COMP-1400
Assignment #2

OBJECTIVE:

Through three assignments, we will learn how to apply the concepts learned in this course to develop a simple command-line calculator using the programming language C.

In the second assignment, we are going to continue from and complete the calculator program we developed in the first assignment. This version will cover user inputs, simple operations, and displaying the results on the screen.

Part A [60 marks]: Completing the Calculator Functionality

In this part we complete the calculator program from Assignment 1. Refer to the example input/output format provided in Assignment 1.

Complete the following functionality:

1. Display the full list of available options to the user (B, U, A, V, E) as previously shown in the main menu from Assignment 1. The program should read the user's option (as a character B/U/A/V/E) and act accordingly. It is invalid to use a number to select an option.
2. Complete functionality for option B (binary operations). The program should read the first floating point number, an operation, and a second floating point number in that order. Valid binary operators could be + for addition, - for subtraction, * for multiplication, / for division, % for the remainder (floating point remainder, see math.h library), P for power, X for maximum, and I for the minimum. The operator is inserted as a character. It is invalid to use a number to select an operator.
3. Complete functionality for option U (unary operations). The program should read an operation followed by a floating point number, again in that order. Valid unary operators could be S for square root ($\sqrt{x}$), L for logarithm base e ($\log x$, or lnx), E for exponentiation (e^x), C for the smallest integer value greater than or equal to x (ceiling(x)), or F for the largest integer value less than or equal to x (floor(x)). It is invalid to use a number to select an operator.
4. Complete functionality for option V (variable memory). The program contains 5 memory slots, 'a', 'b', 'c', 'd', and 'e'. These memory slots can hold values given by the user (any floating point number) and their default state is 0.00. The program should allow functionality to set these memory slots for use in Option A. The program should first read the memory slot (as a character a/b/c/d/e) followed by the floating point value to save in the selected slot. It is invalid to use a number to select an operator.
5. Complete functionality for option A (advanced operations). A submenu must be displayed and present options B/U/E where B/U are the same as before with the added usage of variables, and option E allows the user to exit back to the main menu. With variables, the user gains the ability to perform the same operations as in option B and U with the added benefit of using variable names in their calculations where the program will use the saved values (from option V) in their place. This means the user can provide a single character to use the memory's saved value instead of a number, or simply use a number as normal. It is invalid to use a number to select a variable.
    - For example, the user can request a calculation for *"a + 12.00"* which will result in *"0.00 + 12.00 = 12.00"* since the value of memory slot 'a' is default 0.00. The user could exit and go to option V to save a value in 'c' as 4.23. From there they can proceed to option A and calculate *"2.5 * c"* which will result in *"2.5 * 4.23 = 10.575"*. Finally, the user does not require a variable, so "2.00 + 2.50 = 4.50" is valid, and likewise can use all parameters as variables such as "a + c" which results in "0.00 + 4.23 = 4.23".
6. If the user enters E, then the program will be terminated after showing a goodbye message.

Part B [40 marks]: Validating User Inputs

In your program, it is expected that any input the user provides will be validated by the program. It is expected that your program does not crash for any given user input and provides either the proper output for valid inputs, or an error message with a request to recollect the user's input for the failed entry. Error messages and the request for a new input should occur the moment the user has provided the invalid input. Below are some input validations your program should handle:

1. For any character that must be read by the program, it must validate that the character given is one of the available options. Each section has a different set of characters that must be validated: the main menu (B/U/A/V/E), the binary operators (+, -, *, /, %, P, X, I), the unary operators (S/L/E/C/F), the memory variables (a/b/c/d/e), submenus (B/U/E), etc. If an invalid character is detected, the program must print an appropriate error message and request a new character. This is repeated in a loop until success.
2. Any time the user is expected to give a floating point value the program should handle the case where they might insert a character by accident or something non-numeric. In other words, if the program fails to read a floating point value, then it should display an appropriate error message and request the number again until one is read in. This is repeated within a loop until success.
3. In option A, the user is allowed to provide a mix of variables or numbers for their calculations. This means the input could potentially be a character or a number. Your program must be able to automatically detect whether the input is a number or a character, you **should not** request the user to specify which one they'll provide. For example, the program can first check if the input is a number. On failure it will then try to read a character and validate that it is one of a/b/c/d/e. On failure again, the program should display an error message and request a new input, otherwise use the number or character read in successfully. The process loops until it successfully reads a number or valid character. HINT: scanf can return the number of inputs successfully read.
4. Finally, make sure to run the program and ensure it does not crash/fail on any given input.

Save the file as a C source file named **clc.c** in your working directory, i.e., **/home/yourUserName/fall/comp1400** (or other base directories). Note that ~ (also known as the home directory) is equivalent to the **/home/yourUserName** directory.

While in your working directory **(~/fall/comp1400)**, you can now compile the C program with the following command to produce an executable.

gcc -Wall clc.c -o clc.out

You can run the executable file with the following command.

./clc.out

Submission Instructions:

You have to submit your solution as **A SINGLE C FILE** into Brightspace before the deadline. The file name is your student id.c (for instance, if your student id is 1234567, the file name would be 1234567.c).

PLEASE DO NOT submit any other types of files, such as compiled files, files associated with specific software, etc. Therefore, for this assignment, you just need to submit the following file:

- One text file, named "your_id.c", which should be a simple text file (extension .c), containing your c program.

Submission of other file types will make problems for grading your assignments, and you will lose partial or complete marks.

Good Luck 😊