DIY Floating Point Addition

This program reads two 32 bit floating point numbers from standard input
and adds them together using Câ€™s integer operations.

The approach I took to completeing this program was to reuse parts of the last
program, A1. I reused the int to binary output function, and by doing so
I just had to make sure I outputted the correct number of spaces / newlines.
I then added in a chunk of code from the example program, a struct with
32 unsigned bits, which allows me to access specific parts of the mantissa and
exponent. I then typedefâ€™d the union so that I could load the two floating
point numbers into different unions, and I created the following function:

```
float add_floating_point(float_32 first_int, float_32 second_int);
```

Inside this function I use a couple of if statements to detect a few cases
of floating point addition. The exponents are either the same, or different,
and if they are different one is larger then the other. So I adjust the
exponents so that they are the same, and to do this I increment the smaller
exponent while right shifting the smaller exponentâ€™s mantissa. This gets the
two exponents to be the same, and I deal with certain cases like the hidden bit,
shifting too far right, and rounding being off by 1 with additional if
statements.
I finally have one other case to detect infinite numbers and if I find this,
I set the exponent to 255 (1111 1111) and the mantissa to
0 (000 0000 0000 0000 0000 0000).

All of my test cases pass, as well as the provided test cases. I verified this
by running them manually and with the Makefile that I created. I compared the
output of my program (emulated portion) to what the hardware outputted.
The hardware is basically just printing out the addition of two floating point
numberâ€™s using Câ€™s standard library.

I learned a fair amount about floating point numbers. For one, they are a pain
to deal with manually. There are so many random cases to deal with, and a lot of
test cases were required to make sure the program was outputting correctly.

I finally feel that this program deserves a .85, as all of my test cases pass
successfully, and the output of my program matches the hardware output on my
machine.