Numeric Base Conversion and Calculator

My approach to the problem involved taking every number format and converting it to a *string* of 1's and 0's that represented a bitstring. Using these bitstrings, I computed arithmetic by comparing the two strings (and of course keeping in mind the carry). If I needed to compute subtraction or a negative decimal while parsing, I would just take the two's complement of that number. As per instructions, I ignored any negative sign in front of an octal, hexadecimal. and binary string of bits.

I also implemented a multiplication function.

The hardest part of this method was definitely printing out in different formats. Printing out as a binary bit string was just one line, but printing out into decimal involved multiple steps from bitstring to integer to strings one character long. Octal presented a challenge since I had to take into account that groups of 3 bits do not divide evenly into 64 bits. I did this by assuming that the final bit could be considered a group of either 001 or 000 (which means that the leading character in a maximally long 64 bit octal input/ouput must be either 1 or 0, respectively).

In retrospect, I do not think that I had to finish my project the way that I did, but ultimately I am happy that I did so because it taught me a lot about C strings, dynamic memory, and formatting. Oh, and also the reuse of functions in both calc.c and format.c reinforced a lot of learning about the command line, building object files, linking source code, modularizing functions, etc. The only time I use anything but %s to print out something is during error messages.

All of my strings have 64 elements, and the number of inputs is constant. As such, since I loop through each string based on its size, the time it takes to traverse is constant. However, the only times when the number of loops is variable are when I am taking input (as I loop based on how many alphanumeric characters are input) and when I output a decimal (I output constant size 64, 22, and 16 strings for binary, octal, and hexademical bits respectively). So my program runs in linear time (O(n), where n is the size of my decimal input and/or output.).

TESTCASES (in addition to the ones provided in calc.pdf):

./calc + xFFFFFFFFFFFFFFFF d1 d d0

./calc - o72232 xFF24 b b1111111111111111111111111111111111111111111111111110111010101110110

./calc + b1000000000000011 d1111 x x000000000000845A

./calc - xFFFF o467 d d65224

./calc - xfff x110r d Unrecognized numeral in hex string: r. Results are now invalid. -d257

./calc -d100 xF0 d -d24000

./calc -d100 -d466 o o00000000000000000133010

./calc xFF xFF b b00000000000000000000000000000000000000000000000000000000011111110000000001

Format Interpretation

This part of the project was significantly easier than the former. In fact, I was able to use one of my functions from the previous to do half of what was asked of us to do, and the other half was handled mostly by the C language and by Professor Russell's provided code. I feel like the only big things I had to watch out for were the definitions of NaN, pinf, and ninf, and the fact that the compiler preserves the rightmost bit's sign when shifting right, meaning all right shifts are arithmetic. This threw me off when coding the conditions for pinf and ninf, but it wasn't anything that a few minutes in gdb couldn't explain.

The only part in which my format.c runs in loops variable amounts of times is outputting the two's complement integer, as it loops based on how many numerals are in the number. This means that while it is near constant, format runs in linear time.

TESTCASES (in addition to the ones provided in calc.pdf):

NOTE: I didn't check more int outputs than was provided since it was just my method from calc.c, which seemed to be working correctly both in calc and in calc.pdf format cases I checked my results against other online floating point calculators, namely: https://www.h-schmidt.net/FloatConverter/IEEE754.html

./format 01111111100000000000000000000001 float NaN

./format 01111111100000000000000000000000 float pinf

./format 11111111100000000000000000000000 float ninf

./format 11111101001111111111111000000000 float -1.595009e+37

./format 11111111001111111111111000000000 float -2.552014e+38

./format 10011111001111111111111000000000 float -4.065592e-20

./format 10011111001111111111111000000000 testingforformat The requested output format is not supported.

./format 100111110a11111111111111000000000 float The character a is displayed at position 9 instead of binary