ECE250 Project 1: A Simple Calculator

Due Date: Friday, January 27, 2023 @11pm. Submit to the dropbox on Learn

Overview

The goal of this project is to implement a simple calculator that can add and subtract variables stored in a linked list. You must create your own linked list class for this project, using proper object-oriented design principles. **You are not permitted to use any classes from the STL C++ library in this project.**

Understanding the calculator

You will be given script files that your code must parse. The script files will be comprised of statements, one per line, that hold either variable declarations, commands, or desired computations. Variables are comprised of a name string that follows C++ variable naming conventions and a value, which you may store as a double.

Your code must keep track of the variables that it encounters in a linked list, which grows and shrinks dynamically as variables are added or removed. Then, when variables are used in arithmetic expressions, your code must search the linked list for the variable and perform the appropriate computation, or it should indicate that the variable has not been declared. To understand the various expressions, see Table 1 below.

Program Design and Documentation

You must use proper Object-Oriented design principles to implement your solution. You will create a design using classes which have appropriately private data members and appropriately public services (member functions). It is not acceptable to simply make all data members public. **Structs are not permitted.**

Write a short description of your design to be submitted along with your C++ solution files for marking according to the template posted to Learn.

Input/Output Requirements

You must create a test cpp file that contains your main function. This program must read commands from standard input and write to standard output. The program must respond to the commands shown in Table1 below. The outputs listed in the "Output" column must appear as shown. For instance, the first row has "success" as an output. This means that the string "success", written in lowercase, is expected, followed by a newline.

Table 1: Testing Commands

Command	Parameters	Description	Output
CRT	N the maximum number	Create a new linked list to store variables. This linked list should be able to contain a maximum of N variable names. Note that N is to prevent the list from hitting hardware constraints. You may assume CRT starts every test file and that N can be read as a positive integer. You may assume that CRT appears only once per input file.	**success**
DEF	name val	Define a new variable with name "name". This variable must store value val. You may assume that name is always a valid C++ variable name, but you must verify that the variable does not already exist in the list. You may assume that val can be read as a double.	**success** if the linked list is not yet at capacity **failure** if the linked list is at capacity or the variable exists

Command	**Parameters**	**Description**	**Output**
ADD	x y z	Add the values stored in variable names x and y and then store the result in the variable name z. This is equivalent to the following C++ statement: z = x+y; Note that duplicate names are allowed in this command	**success** if all three variables exist **failure** if any of the three variables does not exist
SUB	x y z	Subtract the values stored in variable names x and y and then store the result in the variable name z. This is equivalent to the following C++ statement: z = x – y; Note that duplicate names are allowed in this command.	**success** if all three variables exist **failure** if any of the three variables does not exist
REM	x	Remove the variable name x from the list	**success** if the variable exists and has been removed **failure** if the variable does not exist in the list
PRT	x	Prints the value of the variable name x	**If the variable name (x) exists:** prints the numerical value stored in the variable x followed by a newline **If the variable name (x) does not exist, print this followed by a new line:** variable x not found
END		All input files finish with end	This command does not print any output. Once it is encountered your program should end

The expected runtime of all operations above is O(n), where n is the number of variable names in the list, except for the "CRT" command, which is O(1). In your design document, you must describe how you have achieved these runtimes in your implementation.

Valgrind and Memory Leaks

5% of the grade of this project will be allocated to memory leaks. We will be using the Valgrind utility to do this check. The expected behaviour of Valgrind is to indicate 0 errors and 0 leaks possible, with all allocated bytes freed. To test your code with Valgrind, presuming you are using an input file test01.in, you would use the following command:

```
valgrind ./a.out < test01.in
```

Test Files

Learn contains some sample input files with the corresponding output files. The files are named test01.in, test02.in and so on with their corresponding output files named test01.out etc.

Submitting your Program

Once you have completed your solution and tested it comprehensively on your own computer or the lab computers, you must transfer your files to the eceUbuntu server and test there. **A makefile is required for this project since the exact source structure you use will be unique to you.** We perform automated testing on this platform, so if your code works on your own computer but not on eceUbuntu it will be considered incorrect.

Once you are done your testing you must create a compressed file in the tar.gz format, that contains only the following:

- A typed document, **maximum of two pages**, describing your design. Submit this document in PDF format. The name of this file should be xxxxxxxx_design_pn.pdf where xxxxxxxx is your maximum 8-character UW user ID (for example, I would use my ID “mstachow”, not my ID “mstachowsky”, even though both are valid UW IDs), and n is the project number. In my case, my file would be mstachow_design_p1.pdf.
- A test program containing your main() function.
- Required header files that you created.
- Any additional cpp files that you created.
- A makefile, named Makefile, with commands to compile your solution and create an executable. This makefile will be run using the “make” utility. Do not specify an output file name, the default of a.out is used in the automated testing.

The name of your compressed file should be xxxxxxxx_pn.tar.gz, where xxxxxxxx is your UW ID as above and n is the project number.