CS 50 Homework 1:
Floating Point Utilities in C

The IEEE floating point standard is a rich and complex design, and the best way to really understand how it represents numbers is to build utilities to manipulate floating point numbers as raw bits. You will need to write three functions for dealing with floating points numbers: floating_info, ieee_16_info, and as_ieee_16.

The first two functions are string formatting functions that will convert the input (either a union type or a 16b value depending on the option) to a formatted string describing the internal information. For a normal floating point number it it should have a + or - depending on the sign bit, then the significand in binary in the format of the most significant bit, a decimal point, and then the remaining 23 (for 32b floats) or 10 (for 16b floats) bits, a space, and then the exponent as 2^ and then a signed integer. As an example, the floating point number .5 is represented as "`+1.00000000000000000000000 2^-1`" (without the quotation marks).

For denormalized numbers you should output them with a leading 0, then all 23 or 10 bits and with the exponent. 0 should be "`+0`" or "`-0`" depending on the sign. An infinity should be "`+INF`" or "`-INF`", and a NAN should be "`NaN`". These functions should be safe: if the buffer passed in is insufficiently large it should truncate the output to fit.

The final function, as_iee_16, should take a 32b floating point value (passed as a union type to make direct manipulations easier) and convert it to 16b IEEE half-precision floating point. There are several corner cases you need to consider, including subnormals, infinities, NANs (which remain NANs), and rounding (using 'rounding towards even') when converting a 32b floating point value to a 16b value.

You will only turn in your `floating.c` file, and you must not change the header file `floating.h`, and you can't include any other libraries into `floating.c`. This assignment is due on April 24th at 2000 PDT (10:00 pm Davis time). You must work on this assignment individually.

Expected Homework Outcomes

1. Really ***understand*** the structure and complexity of IEEE Floating Point

Hints

You can (and indeed should) write more tests in `main.c`. The main function you are provided just takes the arguments on the command line, use `strtof` to read them as floating point numbers, and then pass them through the various functions and print them out. Although useful for development you would be advised to write further tests, such as ones that take known hexadecimal values. A useful utility for creating such tests is the h-schmidt.net floating point converter page at https://www.h-schmidt.net/FloatConverter/IEEE754.html

One particular important set of tests will be various corner cases, best described as "magic numbers". You ***must not*** privately share interesting magic numbers you discover with your

fellow students. Instead, you ***may*** post them publicly in the “Magic Number” announcement thread, either in decimal form (e.g. `3.141592` or `-1e-45`) or in hexadecimal (e.g. `40490fd8` or `80000001`), and include a notion why they are special (in this case, “the closest floating point approximation to pi, because 🥧 is tasty” and “the negative number closest to 0 in 32b floating point”)

The reason for this policy is that the autograder is going to, like Homework 0, be blind: A basic sanity test will be public but the rest will be hidden cases. So coming up with tests is important. And at the same time I want students to collaborate on interesting tests, but I want such collaboration to be *universal*: all students should benefit from shared expertise.

So, for example, if your fellow student asks for help and you see a magic number that would trigger a bug, don’t tell your fellow student, instead tell ***everyone***.