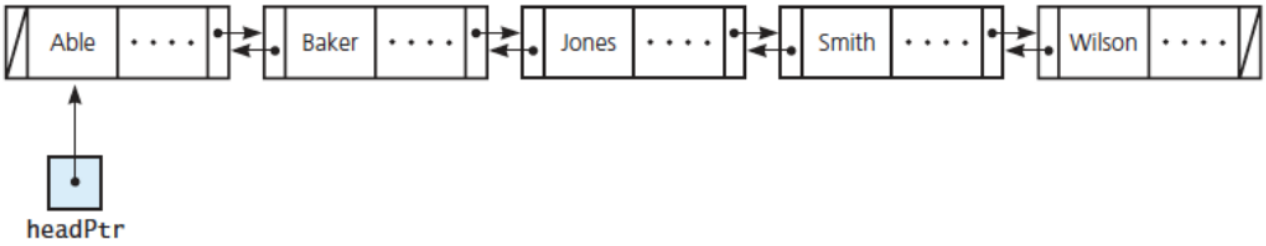
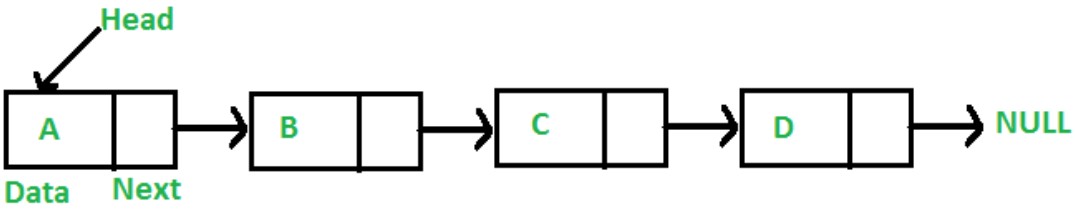
Project 3: Doubly↔Linked Lists



Background:

Your objective for this project is to implement a Doubly-Linked List. In order to successfully complete this project, you must understand the prerequisite material from the first and second projects (especially when it comes to the concept of templates), and you **must** understand the concept of a Linked List ADT.

Much like an array, a linked list is what is known as a linear data structure, however the elements are not stored at contiguous memory locations. The elements in a linked list are linked using pointers. The elements themselves consists of ***nodes*** where each node contains a data field and a reference (or link) to the next node in the list.



The above is an example of a singly linked list because the links or references go in one direction. Now while this is useful in some cases, much like the boy band, one direction just simply isn't good enough for us.

Project 2 will make you work with ***Templates and ADTs*** once again but now you will implement a doubly linked list. A doubly linked list is a slightly more complex type of linked list in which a node contains a pointer to the previous as well as the next node in the sequence. Therefore, in a doubly linked list, a node consists of three parts: node data, pointer to the next node in sequence (next pointer), and pointer to the previous node (previous pointer).

Implementation:

- ***You will also be given an hpp for `DoubleNode` and `DoublyLinkedList`, two template classes which you must implement (you will write the cpp yourself)*** based on the comments given in the hpp for each function.
- You will notice this project also contains a destructor. You shouldn't have to worry about it too much (until 335) aside from knowing it will call the clear() function. A destructor is invoked automatically whenever an object is going to be destroyed at the very end of the program.

Additional resources:

• ***Linked Lists:***
  ○ Geeks for Geeks
  ○ Tutorials Point
	• ***Template Classes:***
	  ○ CPP Manual
	  ○ Geeks for Geeks
	  ○ Tutorials Point
		• ***Deep vs. Shallow Copy***
		  ○ UTexas
		  ○ Geeks for Geeks

Task: Implementing the `DoublyLinkedList` functions!

Note all the functions you will need to implement and explanations for each are provided for you in the hpp so we won't go too in depth into what you need to do for each function.

IMPORTANT: SOMETHING TO CLEAR UP ABOUT POSITION:

Alright so we've always known as the first element being index 0. But note for this particular project there is no position 0, we start from position 1. So inserting at position 1 is inserting at the head of the list. Inserting 0 should be out of bounds in this case.

This is so counter intuitive, why would you do that?

Honestly, sometimes life just throws a curveball at you. Meaning I don't know but after being halfway through making this project there was no shot I was going to change it. But hey, just make sure to account for that in your functions!

What do you mean by "case":

An edge case is a certain scenario your program will run into in which you will need to account for. For example for insert I give you the list of basic cases you need to check for:

```
* Handles edge case of an invalid position parameter
* Case: Inserting into head of the list
* Case: Inserting into rear of the list
* Case: Inserting into a position that is not an extremity
```

This means you should have various if statements checking to see which position you are actually inserting in. If it's position 1 (the beginning of the list) then it will clearly be different than inserting an element in the middle of the list. Think about why. If we are inserting in the beginning of the list then our element is going to be the new head pointer. But if we are inserting in the middle then the previous will be an actual element. But if we are inserting at the end of the list then the next pointer will just point to *nullptr*.

The clear function and copy constructor:

The clear function won't just simply set the item count to zero unfortunately, just like how the copy constructor won't simply repoint to the parameter but will make something known as a ***deep*** copy.

Check out this medium article for more info about what a deep copy is in linked lists:
https://medium.com/spotthedifference/deep-copy-a-linked-list-b90d8376223f

Likewise, your clear function will need to go through each node and delete the pointer and set it to *nullptr*, we can't have memory leaks going on!

Testing

How to compile:

```
g++ <test main file> -std=c++17
```

You must always implement and test your programs **INCREMENTALLY!!!** What does this mean? Implement and test one method at a time.

- Implement one function/method and test it thoroughly (multiple test cases + edge cases if applicable).
- Implement the next function/method and test in the same fashion. **How do you do this?** Write your own `main()` function to test your class. In this course you will never submit your test program, but you must always write one to test your classes. Choose the order in which you implement your methods so that you can test incrementally: i.e. implement mutator functions before accessor functions. Sometimes functions depend on one another. If you need to use a function you have not yet implemented, you can use stubs: a dummy implementation that always returns a single value for testing. Don't forget to go back and implement the stub!!! If you put the word STUB in a comment, some editors will make it more visible.

Grading Rubric

Correctness 80% (distributed across unit testing of your submission) **Documentation 10% Style and Design 10%** (proper naming, modularity, and organization)

Important: You must start working on the projects as soon as they are assigned to detect any problems with submitting your code and to address them with us **well before** the deadline so that we have time to get back to you **before** the deadline. This means that you must submit and resubmit your project code **early** and **often** in order to resolve any issues that might come up **before** the project deadline.

There will be no negotiation about project grades after the submission deadline.

Submission:

You will submit the following file(s)**:** `DoubleNode.cpp` `DoublyLinkedList.cpp`

Your project must be submitted on Gradescope. Although Gradescope allows multiple submissions, it is not a platform for testing and/or debugging and it should not be used for that. You **MUST** test and debug your program locally. Before submitting to Gradescope you **MUST** ensure that your program compiles (with g++) and runs correctly on the Linux machines in the labs at Hunter (see detailed instructions on how to upload, compile and run your files in the "Programming Rules" document). That is your baseline, if it runs correctly there it will run correctly on Gradescope, and if it does not, you will have the necessary feedback (compiler error messages, debugger or program output) to guide you in debugging, which you don't have through Gradescope. "But it ran on my machine!" is not a valid argument for a submission that does not compile. Once you have done all the above you submit it to Gradescope.