Anirudh Tunoori
netid: at813
RUID:165003175
pa2: calc.c / format.c

****Please note: As far as program implementation is concerned I ignored some of the vagueness in the project spec. and designed my algorithm and functions to correctly evaluate the test cases provided by the spec. All program interpretations were made taking the test cases into account.

Calc.c:
My program (on the basic level) utilizes a type defined struct called NumberN (see calc.h). For every invocation of my program the two number arguments are extracted and the appropriate fields of the struct like initialForm are filled out by my create method. Before that however, each argument of the input is checked for errors and each of the numbers are passed to an is_function for example isBinary determines whether a string is in fact a binary number. The sizes of the input are also checked at this stage. After the number structs are created, the program then avails to obtain a numeric (integer) value of each number. This process is split into two main functions bin_oct_dectoInt and hexToInt. The first function determines the value of a string that was defined to be an octal, binary, or a decimal while the latter function determines the value of a hexadecimal string. These two functions took up a considerable amount of time and planning. Testing these functions were time consuming as I (using pencil and paper) learned to become comfortable with converting between the number systems. These two functions are helped by the power function and proper use of the modulus operator. Since the program spec denied the use of type casting I had to use a (scratch made) power function that dealt with integers instead of doubles. luckily unlike the power function that I used in format.c the results of exponentiating were always integers. My reverse function was also useful in applying the sign and letter (x,d,o, or b) whenever necessary. The next part of my program involved conducting the appropriate operation. Finally the result of this operation was converted back to the string representation of the desired output base. Once again this involved two functions (one for hexadecimal and one for binary, octal, and decimal) that essentially reverse engineered the toInt functions. Finally the program calls a function that checks for errors and then the output is printed. The main challenge of this program came with checking the small details and coding around a hundred if statements. Developing my header file and adding functions as needed helped with the organization aspect of this assignment.

I did not use any advanced data structures outside character arrays/strings or techniques outside of working with pointers. In the context of runtime analysis I made use of dynamic allocation about three times throughout one execution cycle and two of these malloc allocations involved

my NumberN struct. The sizeof function was useful in this allocation. Time wise assuming n is the input size of the two number arguments The big-O of this program was 2n^2 +n +k or just O(n^2). The major contributor to this was processing the string bit by bit and latter converting the result back to a string. The actual operation/calculation was of a smaller order of magnitude.

Special features and considerations:
My program can do calculations involving multiplication as well. My program design enabled me to implement multiplication relatively easily. However when entering the input careful consideration needed to be made when entering the multiplication operation '*'; namely, the operation needed to be enclosed in either single or double quotes unlike the rest of the command line argument.
ex: ./calc '*' d4 d11 d works (d44) but,
./calc * b11 o23 x does not work.
I think this is because the c compiler does not interpret '*' as part of an argument. Anyways, besides that this feature seems to work for the test cases that I have tested.

Format.c:
This program had fewer functions and I did not utilize any structs or unions. In fact my header file solely consists of function definitions. I had a total of 8 functions (not including the main function). I had an isBinary function which checked the input sting to make sure that the inout string was 32 bits and consisting of only ones and zeros. I had a reverse function like in calc.c that reverses a string, which was useful in adding the sign, decimal point and other modifications. Also the bits had to be read in one direction of significance processed in another and returned/outputted in the firs direction. I had a power function that returns a double. This function can handle positive, negative, and zero exponents. I tried using two strategies for the numeric conversion part of the program the memcpy and floatToAscii code was giving me a segmentation fault that I could not figure out and after spending a few hours trying to design a similar function I abandoned that strategy entirely. I faced a similar problem with the union and bit shifting strategy which usually gave me an overflow error. It was difficult to test these functions and I was usually losing or somehow (unintentionally) flipping bits. So Instead i utilized a power function, the modulus operator, integer division, and a log10 function which was built using the change of bases formula with an ln(function) and the ln(10) constant. I also had a special cases function that came up with the appropriate output in the following case: 0.0e0 and -0.0e0, pinf, ninf, and NaN. Finally my last two functions dealt with the numeric conversion. the toInt function used 2's complement to do the conversion. The toFloat involved understanding all of the rules of IEEE754 single point precision. I have to thank my recitation instructor for helping me understand the structure and boundaries ie: 8 bits, 23 bits, and one sign bit. All in all outside trying and testing different strategies to do the numeric conversions there were not too

many challenges. There was a considerable time commitment that was necessary as each strategy needed to be almost entirely implemented before testing it.

Runtime wise I believe my program ran in 3n (or something really similar) -> O(n) time.
My space consumption/performance was also on the same order.

I feel the need to add that making the makefile was a considerable challenge and I really struggled with them. I do feel comfortable with them now and I don't force too many difficulties in the future. As it was my first time making a makefile…I spent a lot of time trying to figure out what was wrong with mine. The spec's instructions were a little vague on that front.

All in all i faced an enjoyable challenge with this project.