C Calculator

Purpose: The purpose of this assignment is to give you the opportunity to put your newly-learned C programming skills to the test and to brush off the cobwebs from what you already know about programming. **This is an individual assignment, not a team assignment.** A future assignment will build off of this assignment.

Project setup on your server (5%)

1. Login to your server.
2. Change directories to be in the directory `/home/your_user_name/CS_1337/assignments`
3. Create a directory called `lab_2`
4. Change directories to be in the directory `/home/your_user_name/CS_1337/assignments/lab_2`
5. Create a new file called `ccalc.c` (case-sensitive). No input parameters on the command line will be required.
6. Using good programming practices (comments, frequent testing, good variable naming, etc.), implement a program that satisfies all of the requirements listed below.
7. When you submit this assignment, the submission form will ask for the path to your assignment on your server. You can find the path by typing `pwd` in the assignment directory which will **p**rint the **w**orking **d**irectory. You should submit the path exactly as it is returned from the `pwd` command. It should look something like `/home/your_user_name/CS_1337/assignments/lab_2`

Program correctness requirements (80%)

Program correctness will be determined by the correct functionality of the program as well as adherence to format specifications.

Prompt user for operator

8. (5%) The program will first prompt the user for an operation. The possible operations are +, -, * and /. If *q* is entered as the operation, then the program should quit.
9. (5%) If any other operation is input (including inputs longer than a single character), the program should print the message “Please enter a valid operation” and reissue the original prompt on the next line.
10. (5%) The code that you use for validating the operator input should be extracted into a function.

Prompt user for the operand count

11. (5%) After a valid operation has been entered, prompt the user for an integer. (This will represent the number of integers to include in the calculation.)
12. (5%) The user must enter a valid integer greater than one. If the user's input is invalid (e.g., not a number, a floating point number, etc.), you should print the message "Please enter an integer greater than 1" and reissue the prompt. I suggest considering the use of the sscanf function for this purpose. We will not test your programs with integers that overflow the maximum integer size.
13. (5%) The code that you use for validating the operand count should be extracted into a function.

Prompt user for operands

14. (5%) Finally, prompt the user for *n* integers, where *n* is the number you received as input from the previous prompt. There should be one prompt per integer (in the manner shown below).
15. (5%) Any invalid input (e.g., not a number, a floating point number, etc.) should result in the message “Please enter an integer”, followed by the prompt for that integer on the next line. If the user specified the division operator, you should ensure that the input will not produce a divide-by-zero error (i.e., that all but the first of the n integers prompted for are nonzero). I suggest considering the use of the sscanf function for this purpose. We will not test your programs with integers that overflow or underflow the maximum or minimum integer sizes (negative operands should work as expected).
16. (5%) You should use an array to store these values.
17. (5%) The code that you use for validating the operands should be extracted into a function.

Perform the calculation

18. (15%) After receiving valid input, the program should then perform the operation on the numbers in the form `firstnum operator secondnum [operator thirdnum etc]`. The expression should be evaluated from left to right. The program should then print back the entire expression being computed (including the operands, operation, and result).
19. (5%) Your program should use a switch statement in this portion of the assignment.
20. (5%) For division, you should compute the result as integer division (i.e., dividing two numbers returns the integer part of the result).

Example program execution (with correct formatting)

(5%) A major goal of this assignment is to test your attention to detail. **Therefore, you must follow the output format shown below exactly**. Prompts in your program must exactly match the prompts shown. The same number of spaces that are used below must be used in your program. In particular, there should be one space after the prompt before the user input. There should be a space between each operand and operator. There will be no spaces on the right-hand side of any line after the output. Your program will be diffed against expected output, and formatting errors will result in a loss of points.

Below is the format for a sample operation, where red text indicates user-entered values:

```
Enter operation: +
Enter number of integers: 3
Enter integer 1: 33
Enter integer 2: 66
Enter integer 3: 1

Computing: 33 + 66 + 1 = 100

Enter operation: q
```

Here is another output of a possible execution of the program.

```
Enter operation: @
Please enter a valid operation
Enter operation: /
Enter number of integers: #
Please enter an integer greater than 1
Enter number of integers: 3
Enter integer 1: 10
Enter integer 2: F
Please enter an integer
Enter integer 2: 3
Enter integer 3: 3

Computing: 10 / 3 / 3 = 1

Enter operation: q
```

Other requirements (15%)

21. (5%) It will get tiresome retyping the compile command every time you make a change. For this reason, it is common to use a **makefile**. Makefiles are a simple way to organize code compilation. For this assignment you need to create a

makefile (using vim) called "makefile" that will build your executable from your source file(s). You will first need to install make by issuing the command: `sudo apt install make`. Follow this makefile tutorial here: https://makefiletutorial.com/. The first rule in your makefile should compile the code for this assignment into an executable called `ccalc`. To specify the name of the executable (recall the default was `a.out`), you use the -o flag followed by the desired name as exemplified in the makefile tutorial.

22. (5%) Before submitting this assignment, you need to push your code to your GitHub repo (see the EC2 tutorial section entitled "Everyday Git commands"). You should include a README.md file (case sensitive) in the project directory to explain briefly to the user how to compile (i.e., call `make`) and execute your program.
23. (5%) Your code will be evaluated for quality. In particular, you should observe the following:
    1. **Variable, class, and function naming conventions**. Each language has its own conventions. The most important thing is consistency and using meaningful names.
    2. **Clear and concise comments**. Functions should have comments. In addition, each block of code should be prefaced with a comment.
    3. **Indentations**. The most important thing is readability and consistency.