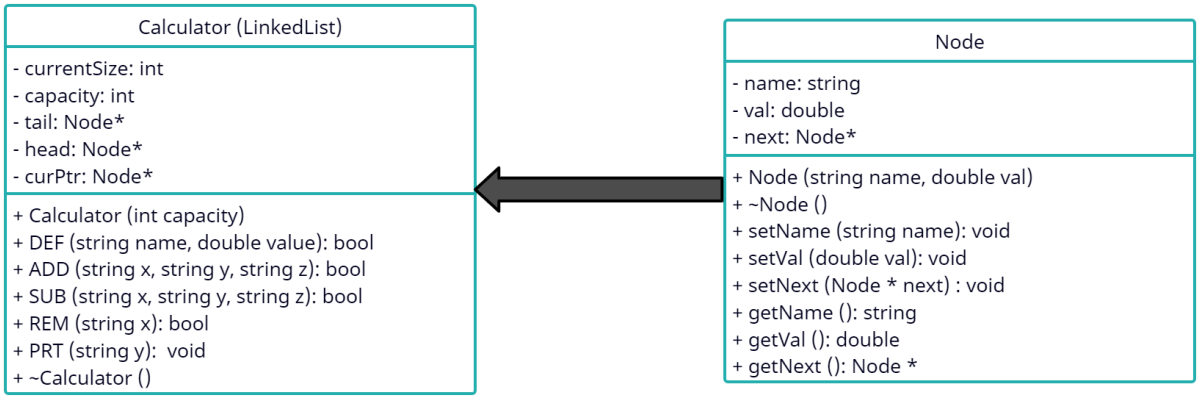
Jefferson Fu Project 1 Design Document
UML Diagram:



Node:

My node class included three different private member variables, a string "name" to identify the node, a double "value" for the node to have, and node pointer "next" to point to the next node within the linked list. The rest of the members of the class are public functions for a constructor, destructor, getters, and setters.

- Constructor: takes a string "name" and double "value" to identify each node uniquely, I decided to leave out the option of defining the next pointer value because it will always be nullptr until a new node is created.
- Destructor: The destructor is blank because all nodes will be deleted during the destructor of the linked list and no left over values will exist.
- Getters & Setters: The following getters are getName, getVal, getNext, their return types correspond to their variable types. The following setters are setName, setVal, setNext, they are all void functions because nothing is required to be returned when you are setting them. Instead, they take their corresponding variable as a parameter.

Calculator:

My calculator class implemented a linked list using nodes defined by the class above. I decided to have five private member variables: an integer "currentSize" to keep track of the size of the linked list, an integer capacity to keep track of the max size of the linked list given in input, and three node pointers for head, tail and current pointer of the linkedlist. Head to know where to start when I iterate through it, a tail to know where to append to when a new node is inserted and a current pointer as an iterative pointer to avoid redefining it in a function every time I traverse through the linked list.

I chose most of the public member functions of the class to be boolean because that way I can get the result as true, printing success, and false, printing failure within the test file. The only exceptions are the void PRT function where the result is printed within the calculator class because of its unique output, constructor and destructor.

These are the following member functions:

- Constructor: This is called when the CRT command is called. I take one integer “capacity” as a parameter to set the capacity of the linked list. There are no other parameters needed as there is nothing else the user can control, size of the linked list will always be zero and the pointers will all point to null because there are no nodes yet.
- DEF: takes a string “name” and a double “value” as a parameter to understand the correct data to create the node. Within the function, we first check to see if we can create the node which involves checking if we are not at capacity and then seeing if the node name already exists within the list. The function returns false(failure) if any of those checks fail. If everything is good to go, we can increment the size of the linked list and create a new node with the new data. If this is the first node of the linked list, we set it to become the head, otherwise we change the current tail’s next to point to this node and then change the tail to the current node.
- ADD/SUB: My ADD and SUB functions are extremely similar. In both functions I define three different node pointers xPointer, yPointer, and zPointer and set their default values to nullptr. Upon traversing the linked list, I make three different if statements to see if the name matches with x, y, or z which are the node names we are trying to find. If they match, I set the respective pointer to become the node that the iterative pointer is currently at. Within this step, I noted to not use else if or else because if any of the nodes we are searching for are the same, they will be skipped using else or else if. After traversing the list, I check to see if any of the pointers still point to nullptr, as if any of them are, I will return false and print failure. If none of them point to nullptr, then I know all of them are found and I add or subtract the xPointer value and yPointer value to get a sum or difference. Lastly, I set the zPointer value to the sum or difference and return true to let my program know that the function was a success.
- REM: My REM function only takes one parameter which is the name of the node I am trying to find. The function traverses the list while keeping track of a newly created prev pointer pointing at the previous node we just looked at. If I find the node we are looking for in my iteration, I change the previous node’s next node to be the current pointer’s next node and then delete the current node. I also make sure to update the tail and head if those are the nodes being deleted. At the end of that, I decrement the current size of the linked list and return true as I successfully deleted a node. If I did not find the node that I wanted to find, I would return false.
- PRT: Similar to the REM function, I take the name of the wanted node as the only parameter. I then traverse through the whole linked list to search for the node and if I find it, I’ll print the node’s value, and if I don’t I’ll print “variable ‘x’ not found”. Variable x being the node I am looking for.
- Destructor: In order to make sure everything is deleted, I traversed through the whole linked list and deleted each value. I did this by making a temporary variable keeping the head’s next node, deleting the current head node, and then moving the head node to the temporary value.

It's also worth noting that every function except for creating the list is O(N) time because each function requires traversing through the list to see if a node(s) exists. Creating a list is O(1) time because we just set the linked list to the given capacity.

Other Decisions:

- I did not include a search function because each time I traversed through the list, I was looking for different parameters so I did not see how one search function would satisfy my needs for each different function.
- I took care of the END case in my test file as I can delete the calculator object and that will call it's destructor.