Seong Kon Kim

Assignment 2

(1) Calc

I implemented this coding by using two functions. I tried to do the extra credit for multiplication. I found out that putting the * is getting the directory of my files so I used “” (quotations) around the asterisk to get the multiplication of 2 numbers. The Big-O of this program should be a O(n) as the program only goes through the value once to get the result.

Main

- It checks to see if there is any error in the input. Arithmetic operators that puts in the input is calculated in the main function and goes into AnswerConvesion() function. First value goes into decimalOne and next to decimalTwo, convert them to decimal with Conversion();

Int Conversion()

- This Function runs two times: argv[2] and argv[3]. It depends on decimal/binary/octal/hex type to go through the if-statement. It will go through 4 types (if not error) and change them to decimal number and returns the decimal result back to Main function.

Void AnswerConversion()

- Before this function, DecimalOne and DecimalTwo will go through arithmetic operator, + or -, or *; and variable answer will have the resulting answer in decimal form. the program will check to see what output form wanted in argv[4] and change to d/b/o/h.

Calc.h

- I didn’t have that much things to put globally so I had int error to pass on the value of error to print out and conversion() as function prototype.

(2) Format

- This program is changing 32bit binary sequence to int or float. I have two functions for int to string and float to string to print the value. The Big-O of the program should be O(nlogn) because it goes though the value once but on the mantissa there is a lot to take in for the program, making the time a bit slower.
-

Main

- In the main I check to see if argc is 3 and if bit sequence has 32 binary numbers. I actually put bit to int in main because it is simple. Using the 2's compliment, for positive number I just multiply every bit by 2. For negative number, I changed the bits, 0 to 1 and 1 to 0, and add by 1 to get the negative value of the product.

Char* floatFormat

- it divides the bit sequence in 3 parts: sign, exponent and mantissa.
- **Sign** just checks the first bit and see if it is negative(1) or positive(0) and moving the pointer by 1. Next exponent changes the binary to decimal to see the value.
- the **exponent** value changes by subtracting bias(127 in 32 bits) and also moves the pointer by 8.
- Lastly **mantissa** calculation depends on the value of the exponent. If the exponent equals -127, the result can be either 0, where mantissa is 0, and denormalized real number, where mantissa is not 0. If exponent equal to 128, then it can be either +/- infinity, where mantissa is 0, and not a number, where mantissa is not 0. And exponents between -126 to 127, is a normalize real number where mantissa is added by 1.0**.** I also calculated the number of digits after the decimal points.

Int intToString

- The function bring the value(int), destination (where to put the value(char array)) and index number for the next free space. It puts int value to string.

floatToString

- Same concept as intToString but it converts float to string.

Mul

- I made my own power function to help me calculate easier.