Overview of Classes

class Node: a single node that will make up a linked list if connected with other node(s)

public: (all members are public as they need to be called by functions in class LinkedList)
there is no constructor/destructor as the default constructor/destructor are enough

string name : has name variable of the node
double val{} : has value of the node (both name and val will be stored in one node)
Node* next = nullptr : pointer to point the next node (points nullptr when created, until next node specified)

class LinkedList: singly linked list

parameter: string- name, x, y, z (as they are not limited to only numbers; also can be characters)
double- val (written in the project instruction)
return type: mostly used bool as I need success/failure returns; used void for prt to directly print from calculator.cpp file. I thought it would be better because I can directly print the getVal() function, which makes my code shorter.

private: those members are private as they will be only used in the public members functions
helper functions: two helper functions in addition to required functions: getNode() and getVal() to avoid repeating the same code multiple times.

int capacity: maximum value of number of nodes of a linked link
int length: number of current nodes
Node* head: the first node of the linked list
Node* getNode(string name): Search the node whose name is same as input value

1. curr goes through the list; while loop runs until 1) the list ends (*wrote curr->next != nullptr as we will need to go to the next node, and curr->next will be nullptr if it was curr != nullptr*) OR 2) find the node
2. if 1), return nullptr; if 2), return the pointer of node found

double getVal(string name): get the value in a specified node

1. get node by getNode() so that we can get value from it
2. loop while curr reach the node found
3. value is at address which is from curr->val
4. return the value

public:

LinkedList (int capacity): constructor- head is the very first node (*chose this way instead of assigning head to point the first node as it is easier to track nodes for me*)
capacity will be specified by cin; length is zero at first before CRT is ran
~LinkedList(): destructor- deletes every nodes when ending the program (referred to ECE 150 notes)

1. check if it is an empty list. if not, repeat 2-3 below until reaching the end of the list
2. make another pointer, curr, for the same node as current head, and move head to the next node
3. delete the curr node (which would be the previous node for head)

bool def(string name, double val): define name and value for the node at the end of the list

1. return false if name already exists or list is already full
2. if not, add a new node and put it at the end of the list
3. assign name, val and next to the new node; increase the list size by 1

bool add(string x, string y, string z): input 3 names, third name's val = first name's val + second name's val

1. return false if any of the 3 names do not exist
2. else, get values for first and second names using getVal() and add
3. search for the node with the third input name and reach it by using while loop
4. change the value for the third node to the first val + second val value

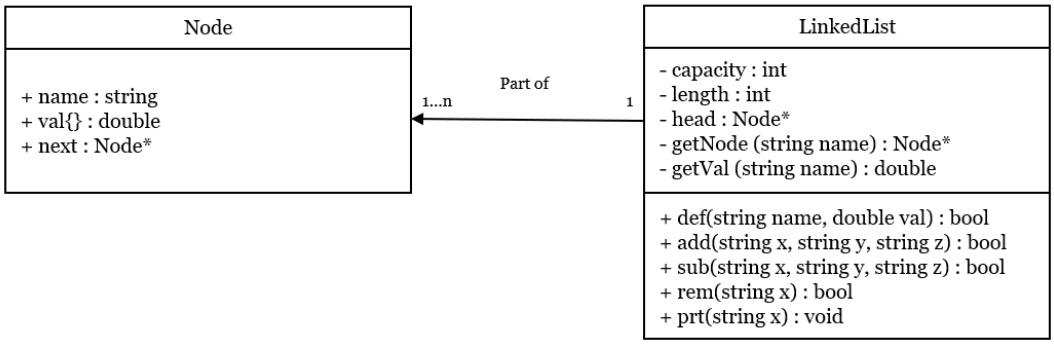
bool sub(string x, string y, string z): same as add() function except that this is first val - second val instead of plus
bool rem(string x): remove a node with name matches with input

1. find the node with the name same as input; return false if not exist
2. if exist, reach to the node using while loop (same reason as Node* getNode() for using curr->next!=getNode instead of curr)
3. create a new temp pointer, temp, which also points to the Node with input name
4. assign curr's (1 node before temp) next is temp's next node
5. delete temp node, and decrease the list size by 1

void prt(string x): print value of input name

1. find the node with the input name using getNode(); if not found, print variable [name] not found
2. if node found, get its value using getVal(), and print the value

UML Diagram



Efficiency

Command	Function	Time	Explanation
CRT	LinkedList(int n_capacity)	O(1)	Creating a linked list (insertion) does not depend on the number of nodes in the list as it just needs to have head, capacity and length set.
DEF	def(string name, double val)	O(n)	def uses getNode() while checking if x already exists. It also goes to the end of the list from the head to def the new node at the end of the list. Therefore, the time depends on the number of nodes in the list.
ADD	add(string x, string y, string z)		add uses getNode() while checking if any of the three inputs do not exist. It also uses getVal() and getNode() when getting nodes/values for the 3 inputs if they exist. Also, if the z (the last input) is at the end of the list, it will be the worst case. And all of these depend on the number of nodes.
SUB	sub(string x, string y, string z)		sub works the same as add except for changing plus to minus.
REM	rem(string x)		rem uses getNode() to check if there is such a node with the name [input] and to get to that node. If the node is at the end of the list or does not exist, it needs to go next until the last node is reached. So, the time depends on the number of nodes.
PRT	prt(string x)		prt uses getNode() and getVal() functions to check if node of [input] exists and to get the value of the node. The time depends on the number of nodes as going through 1 node and many numbers of nodes will take different times.
END	~LinkedList()		As all the nodes need to be deleted, the time fully depends on the number of nodes in the list. It needs to go one by one and delete them.
.	getNode(string name)		Node* curr needs to go through the whole list if the node it's looking for is at the end, which is the upper bound. So, time depends on the number of nodes as it will take more time if there are more nodes.
.	getVal(string name)		To get a value of a node, we need to get the node first; we use the getNode() function to find the node. There is also a while loop to get to the node. So, the time is dependent on the number of nodes.