Derek Schatel
RUID: 032004123

Design Document – Calc.c and Format.c

Calc.c

The program initially verifies the input by making sure there are a sufficient number of arguments in the input string and that the arguments are valid. Afterward, it analyzes the input strings for the two numbers and creates an ArbitraryInt object that holds the data that the rest of the program will manipulate. This object holds values for whether the number is negative, the input format (Decimal, Octal, Hexadecimal or Binary), and a character array of the input ASCII string. It also holds data for an integer array that will be filled with 0's and 1's once the ASCII string is converted for arithmetic purposes. I decided to hold the data in this way in anticipation of implementing Arbitrary Precision arithmetic, but ultimately was unable to do so. Consequently, this program can only handle decimal values up to 32 bits in size.

Once these ArbitraryInt objects are created, the program verifies the format of the ASCII string based on the specified input type (for example, if the specified input type is Decimal, the ASCII string can only contain 0-9 as characters). Although this means that the program is essentially checking the input string twice (once for validity and once to convert it to an integer array), this is nontheless on the order of 2n, which is still linear time. I chose this because I did not want to process the string and create an integer array unless the string was first valid. Next, the program formats the number based on the input and converts it into an integer array of 0's and 1's. If the number is negative, it performs a "Twos Complement" conversion on the integer array in order to make arithmetic easier. It also 'pads' the integer array with additional 0's or 1's at the beginning in order to accommodate carrying bits as well as the sign bit when performing arithmetic. Before performing the arithmetic, the program "evens out" the two integer arrays so that they are the same length. This makes arithmetic easier because it can account for additional carries as well as preserving the sign bit.

Next, the program performs the addition or subtracting using standard binary addition or subtraction on the array entries, and returns a new integer array with the resulting bitstring. Finally, the program converts the resultant integer array back into the specified ASCII output (Decimal, Octal, Hexadecimal or Binary). It first does another Two's Complement conversion on it (since the outputs are considered positive numbers but for the '-' character preceding the output string), and then prints out the result in the specified format.

Important Note about Output: Due to the procedure used (specifically, padding the integer array and "evening out" the two arrays to make arithmetic easier), formatted outputs in Binary, Hex and Octal will inevitably have leading 0's. For example, instead of x422 as a Hex output, the program will display x0422. These are nonetheless the same result and the 0's can be ignored.

Format.c

The program first does some error checking to ensure that there are sufficient arguments and that the inputs have been entered correctly. If the format type designated is 'int', the program processes the 32-bit string and converts it to an integer value. It then converts that integer value into a string equivalent and reports it. The program does the same for the float value: it separately tracks the sign bit, exponent portion of the bitstring, and the mantissa. It then calculates the float value using a IEEE 754 Single Precision algorithm. Prior to actually processing the bitstring, the program parses the string for specified formats for 'exceptional' cases of positive and negative infinity, positive and negative NaN, and Zero. Although this means that the program will process a float bitstring's input twice, this will not take any extra time because the string will always at most be 32 characters in length, so this program can run in constant time.

After processing the string into a float value, the program finally converts it back into an ASCII string using an algorithm that Dr. Russell provided in class.