CSCI 2720
Assignment 3 - **Doubly Linked Lists**
Due: October 17th, 2021 11:59 PM

This assignment is to implement a **Sorted Doubly Linked List**. This program must accept all the basic commands as described later in this document.

Unlike with Assignment 2, you will not need ItemType for this assignment. Instead, you will use C++ templates to make your programs support three different data types (**`int`**, **`float`**, and `std::`**`string`**). The data type will be specified by the user before any operations are run, as described later in the document.

Important Points
1. Create one folder named DoublyLinkedList. Main.cpp, DoublyLinkedList.h and DoublyLinkedList.cpp must be in the DoublyLinkedList folder.

2. In this assignment, place your NodeType definition in DoublyLinkedList.h.

3. You will not need ItemType, and will instead make your program "generic" using templates.

4. You will use 3 input files: int-input.txt, float-input.txt, and string-input.txt.

5. You should allow duplicate values to be stored in your data structure.

6. Be sure to properly document your code with comments, and add your name above functions that you implement if you are in a group.

7. You must follow the exact submission instructions given at the end of the document.

Templates
You should make your linked list class templated, as well as the NodeType struct, by adding `template<class T>` before the class/struct definition. For example, to make NodeType and DoublyLinkedList templated, use the following code snippet in DoublyLinkedList.h:

```
template<class T>
struct NodeType {
      // NodeType members
};
```

```
template<class T>
class DoublyLinkedList {
      // DoublyLinkedList members
};
```

Note that the "T" works just as the parameter for generics in Java, and may be replaced by something else, such as "U". However, unlike with Java, you must specify what data types you are planning to support, which can be done in your .cpp files. For example, to do this for DoublyLinkedList, add the following lines of code at the bottom of DoublyLinkedList.cpp, below all your method implementations:

```
template class DoublyLinkedList<int>;
template class DoublyLinkedList<float>;
template class DoublyLinkedList<std::string>;
```

Your program should support storing data of type `int`, `float`, and `std::string` depending on input taken from the user at the very beginning of the program. The user should be able to enter "i" for `int`, "f" for `float` or "s" for `std::string`. Please see the following sample output:

```
./main input.txt
Enter list type (i - int, f - float, s - std:string): s
```

In the above example, the user has provided "s" as the input. In this case, you should initialize your generic list to store `std::string` values. This means you should also read the values in the given text file as `std::string` values and enter them to the list as `std::string` objects. The insert, delete and print commands should likewise support `std::string` inputs. If the user provides "i" or "f", your program should be able to work with ints or floats respectively.

The user is responsible for selecting an appropriate type for a given program run depending on the provided input file and the values that are expected to be stored in the list. You will be provided sample text files for each appropriate data type: one for `int`, one for `float`, and one for `std::string`.

DoublyLinkedList.h should be composed of a struct called NodeType and the following function/member declarations:

NodeType should contain the members:

```
T data;
NodeType<T> *next;
NodeType<T> *back;
```

Public functions:

`DoublyLinkedList()`

Post-Condition: the list is created.

`~DoublyLinkedList()`

Pre-Condition: the list is created.
Post-Condition: all nodes are freed.

`void insertItem(T &item)`

Pre-Condition: the list exists and item is initialized
Post-Condition: the item is inserted into the list, maintaining sorted order.

`void deleteItem(T &item)`

Pre-Condition: the list exists and item is initialized.
Post-Condition: the node that contains item is removed from the list. If the item is not present in the list, print the message that is shown in the example output.

`int lengthIs() const`

Pre-Condition: the list exists.
Post-Condition: return the length instance variable.

`void print()`

Pre-Condition: the list exists.
Post-Condition: items in the list are printed to standard output.

`void printReverse()`

Pre-Condition: the list exists.
Post-Condition: items in the list are printed to standard output in reverse order.

The following functions can be implemented however you like, just make sure their input and output formatting matches the sample output below. Implement these functions as a class function using prototypes of your choice.

`deleteSubsection` function - This function will take input from the user for the lower and upper bound (both inclusive) for a range of values that you will delete from the list

- Example:
  - Enter lower bound: 9
  - Enter upper bound: 34
  - Original List: 3 5 10 20 34 56
  - Modified List: 3 5 56

So in the example above you have deleted all the numbers between 9 and 34 from the list (including 34)

- Note: If the list has no values in the range, don’t delete anything.
  - Enter lower bound: 15
  - Enter upper bound: 18
  - Original List: 3 5 10 20 34 56
  - Modified List: 3 5 10 20 34 56
- Note: If the list is empty, do nothing.
  - Enter lower bound: 3
  - Enter upper bound: 45
  - Original List:
  - Modified List:

In the readme file give the pseudo code (steps) for your deleteSubsection operation. Using this pseudocode, explain the complexity (big O) of your deleteSubsection operation.

mode function - This function will return the mode for the current list. You should NOT use any type of array to implement this function. Because the list is sorted, you should be able to find the mode without using an array.

