*Exercise 5: Budling Bit Values(float.cc)*

***Explained***

* The first puzzle here seems to be that, without manipulating any of the original code, the f value is different from the original.
* Not just in terms of it being converted to scientific notation, but why is a positive value suddenly negative?
* This is a case where we must first remember how binary works, specifically 2’s compliment. The left most bit will always indicate either negative or positive. In this case we have an uint32\_t x = 2,693,408,940. Now this value is unsigned meaning that the conversion to float should be a little smoother. If this were a signed bit, the leftmost bit would be either a 0 (for positive) and 1 (for negative), and when converting this could throw off the conversion value
* In this case, since it is already unsigned, we have a different set of issues. Floats in C++ are depicted in scientific notation. The problem with that is that scientific notation requires both a bit to represent the sign and a bit to represent the exponent.
* The largest value an uint32\_t is 2^(32-1), meaning our number will experience an overflow. Then when we try to convert that overflow we have issues as it is trying to represent it as a sign.

Text

Description automatically generated

* Not sure if this is part of the problem but reinterpret\_cast<> should only be used on pointers and not actual values (maybe be irrelevant here because float& is implying the pointer as opposed to the value).
* Fixed the issue by first representing our initial x and y values as floats (lol, I know that’s the cheap way out…).
* Just saw this line, “*Starting with the floating-point skeleton code I’ve provided, and using only basic bitwise operations, set the necessary bits in x to represent the value 0b1.101 x 2^-5. The ensuing reinterpret\_cast() should output the decimal equivalent, -5.078125e-02.”*
* *In the interest of time removed the reinterpreted cast in order to do bitwise operations without having to worry about the decimal equivalent of a binary number*