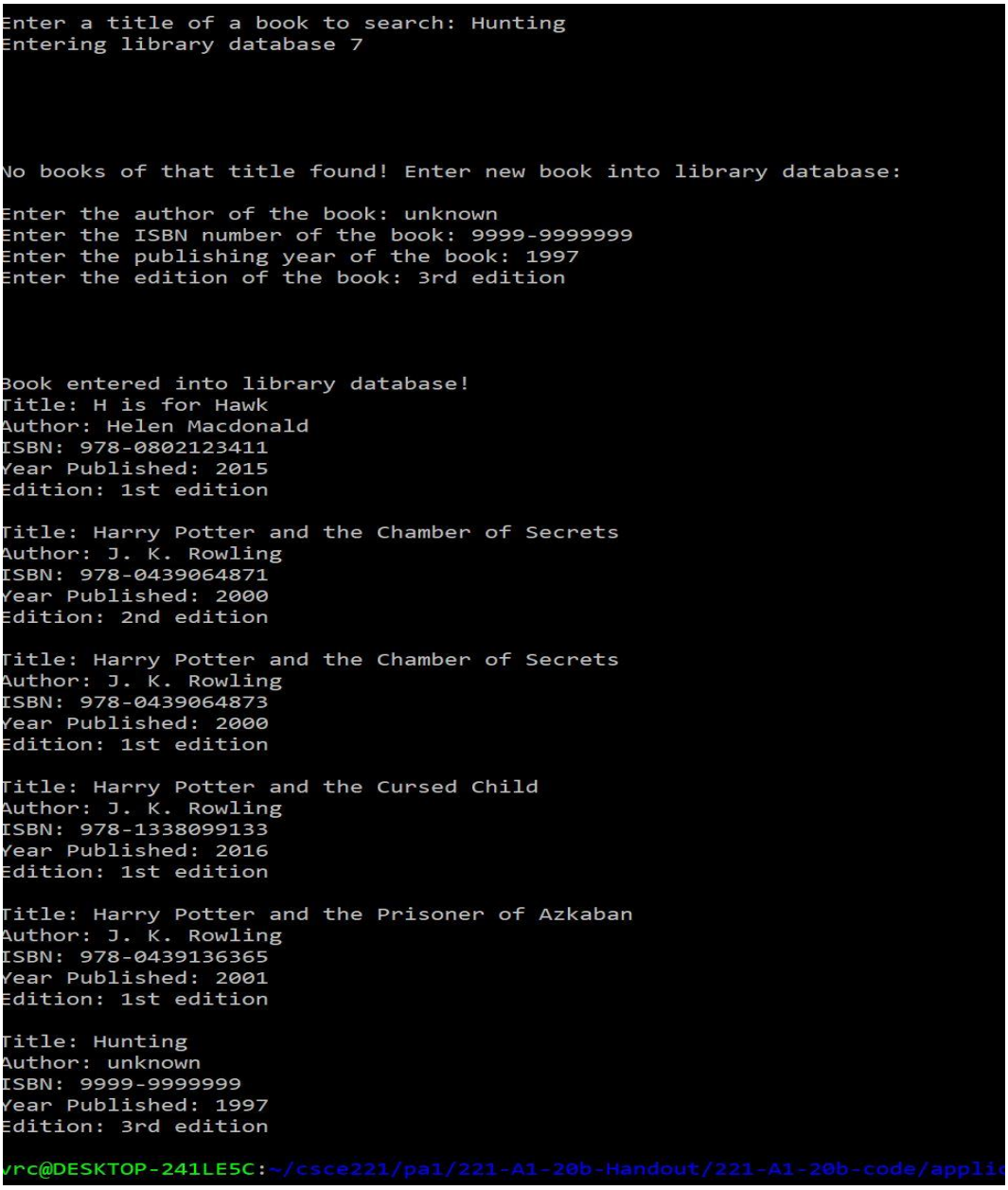
3. Inserting a new book into an empty list

```
Enter a title of a book to search: Victoria
Entering library database 21
No books of that title found! Enter new book into library database:

Enter the author of the book: Victoria Rivera Casanova
Enter the ISBN number of the book: 123-456798
Enter the publishing year of the book: 2020
Enter the edition of the book: 1st edition
Book entered into library database!
Title: Victoria
Author: Victoria Rivera Casanova
ISBN: 123-456798
Year Published: 2020
Edition: 1st edition

vrc@DESKTOP-241LE5C:~/csce221/pa1/221-A1-20b-Handout/221-A1-20b-code/application_of_dll$
```

4. Inserting book at the beginning of a list



5. Inserting a new book at the end of the list



6. Inserting a book at the middle of the list

```
Enter a title of a book to search: Harry Potter and the Half Blood Prince
Entering library database 7




No books of that title found! Enter new book into library database:

Enter the author of the book: J.K. Rowling
Enter the ISBN number of the book: 1212-121212
Enter the publishing year of the book: 2000
Enter the edition of the book: 3rd edition




Book entered into library database!
Title: H is for Hawk
Author: Helen Macdonald
ISBN: 978-0802123411
Year Published: 2015
Edition: 1st edition

Title: Harry Potter and the Chamber of Secrets
Author: J. K. Rowling
ISBN: 978-0439064871
Year Published: 2000
Edition: 2nd edition

Title: Harry Potter and the Chamber of Secrets
Author: J. K. Rowling
ISBN: 978-0439064873
Year Published: 2000
Edition: 1st edition

Title: Harry Potter and the Cursed Child
Author: J. K. Rowling
ISBN: 978-1338099133
Year Published: 2016
Edition: 1st edition

Title: Harry Potter and the Half Blood Prince
Author: J.K. Rowling
ISBN: 1212-121212
Year Published: 2000
Edition: 3rd edition

Title: Harry Potter and the Prisoner of Azkaban
Author: J. K. Rowling
ISBN: 978-0439136365
Year Published: 2001
Edition: 1st edition

vrc@DESKTOP-241LE5C:~/csce221/pa1/221-A1-20b-Handout/221-A1-20b-code/application_of_dll$
```

Questions from instructions:
I described the Record class in the pages above regarding my unique implementation.

For the seven functions in Part one I have describe the insert_first(), insert_last(), remove_last(), remove_first(), these functions have a runtime of O(1). This is because there are no loops to go through because we have direct access to the necessary nodes by using the header and trailer nodes. For the others such as the copy constructor, copy assignment and the move assignment operator, they run O(n) where n is the number of nodes in the DLList. This is because they travers the entire list to do a deep copy (for the copy constructors and copy assignment), and move constructor the runtime should be O(1) because it is only transferring ownership and not doing a deep copy.

For the next question, I talked about this in the previous pages also, with the unique functions I used to implement Part 2 and how they work with the main function and input file stream.

The average run time complexity for the insert and search functions were would be O(1). For these purposes most of the lists were empty do the searching algorithm would only need to traverse one node if that. It would actually go to insert_first(); The insert is the same because there wasn't many Records already inserted it would be closer to O(1) average runtime.

Say there are n Records in the .txt or the user is allowed to enter multiple books in one given session. The runtime for the entire function would still ne O(n). This is because Each individual function has a worst case of O(n) and this is mainly due to the sorting algorithm function. The program has no nested loops that add on to the overall complexity.