General Guidelines Regarding the Operation of the Program : Calc

- The calc.c program does account for multiplication and follow the specifications provided in lecture.
- Please be sure to use quotations when entering the multiplication argument if you choose to use multiplication arithmetic.
- If the computation of two numbers is negative, then the given computation of the two numbers will be treated as positive and the converted result will consist of a '-' sign indicating that the computation of the two numbers were negative.
- It is expected that all numbers follow the rules of their specified type and entered in their format. For example binary numbers only consists of 1s and 0s, therefore it is expected that the input from the user follows such rules.
    - Example of acceptable format:
        - + b101 b100 d
        - - -b100 b101 o
        - "*" d123 b101 OR "*" "b100" "b101" "d'
        - + x4B5 -x4B5 d
- Please use lowercase letters when indicating the type of number being computed and the output desired. Refer to the examples above.

Implementation of calc.c

- The program will first evaluate the four arguments being given, which should be given in the order <operator> <number1> <number2> <output>. The main function will check if the arguments are empty. If so an error message will be displayed specifying, which component is missing. Every string input for number1 and number2 is converted to a decimal number, which is used to compute the total (addition, subtraction, multiplication). Through a series of if statements, we first check the operator, then determine what type number1 and number2 are.

The numbers are converted to their respective decimal value (positive format regardless of whether number is negative). When computing the total, we check if the first index of the number inputted by the user was negative. If it was, we multiply that value by -1 to make the number representative of the input and compute the total. After the first three arguments in the input are checked and evaluated, the last input is evaluated to determine the desired output. The program evaluates, which kind of output is needed through a series of if statements. A character pointed is created and it is malloced to the necessary size to hold all the characters of the conversion. The integer is then converted to its necessary ASCII value to be returned as a string to the user. As stated in the instructions printf() can not be used, which was not implemented in returning the ASCII. A few challenges I ran into was in the conversion to ASCII as I was having trouble getting the proper format. In terms of big - O, there are 12 checks for each input number + 3 checks to determine the operator +4 checks to determine the output and 2 checks to determine if the number is negative. The program accounts for two 32 bit inputs as stated in lecture and the assignment. O(n) to convert each character into a digit.

Test Cases (Following Test Cases and Produced the Following Results Successfully):

+ b01000001010000100100001101000100 b0 d —> 1094861636
+ d123 d204 x —> 147
+ o37 o38 d —> 63
+ -b101 d23 x —> 12
+ x1D9 x80E1 d —> 33466
- -d123 -d24 x —> Note Total Value is -99 —> -63
- b1000 x1D9 o —> - 721
- d69 -d70 b —> 10001011
- o73 d12 x —> 2F

- b00111010000111111111011000001000 -
b01000001010000100100001110000100 x -> 7 B62398C
“*” b101 b100 d —> 20

Format:

Format.c takes in two inputs: one for the binary string and the second for its conversion to a float or an int. When the inputs are given the program checks if both parameters were given and converts the string input to an integer. Once an decimal is created it needs to be converted to its respective ASCII value. The function integertoDecimalASCII is called and returns a ASCII value. The output is given in decimal ASCII. I have three methods for converting the binary to a decimal ASCII or float ASCII. The program takes in a binary string converts it to a decimal integer then converts the integer to ASCII for int. For float the integer is converted by the `floatToASCII method, which was provided. Test cases have been handled for positive and negative infinity as well as NaN.`