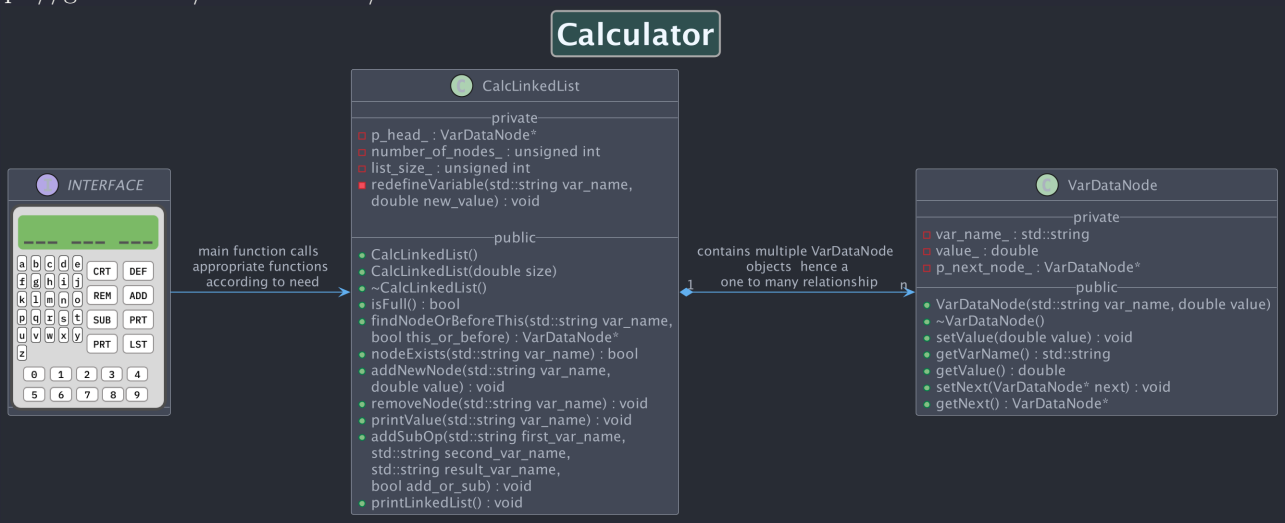
Abstract

This is the design document of Chaitanya Sharma for Project 1 for ECE 250's Winter 2023 offering.

1 Class Structure

My program consisted of two major classes, namely VarDataNode and CalcLinkedList and two independant functions, one of which is obviously the main function, and the other being a type bool function printStatus function. I've tested all edge test cases possible and also used the test cases provided by the jekelautograder on https://github.com/JZJisawesome/ece250-testcases



1.1 Independant Functions

1.1.1 void printStatus(bool status)

In this design document I will be going according to the order of my code as well as sometimes the importance, hence since we're starting with the main file, the printStatus function is a function which I created to reduce the cluttering of my code due to repetitive usage of std::cout«"success"«std::endl; and the opposite, and it runs in O(1) since console output is one clock cycle.

1.1.2 int main()

The main function is the entry point of my program, and it uses the parsing structure which I borrowed and understood from what our Lab Instructor showed in the vide HowToParseInGeneral on LEARN.

It runs in O(n), as it is a while loop which runs until there are inputs, and then the input is disected into command conditions like CRT, DEF, etc. Then, if the conditions provided to us in the table of the project descriptions are true, and the program succeeds in accomplishing the task given, it uses the printStatus to provide the output of success, and vica versa. But since it runs the program itself, we discard its runtime costs for calculation and theory of runtime purposes, as it is the base of the program itself.

1.2 class VarDataNode

This class is the class which is used to create the nodes of the Linked List, hence it needs to store all the data needed in a node, according to use case, which in our case is the variable name(as a string), and the value of the variable(as a double), and a pointer, since this is supposed to be private information, they're designated privately as var__name__, value__ and p__next__node__ respectively. Also, they have their own setters and getters(all in O(1)), but in the case of the variable name, there's no setter since I didn't find any use case for it.