- Example:
  - 2 4 4 9 9 9 9 35 78 104
  - Mode: 9
- Note: If multiple values are the mode, just list the first one in the list:
  - 4 7 8 8 8 10 10 10 14 67
  - Mode: 8

In the readme file give the pseudo code (steps) for your Mode operation. Using this pseudocode, explain the complexity (big O) of your Mode operation.

swapAlternate function - This function will swap every other node in the list. (For example: swaps nodes 1 and 2, nodes 3 and 4, nodes 5 and 6 and so on)

Note: Implement this function by changing pointers to nodes only. Do not copy the item type objects inside nodes and swap them to implement this function.

- Example:
  - Original List: 3 5 10 20 34 56
  - Swapped List: 5 3 20 10 56 34
- Note: If there is an odd number of nodes, do the swap all the way to the last node and leave the last node at the end:

    - `Original List: 45 67 89 102 120`
    - `Swapped List: 67 45 102 89 120`
- `Note: If the list is empty, do nothing.`
    - `Original List:`
    - `Swapped List:`

In the readme file give the pseudo code (steps) for your swapAlt operation. Using this pseudocode, explain the complexity (big O) of your swapAlt operation.

Private members and functions:

```
NodeType<T> *head
NodeType<T> *tail
```

DoublyLinkedList.cpp should implement the struct and the functions listed in DoublyLinkedList.h.

Sample Output:

You must use the exact same command characters and instruction phrases in your programs.

Sample Output 1 (Doubly Linked List) - int-intput.txt:

```
./main int-input.txt
Enter list type (i - int, f - float, s - std::string): i
insert (i), delete (d), length (l), print (p), deleteSub (b), mode (m),
printReverse(r), swapAtl(s), quit (q)
```

//1. Test PRINT(p)

```
Enter a command: p
3 9 10 19 37 45 63 84 100
```

//2. Test PRINTREVERSE(r)

```
Enter a command: r
100 84 63 45 37 19 10 9 3
```

//3. Test LENGTH(l)

```
Enter a command: l
The length is: 9
```

//4. Test INSERT(i)

//4a. Insert the first element

```
Enter a command: i
Item to insert: 1
1 3 9 10 19 37 45 63 84 100
```
//4b. Insert at the middle/end
```
Enter a command: i
Item to insert: 25
1 3 9 10 19 25 37 45 63 84 100

Enter a command: i
Item to insert: 150
1 3 9 10 19 25 37 45 63 84 100 150
```
//4c. Insert into empty list
//   (Check it below after delete)
//5.Test DELETE(d)
//5a. Delete first element
```
Enter a command: d
Item to delete: 1
3 9 10 19 25 37 45 63 84 100 150
```
//5b. Delete last element or an element in the middle
```
Enter a command: d
Item to delete: 37
3 9 10 19 25 45 63 84 100 150

Enter a command: d
Item to delete: 150
3 9 10 19 25 45 63 84 100
```
//5c. Delete a non-existing item
```
Enter a command: d
Item to delete: 50
Item not in list!
3 9 10 19 25 45 63 84 100
```
//5d. Continue deleting until the only element left is 100 and delete the last element and then try to delete from an empty list
```
Enter a command: p
100

Enter a command: d
Item to delete: 100



Enter a command: d
```

```
Item to delete: 20
You cannot delete from an empty list.
```
//4c. Insert into empty list
```
Enter a command: i
Item to insert: 45
45
```
//6. Test QUIT(q)
```
Enter a command: q
Quitting program...
```

Sample Output 2 (Doubly Linked List) - float-input.txt:

```
./main float-input.txt
Enter list type (i - int, f - float, s - std::string): f
insert (i), delete (d), length (l), print (p), deleteSub (b), mode (m),
printReverse(r), swapAtl(s), quit (q)
```
//1. Test PRINT(p)
```
Enter a command: p
3.1 9.6 10.3 19.3 37.3 45.9 63.5 84.5 100.7
```
//2. Test PRINTREVERSE(r)
```
Enter a command: r
100.7 84.5 63.5 45.9 37.3 19.3 10.3 9.6 3.1
```
//3. Test LENGTH(l)
```
Enter a command: l
The length is: 9
```
//4. Test INSERT(i)
//4a. Insert the first element
```
Enter a command: i
Item to insert: 1.2
1.2 3.1 9.6 10.3 19.3 37.3 45.9 63.5 84.5 100.7
```
//4b. Insert at the middle/end
```
Enter a command: i
Item to insert: 37.2
1.2 3.1 9.6 10.3 19.3 37.2 37.3 45.9 63.5 84.5 100.7

Enter a command: i
Item to insert: 150.2
1.2 3.1 9.6 10.3 19.3 37.2 37.3 45.9 63.5 84.5 100.7 150.2
```
//4c. Insert into empty list
//   (Check it below after delete)

