Design Document
Jonathan Lam

I designed a List class for the linked list and a Node class for each node within the linked list. The linked list implementation for this design is a singly linked list.
jlc2lam_p1.hpp is the header file containing the class declarations.
jlc2lam_p1_helper.cpp contains the class definitions.
main.cpp contains helper functions for printing outputs and main().

The member variables for the Node class are:

- std::string nodename{}, double value{}; nodename and value stores the name of the node as a string and its value as a double respectively as required in the project.
- Node *p_next; p_next is the pointer to the next node for the linked list implementation.

Member variables are private to avoid unintentional assignment to variables within main().
The member functions for the Node class are:

- Node(double val, std::string name); The constructor takes in the value and name of the node that it needs to store as required in the project. The constructor initializes p_next to nullptr and variable value and name with the value and name that it needs to store that gets passed through the parameter respectively.
- ~Node(); The destructor is just the default destructor. Memory deallocation is done in the destructor of the class List.
- Node *getnext(); Returns the private member Node pointer p_next for the pointer to the next Node. Has a runtime of O(1).
- void setnext(Node *p_new_next); Takes in a Node pointer p_new_next in the parameter and assigns pointer p_new_next to p_next. Does not need to return anything. Has a runtime of O(1).
- std::string getnodename(); Returns the private member string variable nodename. nodename does not need to be modified for this project and hence no setnodename() function. Has a runtime of O(1).
- double getvalue(); Returns the private member double variable value. Has a runtime of O(1).
- void setvalue(double val); Takes in a double val in the parameter and assigns val to value. Does not need to return anything. Has a runtime of O(1).

The member variables for the List class are:

- Node *head; Node pointer head that points to the head of the list.
- unsigned int curNode; unsigned int curNode to keep track of the number of Nodes in the linked list.
- unsigned int maxNode; unsigned int maxNode holds the maximum number of nodes allowed for the linked list.

Member variables are private to avoid unintentional assignment to variables within main().
The member functions for List class are:

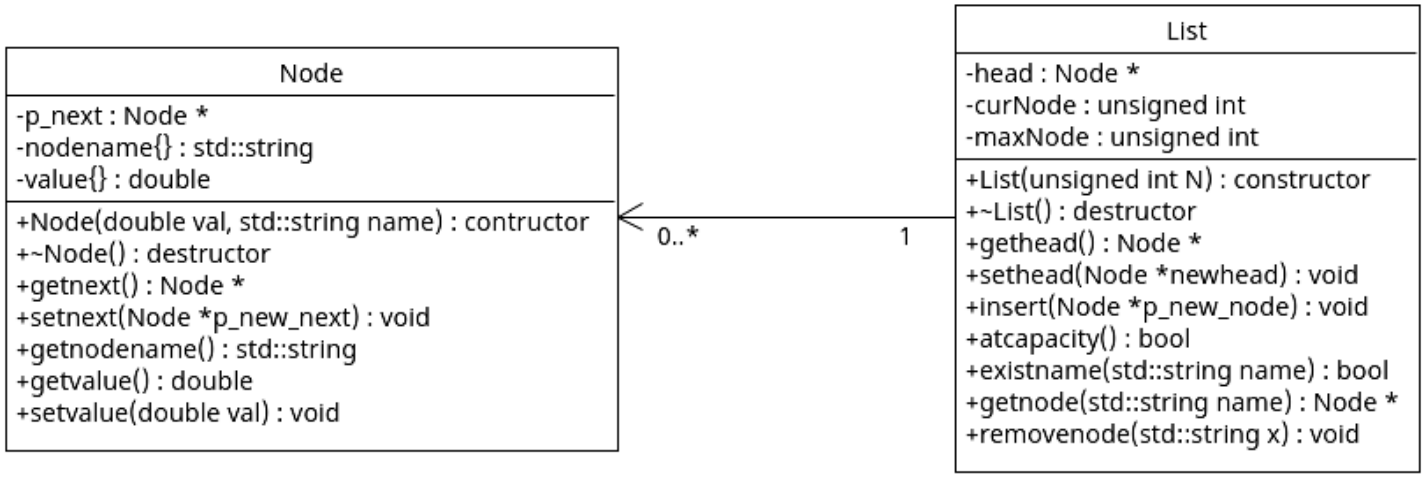
- List(unsigned int N); The constructor takes in unsigned int N that indicates the maximum number of Nodes the list can hold and initializes maxNode with N. Node pointer head is initialized with nullptr. CurNode is initialized with 0 since the linked list is empty at the start.
- ~List(); The destructor creates a temporary pointer and a temporary next pointer and both traverses through the linked list from head pointer to deallocate every Node.
- Node *gethead(); Returns the private member Node pointer head for the head of the linked list. Has a runtime of O(1).
- void sethead(Node *newhead); Takes in a Node pointer newhead in the parameter and assigns pointer newhead to head. Does not need to return anything. Has a runtime of O(1).
- void insert(Node *p_new_node); Inserts a Node to the list. If head is nullptr, it indicates the linked list is empty and head is assigned with p_new_node. Else if curNode is less than maxNode, a temporary pointer traverses the list until getnext() returns nullptr and sets its p_next with p_new_node. Both increments curNode. Else, deallocate p_new_node as it does not fit the conditions for inserting into the linked list. Does not need to return anything. Has a runtime of O(n) since it goes through the linked list once.
- bool atcapacity(); Returns true if linked list is at capacity (curNode == maxNode). Otherwise, returns false. Has a runtime of O(1).