Though we're not asked to redefine a variable's value anywhere explicitly, i n the design document, it is given in context of addition and subtract operations, that the result variable needs to exist already, meaning that the setter for the variable value is needed. All other setters and getters are necessary for the functioning of the linked list.

1.2.1 CONSTRUCTOR AND DESTRUCTOR

As there's no use of a no-value constructor, but I have included a value-init constructor, which initializes appropriate values (in O(1)). I have used C++'s internal feature of a Default Destructor, by equating the destructor declaration to default, since this a very simple and and independant class which does not allocate anythin on the heap itself.

1.3 class CalcLinkedList

On the subject of head__ pointer, it is the most crucial part of a linked list, also, the list__size__ which is required for initializing the list with a defined size is also kept private and due to no need of modifying/accessing them outside of this class, there's no setters/getters.

I added the number_of_nodes_ private variable to keep track of the number of nodes in the list, as it is a given requirement to not let the user define more nodes than the original size specified, and at some advanced level, it would be more computationally sound to keep track rather than calculate it.

I also added a redefineVariable private function, which is used only in the add and subtract functions as they assign the result value to an already-defined variable.

1.3.1 CONSTRUCTOR

I've added a no-value constructor, which initializes the head_ and list_size_ to nullptr and 0 respectively, and the number_of_nodes_ to 0. A value-init constructor, which initializes the head_ to nullptr, list_size_ to the size given, and number_of_nodes_ to 0. "CRT"

1.3.2 DESTRUCTOR

Since memory leak is a major grading ground of this project, hence even though small, but this was one of the most focus-requiring portion of my code, I create a temp pointer which starts at head and deletes it and moves on to another node until it sees a nullptr. "END"

1.3.3 bool isFull()

This function just returns the result of the comparison of the number_of_nodes_ and list_size_ through the equality operator.

1.3.4 VarDataNode* findNodeOrBeforeThis (std::string var_name, bool true_is_this_false_is_before_this) ★ a special feature of my code ★

A problem was running while loop in many of my functions, an iterative finder function would be not intuitive since removeNode would still traverse and the other three function printValue, addSubOp and nodeExists wouldn't, hence it was hard, BUT I COULD make a function which found the node before the node to be found, according to a condition, where the removeNode function would directly get the previous node to bypass the node to be deleted using a bool parameter false, while the other functions use true to get the pointer to that exact node. I take in the second parameter to decide whether the function is to return the pointer for node for variable name given, or the pointer for the node before the node for variable name given. In this function, I am basically exploiting the fact that O(n+1) is still == O(n) as it uses a traversal loop to find the node.

1.3.5 bool nodeExists (std::string var_name)

This function just returns the result of the comparison of the result from findNodeBeforeThis with the arguments of the variable requested and true

1.3.6 void addNewNode (std::string var_name, double value)

This function first allocates a new VarDataNode on the heap, and then sets its next pointer to the head pointer, and reassigns the head_ pointer to the new node, and then increments the number_of_nodes_ by 1. "DEF"

1.3.7 void removeNode(std::string var_name)

For the head node, I just set the head_ pointer to the next node, and for all else, I get the pointer to the node before the node to be deleted, and then sets its next pointer to the next node of the node to be deleted, and then finally deletes the node at the address from temp pointer. The runtime of this function is O(n) since it uses the findNodeOrBeforeThis function for traversal. "REM"

1.3.8 void printValue(std::string var_name)

This function uses the findNodeOrBeforeThis function to get the pointer to the node with the variable name given, and then uses std::cout to print the value of that node by using getValue function. "PRT"

1.3.9 void addSubOp (std::string first_var_name, std::string second_var_name, std::string result_var_name), bool add_or_sub

This function uses the findNodeOrBeforeThis with function to get the value of the operands, then according to the boolean parameter, it adds or subtracts the two values if true or false respectively and then assigns the result. The runtime of this function is O(n) since there are only sequential while loops and not nested ones and that too due to calling findNodeOrBeforeThis function. "ADD" AND "SUB"

1.3.10 void printLinkedList() CONVENIENCE FUNCTION: beautifully prints the linked list

"LST"