//5.Test DELETE(d)

//5a. Delete first element

```
Enter a command: d
Item to delete: 1.2
3.1 9.6 10.3 19.3 37.2 37.3 45.9 63.5 84.5 100.7 150.2
```

//5b. Delete last element or an element in the middle

```
Enter a command: d
Item to delete: 37.3
3.1 9.6 10.3 19.3 37.2 45.9 63.5 84.5 100.7 150.2

Enter a command: d
Item to delete: 150.2
3.1 9.6 10.3 19.3 37.2 45.9 63.5 84.5 100.7
```

//5c. Delete a non-existing item

```
Enter a command: d
Item to delete: 19.1
Item not in list!
3.1 9.6 10.3 19.3 37.2 45.9 63.5 84.5 100.7
```

//5d. Continue deleting until the only element left is 100.7 and delete the last element then try to delete from an empty list

```
Enter a command: p
100.7

Enter a command: d
Item to delete: 100.7



Enter a command: d
Item to delete: 20.2
You cannot delete from an empty list.
```

//4c. Insert into empty list

```
Enter a command: i
Item to insert: 45.6
45.6
```

//6. Test QUIT(q)

```
Enter a command: q
Quitting program...
```

<u>Example Output 3 (Doubly Linked List) - string-input.txt:</u>

```
./main string-input.txt
Enter list type (i - int, f - float, s - std::string): s
insert (i), delete (d), length (l), print (p), deleteSub (b), mode (m),
printReverse(r), swapAtl(s), quit (q)
```
//1. Test PRINT(p)
```
Enter a command: p
Craig Dern Ford Goodman Macy
```
//2. Test PRINTREVERSE(r)
```
Enter a command: r
Macy Goodman Ford Dern Craig
```
//3. Test LENGTH(l)
```
Enter a command: l
The length is: 5
```
//4. Test INSERT(i)

//4a. Insert the first element
```
Enter a command: i
Item to insert: Aba
Aba Craig Dern Ford Goodman Macy
```
//4b. Insert at the middle/end
```
Enter a command: i
Item to insert: Elsa
Aba Craig Dern Elsa Ford Goodman Macy

Enter a command: i
Item to insert: Roger
Aba Craig Dern Elsa Ford Goodman Macy Roger
```
//4c. Insert into empty list

//   (Check it below after delete)

//5.Test DELETE(d)

//5a. Delete first element
```
Enter a command: d
Item to delete: Aba
Craig Dern Elsa Ford Goodman Macy Roger
```
//5b. Delete last element or an element in the middle
```
Enter a command: d
Item to delete: Ford
Craig Dern Elsa Goodman Macy Roger

Enter a command: d
Item to delete: Roger
```

```
Craig Dern Elsa Goodman Macy
```
//5c. Delete a non-existing item
```
Enter a command: d
Item to delete: Willis
Item not in list!
Craig Dern Elsa Goodman Macy
```
//5d. Continue deleting until the only element left is Macy and delete the last element then try to delete from an empty list
```
Enter a command: p
Macy

Enter a command: d
Item to delete: Macy



Enter a command: d
Item to delete: Craig
You cannot delete from an empty list.
```
//4c. Insert into empty list
```
Enter a command: i
Item to insert: John
John
```
//6. Test QUIT(q)
```
Enter a command: q
Quitting program...
```

<u>Example Output 4 (Doubly Linked List) - int-input.txt:</u>

```
./main int-input.txt
Enter list type (i - int, f - float, s - std::string): i
insert (i), delete (d), length (l), print (p), deleteSub (b), mode (m),
printReverse(r), swapAtl(s), quit (q)
```
//1. Test DELETESUB(b)
```
Enter a command: b
Enter lower bound: 19
Enter upper bound: 68
Original List: 3 9 10 19 37 45 63 84 100
Modified List: 3 9 10 84 100
```

Example Output 5 (Doubly Linked List) - int-input.txt:

```
./main int-input.txt
Enter list type (i - int, f - float, s - std::string): i
insert (i), delete (d), length (l), print (p), deleteSub (b), mode (m),
printReverse(r), swapAtl(s), quit (q)
Enter a command: p
3 9 10 19 37 45 63 84 100
```

//1. Test SWAPALT(s)

```
Enter a command: s
Original List: 3 9 10 19 37 45 63 84 100
Swapped List: 9 3 19 10 45 37 84 63 100
```

Example Output 6 (Doubly Linked List) - int-input2.txt:

```
./main int-input2.txt
Enter list type (i - int, f - float, s - std::string): i
insert (i), delete (d), length (l), print (p), deleteSub (b), mode (m),
printReverse(r), swapAtl(s), quit (q)
Enter a command: p
2 5 7 7 19 21 21 21 25 32 41
```

//1. Test MODE(m)

```
Enter a command: m
2 5 7 7 19 21 21 21 25 32 41
Mode: 21
```

NOTE: DoublyLinkedList should support all 3 data types.

