README

Calc.c
//INCLUDES MULTIPLICATION//

Calc.c is a program that takes in two numbers, a m
athematical operation, and an output base. The pro
gram then applies the operation to the two numbers
 and outputs the result in the base of choice.

The arguments work as follows:
calc <op> <number1> <number2> <output base>

where <op> is the mathematical operator + for addi
tion, - for subtraction, or * for muliplication.

where <number1> and <number 2> follow the form -?(b
|o|d|x)d(n)d(n-1)...d(1)d(0), meaning
the numbers can begin with a negative sign, follow
ed by a base, b for binary, o for octal, d for dec
imal, and x for hexadecimal, followed by the digit
s of the number d(n) to d(0).

where <output base> is one of the following letter
s, b for binary, o for octal, d for decimal, and x
 for hexadecimal.

This program is capable of working with numbers co
ntaining up to 32-bits worth of information. Any a
ttempts to go over this value will result in undef
ined behavior or an overflow error.

This program runs in three main phases, argument c
hecking, base consolidation, and base conversion.

Argument checking is the smallest phase of the pro
gram, which simply checks the four require argumen
ts and makes sure that they follow the given input
 guidelines and regular expression. If they don't,
 errors are thrown and the program ends with a ret
urn value of -1 or (255).

Base Consolidation is the part of the program that takes the given input numbers and converts them into ints. This is done through four separate functions, BinToDec, OctToDec, HexToDec, and DecStringtoDec. Each of these change the given string and base into its proper decimal value and moves back into the main. Each method works slightly differently, with some using simple arithmatic (OctToDec) to bitshifting (HexToDec), but all 4 return an int that we can use the intrinsic C mathematical operations on. It is within these four functions that the checks for proper digits are made, and errors are thrown if not.

At this point in the program, the two ints are then added, multiplied, or subtracted based on the given operator.

Base Conversion is the final part of the program that takes the final product in int form, and transforms it back into a string of the proper type. This is done through four separate functions, DecToBin, DecToHex, DecToOct, and DecToDecString. Each of these are given an int and change that int into a string that represents a number in the given base. These methods heavily rely on bitshifting to accomplish proper value, however, in the case of DecToDecString, there is no bitshifting required, merely -'0' casting.

The number of operations in this code is based on numbers and their input and output bases. For changing inputs to decimal, the number of operations is based on the number of digits, and therefore O(len(n)). For changing the final operated on number back to decimal, the time complexity is based on the base of the number being compared to, where the number of iterations through the loop is based on the number of bits bitshifted per iteration, IE, b= 1 bit for binary, 4 for hex, 3 for octal, which gives the big O written as O(32/b).

0(len(n) + 0(32/b) = 0(2len(n)+32/b)
len(n) is the number of digits in the input numbers, and b is the number of bits required to hold the output number.

This program maintains leading zeroes for outputs to show the proper input/output boundry. This means that binary outputs will always be 32 digits long, octal outputs 11, hex outputs 8.

This program is written to spec, and so follows the following rule regarding negatives:

d6 = b110 = o6 = x6
-d6 = -b110 = -o6 = -x6

Negative numbers are simply their positive value with an - appended to the beginning of the output.