- bool existname(std::string name); Temporary pointer traverses through the linked list and gets each Node's name to check if it matches the string passed in the parameter. Returns true if there is a match, returns false if linked list is empty or there is no match. Has a runtime of O(n) since it goes through the linked list once.
- Node *getnode(std::string name); Takes in a string and traverses with a temporary pointer to find a node with the matching string name. Returns the pointer pointing to the Node before the Node that matches the name if there is a match. This is so that when removing Nodes, for example: remove Node B in linked list A->B->C, B is deallocated using getnode(std::string name)->getnext() and getnode(std::string name) returns pointer to Node A to directly set next pointer to point to C. If the first Node has the matching name, the function returns nullptr. existname function is always used before getnode function in main() to guarantee there will be a match. Has a runtime of O(n) since it goes through the linked list once.
- void removenode(std::string x); Takes in string x, the name of the Node that should be removed. If getnode returns nullptr, meaning the first Node needs to be removed, a temporary pointer is assigned to head, head is set to the next pointer of the head, and head is deallocated and curNode is decremented. Otherwise, initialize and assign a temporary pointer to getnode->getnext Node to hold the Node to be removed, set next pointer of getnode to the temporary pointer's next pointer, and deallocate temporary pointer and decrement curNode. Has a runtime of O(n) since it goes through the linked list once.

Within main(), CRT command has a runtime of constant time O(1) since it creates the list instantaneously. All other commands have implementations with List member functions that have a runtime of O(n).

- CRT command; Creates a new linked list with N maximum Nodes.
- DEF command; Creates new Node with value and name and inserts into the linked list if the list is not at capacity and Node name does not already exist in the list.
- ADD command; Check if all input names exist in the linked list. Then each of x, y, and z has 2 cases: if it is head of the list or not, and assigns x_value and y_value and adds them together to be assigned to z accordingly.
- SUB command; Similar implementation as ADD command. Instead of adding it subtracts.
- REM command; Check if Node name exists in the list, then remove the Node with the matching Node name.
- PRT command; Check if Node name exists in the list, then get the value corresponding to the Node name with 2 cases similar to ADD and SUB commands, if the Node is at the head of the list or not, and execute the command accordingly. Prints with the Node name if the name does not exist in the list.

UML Diagram:



Test Cases:
Tested normal operations of CRT, DEF, REM, ADD, SUB, PRT with list lengths 3 and 10 to make sure of no memory leakage and correct outputs. Edge cases tested were with list length 1 and 2 for the different conditions needed for Nodes at head and ADD/SUB commands where repeated Nodes in the parameters may fail. Test REM and PRT Nodes that do not exist, test ADD/SUB Nodes that do not exist and ADD/SUB Nodes to a Node that does not exist, DEF Nodes more than maximum Nodes allowed and Nodes that already has existing name in the list, test ADD/SUB for the same Node added with itself and stored to the same location, test REM at head, and REM and DEF commands in conjunction for to the limit of the range of the linked list for proper curNode tracking.