Grading Rubric:

Doubly Linked List	
insert()	10%
delete()	10%
print()	5%
length()	5%
printReverse()	5%
Delete Subsection	10%
Mode	10%
Swap Alternate	10%
~DoublyLinkedList()	5%

Works on all three input types	10%
Common Implementations	
Following the specifications	5%
main.cpp (Reading input and handling commands)	10%
Readme, Comments and Makefile	5%
Total	100%

Compiling the Program (Same as assignment - 2):
A Makefile should compile your code into a program called "main". Your program should run with the following command syntax:

```
    ./main <input file name>
```

Commands to run and compile the code should be documented in the Readme file clearly.
Code that fails to compile or the code that compiles but fails to run will receive a grade of zero.

Late Submission Policy:
Except in the case of serious illness or emergencies, projects must be submitted before the specified deadline in order to receive full credit. Projects submitted late will be subject to the following penalties:

- If submitted 0–24 hours after the deadline 20% will be deducted from the project score
- If submitted 24–48 hours after the deadline 40% points will be deducted from the project score.
- If submitted more than 48 hours after the deadline a score of 0 will be given for the project.

Students unable to submit a project due to a serious illness or other emergencies should contact the instructor as soon as possible before a project's deadline. Based on the circumstances, the instructor will decide on an appropriate course of action.

Submission Notes:
You must include your full name and university email address in your Readme.txt file. If you are doing a project in a group of two, list the full names and email addresses of all two group members. If you are in a group you must also describe the contributions of each group member in the assignment.

Submit the following files on Odin:
Directory named- DoublyLinkedList containing

- DoublyLinkedList.h
- DoublyLinkedList.cpp

- main.cpp
- Makefile

One Readme file with all necessary descriptions for all directories.

Odin Login Instructions

This semester we are switching from Nike to a new server called Odin. Unlike with Nike, to use Odin, you must use the UGA VPN first. Please visit remote.uga.edu if you haven't already and install the VPN client for your OS. Be sure to have an SSH client like PuTTY or MobaXterm and use that to connect. The server's name is odin.cs.uga.edu and you will log in with your MyID credentials. For those that have used Nike before, the login process is almost the same, except that nike.cs.uga.edu is replaced by odin.cs.uga.edu, and, as stated before, you'll log in with your MyID credentials instead of your regular Nike credentials.

Group Submissions:

****Note: Assignment can be done individually or in a group of two students. The members of the group should be from the same section. Submit a readme file with your names, UGA emails and any other information that you want to give the TA to run your code. If you are doing it in a group of two make sure to include the contribution of each student in the readme file.**

If you are working in a group of two, your submission folder must be named as below:

```
LastName1_LastName2_assignment3
```

When you are submitting the folder you should use the submit command as below:

```
$ submit LastName1_LastName2_assignment3 <odin-submission-destination>
```

Make sure to replace <odin-submission-destination> with the correct destination from the table on the bottom of this document.

If you are working in a group, you must **submit the assignment from the Odin account of only one of the members** in the group.

Individual Submissions:

If you are working individually, your submission folder must be named as below:

```
LastName_assignment3
```

Submission command should be used as below:

```
$ submit LastName_assignment3 <odin-submission-destination>
```

Make sure to replace <odin-submission-destination> with the correct destination from the table on the bottom of this document.

****For all the submissions, the folder naming and the readme is very important in order to be able to locate your assignment and grade it.**

****Note: Make sure you submit your assignment to the right folder to avoid grading delays.**

When using the submit command, first set your current directory of the terminal to the immediate parent of the assignment folder.

You can check your current working directory by typing:

```
    $ pwd
```

For example if your assignment is in the following directory,

```
    /documents/csci2720/assignments/LastName_assignment3
```

Before executing the submit command, change to the assignment's immediate parent directory:

```
    $ cd /documents/csci2720/assignments/
```

Then execute the submit command:

```
    $ submit LastName_assignment3 <odin-submission-destination>
```

Make sure to replace <odin-submission-destination> with the correct destination from the table on the bottom of this document.

To check if your submission command was valid, there should be a receipt file.

```
    $ ls LastName_assignment3
    // should return rec##### like 'rec54321'
```

Selecting the Odin Submission Destination

You must select your Odin submission destination using the following mapping based on the section of the course you are registered to.

Section	Class Time	Odin Submission Destination
CRN 17504	M 10:20 AM	csci-2720a
CRN 42634	M 04:10 PM	csci-2720c