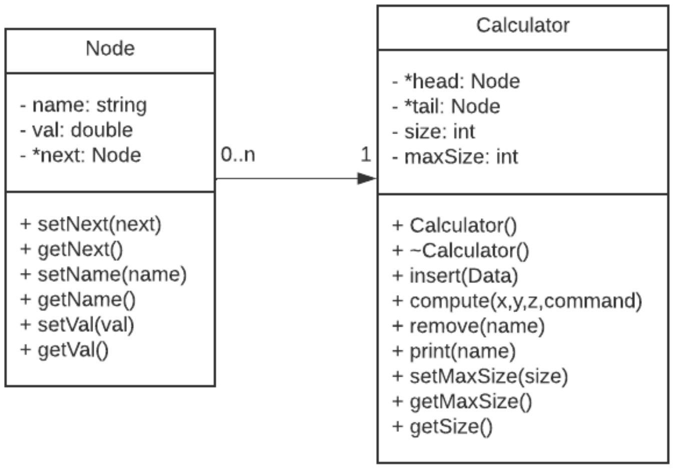
ECE 250 Project 1 Design Document

Brandon Vo

January 27, 2023



Node Class

In the node class, there are three private member variables. 1. "name": a string variable to hold the name of the node. 2. "val", a double to store the value of the node. 3. "next": a pointer to refer to the next node in the linked list. This does not need a destructor, since it does not hold any resources that need to be freed. These nodes will be deleted in the Calculator's destructor. This class also does not need a constructor, since it does not need to initialize its data members.

Setter/Getter Functions

All setters are void functions, as they are not returning a value. They are only setting the value of the member variables in the node. Each of these takes in a value to be set. The getter functions are all assigned to their respective type to return to the program. Getter functions are all constant since they are not modifying any values.

Setters	setNext: Node object parameter for next node in linked list	setName: string parameter to set the name of the node	setVal: double parameter to set the value of the node
Getters	getNext: return Node	getName: return string	getVal: return double

Calculator Class

The calculator class is an implementation of a linked list with all major functionalities for the simple calculator to work. The private member variables are a node pointer to the head and tail of the linked list, and two integer variables to keep track of the current size (size) and maximum capacity (maxSize) of the linked list. In this class, I have implemented a constructor, destructor, insert, compute, remove, print, setMaxSize, getMaxSize, and getSize functions.

Constructor/Destructor

The linked list constructor initializes the head and tail to nullptr. The size and maxSize variables are set to 0. This is done for the program to create an empty list. Additionally, a tail variable is used so that we can insert nodes to the end of the linked list in another function more efficiently. The destructor iterates through the entire linked list by using a while loop with a variable pointer. In this loop, it stores the next node to a variable, deletes the current node the pointer is at, and assigns that variable pointer to the next node. Once we reach the end of the linked list, all nodes will be deleted, so the program can deallocate memory after the Calculator object is destroyed.

Insert
The insert function is used to add nodes to the end of the linked list when using the DEF command. This is a void function since it is not returning a value. The parameters needed for this function are a string for the name of the node, and a double for the value of the node. This function iterates through the linked list using getNext() and checks if a node with the same name exists already with getName(). If it does, it prints out "failure". Otherwise, a new node will be created with the input parameters, then inserted at either the head and tail if the linked list is currently empty, or the end of the linked list using the tail variable if nodes already exist in the list.

Compute
The compute function handles the ADD and SUB commands for the calculator. It is a void function since it is not returning a value. The parameters for this function are three strings x, y, and z for the node names, and a string that takes in the command used (ADD or SUB) to determine which calculation to do. The three nodes are defined by using the getName() property for a node and comparing it to the input names. Without the command parameter, there would need to be two separate functions for adding and subtracting nodes. This would be bad practice since both ADD and SUB require you to do the same process for defining nodes x, y, and z. By reading the command used as a parameter, we avoid repetitive code by reading which computation needs to be executed. After the nodes x, y, and z are defined, it will check if all of these nodes exist. If so, the computation will be successfully executed by using setVal() and getVal() to compute $x \pm y = z$ and "success" will be printed. Otherwise, "failure" will print meaning that at least one of the nodes x, y or z has not been defined.

Remove
The remove function searches through the linked list and deletes the node with the input parameter name using getName(). It does not return any value, so it is a void function. The parameter taken in is a string for the name of the node that wants to be deleted. An edge case we need to handle is if the last node of the linked list is being deleted. If so, the Calculator tail pointer must be updated to the current pointer before it is removed. For checking which node needs to be removed, a current pointer uses the getName() property to compare to the input name. If the node has been found, it will be deleted, and a previous pointer will move to the current's next pointer using getNext(). This is skipping the node that is being deleted and relinking the list together. After deleting the node, the function will print out "success". If it does not find the node we want to delete, it prints out "failure".

Print
The print function is a void type, as it is only trying to print out a value. The parameter is a string for the name of the node it is trying to print. It searches through the linked list once and looks for a node with the same name using getName(). If found, it will print out the value of the node with getVal(). Otherwise, if the node with the given name has not been found after traversing, it will print "variable [name] not found".

Setters/Getters Functions

Setters	setMaxSize: integer parameter	
Getters	getMaxSize: return max capacity of linked list	getSize: return the current size of linked list

Overall Runtime
Every command but "CRT" and "END" has a tight bound of O(n). In the "DEF", "ADD", "SUB", "REM" and "PRT" commands, they all iterate through the entire linked list once at worst. These commands use a while loop which iterates while a current pointer does not equal nullptr. In each pass, the current pointer is moved to the next pointer. This is done until we reach the end of the linked list, or a needed value has been found and executed. The "CRT" and "END" commands have a linear O(1) runtime since they are independent of the size of the input. They do not need to loop through anything, and their commands are executed immediately.