Mohit Mori (mam987)
CS211 Computer Architecture
Assignment 2: Computer Arithmetic

Calc

Description:

This program takes in 4 strings as input. The first is the desired operation. The next two are the numbers on which the operation is performed, and the fourth one is the base in which the output should be displayed. My program handles the operations: addition, subtraction, and multiplication, and can take input/output in the bases of decimal, binary, hex, and octal. The program first converts both numbers from an ASCII string to an int in base decimal. Once both converted, the desired operation is performed on the numbers and then converted back to an ASCII string representing the resulting number in the specified base.

For this program, I used a struct which holds the operation to be performed, as well as an array of integers. The array holds the input numbers. Because it is an array, the program can easily be expanded to hold and perform operations on more than 2 numbers. (Note: when using the multiplication operation, the argument/s should be in quotations or the escape character should be used '\' before the '*' to indicate multiply)

Challenges:

- Converting from an ASCII string to an int in base 10
- Converting from an int in base 10 to an ASCII string in the specified base

Time and Space complexity:

My program runs in time: **O(n + m)** (or Linear), where n is the length of the first input number and m is the length of the second input number. When converting to and from an ASCII string, the program loops through each digit of each number.

The space complexity of my program is: **O(max(m,n))** (or Linear), where n is the length of the first input number and m is the length of the second input number. The memory of the return ASCII String is allocated depending on the length of the output number, as well as the desired output base. In the worst case, the result would have one more place than the max of m and n and the desired output would be in binary. Therefore, the space needed to hold the output string is dependent on the number of bits of the longer input number, making it linear.

Format

Description:

This program takes 2 strings as input. The first one is the 32 bit binary sequence that is to be converted, and the second is the form in which the sequence should be interpreted. If the second string is ‘int’ the input sequence is converted to an integer representation using two’s complement. However, if the second string is ‘float’, IEEE754 will be used to convert the input bit sequence into a single precision floating point number which is printed in scientific notation.

If the interpretation type is int, the program converts the input binary ASCII string to an integer using two’s complement and then returns it. If it is float, the program breaks down the input binary ASCII string into three parts: the sign bit, the exponent, and the mantissa. Each is converted/interpreted individually and the multiplied as necessary to get the result. Format handles all normal cases, +/- 0, NaN, +/- infinity, as well as denormalized numbers. Additionally, the output for a normal float will always display with a negative sign if necessary, the number in the ones place, a decimal point, 6 numbers after the decimal point, followed by ‘e’ and the exponent.

Challenges:

- Making the program handle all the special cases (especially denormalized numbers)

Time and Space Complexity:

My program runs in time: **O(1)** (or Constant) because the input sequence is always going to be 32, and every time, all 32 bits will have to be interpreted. In the worst case, the interpretation is gong to be float, and the exponent is going to be 127, in which case the loop in the function power is going to loop 127 times. However, this still results in constant time.

The space complexity of my program is: **O(1)** (or Constant) because no matter the input, the same amount of space will be allocated for the exponent and mantissa strings which are always 8 and 23 bits respectively. The output being either an int or float will always have a constant amount of space required, not dependent on any input.