The Google Calculator Glitch, could float vs. double be a possible reason?

Asked 16 years, 4 months ago Modified 1 year, 3 months ago Viewed 3k times

6

With Google's newfound inability to do math correctly (check it! according to Google `500,000,000,000,002 - 500,000,000,000,001 = 0`), I figured I'd try the following in C to run a little theory.

```
int main()
{
   char* a = "399999999999999";
   char* b = "399999999999998";

   float da = atof(a);
   float db = atof(b);

   printf("%s - %s = %f\n", a, b, da-db);

   a = "500000000000002";
   b = "500000000000001";
   da = atof(a);
   db = atof(b);
   printf("%s - %s = %f\n", a, b, da-db);
}
```

When you run this program, you get the following

```
   399999999999999 - 399999999999998 = 0.000000
   500000000000002 - 500000000000001 = 0.000000
```

It would seem like Google is using simple 32 bit floating precision (the error here), if you switch float for double in the above code, you fix the issue! Could this be it?

c math google-search

Share
Improve this question
Follow

edited Sep 8, 2023 at 0:06
Stephen Ostermiller ♦
25.5k ● 16 ● 94 ● 114

asked Aug 25, 2008 at 23:09
mauriciopastrana
5,070 ● 7 ● 36 ● 36

7 Answers

Sorted by: Highest score (default)

4

For more of this kind of silliness see this nice article pertaining to Windows calculator.

When you change the insides, nobody notices

> The innards of Calc - the arithmetic engine - was completely thrown away and rewritten from scratch. The standard IEEE floating point library was replaced with an arbitrary-precision arithmetic library. This was done after people kept writing ha-ha articles about how Calc couldn't do decimal arithmetic correctly, that for example computing 10.21 - 10.2 resulted in 0.0100000000000016.

Share
Improve this answer
Follow

edited Sep 7, 2023 at 18:52
Glorfindel
22.6k ● 13 ● 89 ● 116

answered Aug 25, 2008 at 23:19
Frank Krueger
70.9k ● 48 ● 164 ● 211

2

In C#, try `(double.maxvalue == (double.maxvalue - 100))`, you'll get `true` but that's what it is supposed to be.

Thinking about it, you have 64 bit representing a number greater than `2^64` (`double.maxvalue`), so inaccuracy is expected.

Share
Improve this answer
Follow

edited Sep 8, 2023 at 0:08
Stephen Ostermiller ♦
25.5k ● 16 ● 94 ● 114

answered Aug 26, 2008 at 8:19
gil
2,298 ● 6 ● 25 ● 31

2

> It would seem like Google is using simple 32 bit floating precision (the error here), if you switch float for double in the above code, you fix the issue! Could this be it?

No, you just defer the issue. doubles still exhibit the same issue, just with larger numbers.

Share Improve this answer Follow

answered Aug 26, 2008 at 8:08
DrPizza
18.3k ● 7 ● 42 ● 53

@ebel

1

> thinking about it, you have 64 bit representing a number greater than 2^64 (double.maxvalue), so inaccuracy is expected.

2^64 is not the maximum value of a double. 2^64 is the number of unique values that a double (or any other 64-bit type) can hold. `Double.MaxValue` is equal to 1.79769313486232e308.

Inaccuracy with floating point numbers doesn't come from representing values larger than `Double.MaxValue` (which is impossible, excluding `Double.PositiveInfinity`). It comes from the fact that the desired range of values is simply too large to fit into the datatype. So we give up precision in exchange for a larger effective range. In essense, we are dropping significant digits in return for a larger exponent range.

@DrPizza

> Not even; the IEEE encodings use multiple encodings for the same values. Specifically, NaN is represented by an exponent of all-bits-1, and then any non-zero value for the mantissa. As such, there are 252 NaNs for doubles, 223 NaNs for singles.

True. I didn't account for duplicate encodings. There are actually 2^{52}-1 NaNs for doubles and 2^{23}-1 NaNs for singles, though. :p

Share
Improve this answer
Follow

edited Aug 26, 2008 at 17:20

answered Aug 26, 2008 at 15:46

Derek Park
46.8k ● 16 ● 59 ● 76

0

> 2^64 is not the maximum value of a double. 2^64 is the number of unique values that a double (or any other 64-bit type) can hold. Double.MaxValue is equal to 1.79769313486232e308.

Not even; the IEEE encodings use multiple encodings for the same values. Specifically, NaN is represented by an exponent of all-bits-1, and then *any* non-zero value for the mantissa. As such, there are 2^{52} NaNs for doubles, 2^{23} NaNs for singles.

Share Improve this answer Follow

answered Aug 26, 2008 at 17:04

DrPizza
18.3k ● 7 ● 42 ● 53

0

> True. I didn't account for duplicate encodings. There are actually 252-1 NaNs for doubles and 223-1 NaNs for singles, though. :p

Doh, forgot to subtract the infinities.

Share Improve this answer Follow

answered Aug 26, 2008 at 21:54

DrPizza
18.3k ● 7 ● 42 ● 53

0

The rough estimate version of this issue that I learned is that 32-bit floats give you 5 digits of precision and 64-bit floats give you 15 digits of precision. This will of course vary depending on how the floats are encoded, but it's a pretty good starting point.

Share Improve this answer Follow

answered Sep 12, 2008 at 23:14

John Meagher
24.6k ● 14 ● 56 ● 57