TCSS 333 Summer 2017, HW7

Due: by midnight Thursday August 10 (or with 10% late penalty by midnight Friday August 11)

Write an interactive program that will

- allow the user to enter a floating point number
- display the 32 bit binary representation of that number
- show how this binary representation can be converted to the equivalent floating point number

Your program must create the same output as shown in the two sample runs that are included below.
This includes breaking the 32 bits into 3 separate fields (sign, exponent, and fraction) as well as generating the fraction one bit at a time and applying the exponent one multiplication (or division) at a time. To facilitate grading, your formatting should be very similar.

You may write the entire program in one .c file. Organize your solution into a sensible set of functions.
As always, turning in code you may find on the internet is not acceptable. The goal of the assignment is for you to practice manipulating bits. If you solve the problem without examining and manipulating the bits, you may lose a lot of points. Bits are manipulated using shift (<< or >>) and the bitwise operators & | ~ ^. Don't use division (/) or mod (%) where you can simply shift bits instead. / and % are more expensive.

Do not turn all the bits into a string of 0's and 1's.
Do not turn all the bits into an array or list of 0's and 1's.

You will probably need to shift bits around, so be ready to use an unsigned int. Otherwise you may run into sign extension. You may use unions and/or bit fields if you find these useful.

The float data type of C usually requires 32 bits (or 4 bytes). To ensure this is true for your machine, try this:

```
printf("float size in bytes: %d", sizeof(float));
```

If the value printed out is not 4, please talk to me about how to do the assignment.

If you missed class, there is an explanation of the floating point format at:
http://en.wikipedia.org/wiki/Single-precision_floating-point_format

The program can be completed in less than 100 lines of code.
Be sure to test both negative and positive numbers.
Sample runs appear on the following pages.

Test 1:

```
Enter a float: 34.789
Your float was read as: 34.789001
Your float in 32 bits: 01000010000010110010011111110000
Sign: 0
Exponent: 10000100
Fraction: 00010110010011111110000

Creating the fraction:
fraction = 1.000000 (the implicit 1)
fraction = 1.000000, after skipping 0.500000
fraction = 1.000000, after skipping 0.250000
fraction = 1.000000, after skipping 0.125000
fraction = 1.062500, after adding 0.062500
fraction = 1.062500, after skipping 0.031250
fraction = 1.078125, after adding 0.015625
fraction = 1.085938, after adding 0.007812
fraction = 1.085938, after skipping 0.003906
fraction = 1.085938, after skipping 0.001953
fraction = 1.086914, after adding 0.000977
fraction = 1.086914, after skipping 0.000488
fraction = 1.086914, after skipping 0.000244
fraction = 1.087036, after adding 0.000122
fraction = 1.087097, after adding 0.000061
fraction = 1.087128, after adding 0.000031
fraction = 1.087143, after adding 0.000015
fraction = 1.087151, after adding 0.000008
fraction = 1.087154, after adding 0.000004
fraction = 1.087156, after adding 0.000002
fraction = 1.087156, after skipping 0.000001
fraction = 1.087156, after skipping 0.000000
fraction = 1.087156, after skipping 0.000000
fraction = 1.087156, after skipping 0.000000

Applying the exponent:
unbiased exponent = 5
times 2 = 2.174313
times 2 = 4.348625
times 2 = 8.697250
times 2 = 17.394501
times 2 = 34.789001

Final Answer:  34.789001
```

Test 2:

```
Enter a float: 0.1357
Your float was read as: 0.135700
Your float in 32 bits: 00111110000010101111010011110001
Sign: 0
Exponent: 01111100
Fraction: 00010101111010011110001

Creating the fraction:
fraction = 1.000000 (the implicit 1)
fraction = 1.000000, after skipping 0.500000
fraction = 1.000000, after skipping 0.250000
fraction = 1.000000, after skipping 0.125000
fraction = 1.062500, after adding 0.062500
fraction = 1.062500, after skipping 0.031250
fraction = 1.078125, after adding 0.015625
fraction = 1.078125, after skipping 0.007812
fraction = 1.082031, after adding 0.003906
fraction = 1.083984, after adding 0.001953
fraction = 1.084961, after adding 0.000977
fraction = 1.085449, after adding 0.000488
fraction = 1.085449, after skipping 0.000244
fraction = 1.085571, after adding 0.000122
fraction = 1.085571, after skipping 0.000061
fraction = 1.085571, after skipping 0.000031
fraction = 1.085587, after adding 0.000015
fraction = 1.085594, after adding 0.000008
fraction = 1.085598, after adding 0.000004
fraction = 1.085600, after adding 0.000002
fraction = 1.085600, after skipping 0.000001
fraction = 1.085600, after skipping 0.000000
fraction = 1.085600, after skipping 0.000000
fraction = 1.085600, after adding 0.000000

Applying the exponent:
unbiased exponent = -3
divided by 2 = 0.542800
divided by 2 = 0.271400
divided by 2 = 0.135700

Final Answer:  0.135700
```