Only two things are infinite, the universe and human stupidity, and I'm not sure about the former.

— Albert Einstein —

http://www.azquotes.com/quote/87292
## Floating Point Math

### Anti-Patterns:
- Not accounting for roundoff errors
  - Tests for floating point equality
- Not handling special values
- Float used if integer does the job
  - Not always good for “big” numbers

### Floating Point Math:
- Exponent + Mantissa representation
  - 32-bit, 64-bit, others on some systems
- Roundoff errors due to finite number of mantissa bits
- Special values:
  Infinity, Not A Number (NaN), denorms, signed zero

---

**IEEE Floating Point Format**

- Single Precision: 32 bits total

<table>
<thead>
<tr>
<th>S</th>
<th>EXPONENT</th>
<th>MANTISSA</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 bit</td>
<td>23 bits (with implicit leading 1.)</td>
<td></td>
</tr>
</tbody>
</table>

- Value = (+/-) 1.Mantissa * 2(Exponent-127)
- Sign: 0=positive; 1=negative
- Exponent: 127 bias, radix 2
  - value is EXPONENT – 127
- Mantissa: implicit 1.
  - value is 1.MANTISSA (binary)

**Special zero value:**
- zero = 0x00000000
Special Values

- **Inf: Infinity**
  - E.g., result when dividing by zero, or overflow

- **Denormalized**
  - Number smaller than smallest fraction
    - \( \sim 10^{-45} \ldots \sim 10^{-38} \) No implicit leading 1 in mantissa

- **NaN: “Not a Number”**
  - E.g., square root of negative number
  - Signaling NaN throws exception
  - Default is usually “silent” NaN (no exception)

- **Silent NaN Comparison Pitfall:**
  - Comparison with NaN is always false
  - if (CurrentSpeed > SpeedLimit) {shutdown}
    - Comparison is false for CurrentSpeed of NaN \( \Rightarrow \) no shutdown
  - (NaN == NaN) is also false (surprise!); use isnan()
NaN and the Robot Apocalypse

RECsbot Speed Limit Tests

- cmd = 1 m/s: No speed limit violation
- cmd = 3 m/s: Speed limit enforced
- cmd = Inf: Speed limit enforced
- cmd = NaN: Speed limit violated

End of test

Speed-limit violation occurred when exceptional input sent as speed command

Distribution Statement A - Approved for public release; distribution is unlimited. NAVAIR Public Affairs Office tracking number 2013-74, NREC internal case number STAA-2012-10-23

© 2017 Philip Koopman
Roundoff Errors

**Rounding error due to limited bits**
- Mantissa: 24 bits (implicit leading one)
  - E.g.: all zero mantissa bits $\Rightarrow 1.00000000000000000000002$
- More than 24 bits of value won’t fit
  - Converting int to float to int to float in a chain gives:
    $0x72345673 \Rightarrow 1916032640.0 \Rightarrow 0x72345680 \Rightarrow 1916032640.0$

**Rounding error due to imprecise representation**
- IEEE 754 is radix 2, so decimal fractions can be inexact
  - Repeatedly add 0.1 to a 32-bit float and you get....
    $0.1, 0.2, ..., 2.799999, ..., 49.999809, ..., 99.999046$

**Floating point comparison pitfall:**
- if (fa == fb) might not match due to rounding error
  - Use a comparison interval, e.g.,:
    if (fabs(result - expectedResult) < 0.00001)
Prove you are human:

0.1 + 0.2 = ?

0.30000000000000004
Patriot Missile mishap

- 1991: Scud kills 28 American (Desert Storm)
  “after about 20 hours, the inaccurate time calculation becomes sufficiently large to cause the radar to look in the wrong place”
  - “Range gate” used to look where target is predicted to be next
  - Target track is lost if range gate is wrong, resulting in a miss
  - The incident happened 100 hours after the last system reset

What was the root cause?

- Patriot designed for aircraft and frequent mobile relocations
  - Scud missiles travel at Mach 5 (3750 mph); Patriot deployed in fixed location
- Even a small round-off error matters when computing distance = velocity * time
  - Large accumulated base time and high velocity leads to a failure
Time is integer 10ths of second
- Converted to 24-bit fractional value for calculation
- 0.1 seconds is not an “even number” = 0.00011001100110011001100110011001100110011...
- At 100 hours, resultant round-off is 0.000000095 decimal [https://goo.gl/5ik1au]

After 100 hours error was 0.344 seconds = 697 meters error (per GAO report)
Best Practices for Floating Point

- **Use integer math if you can**
  - Scaled integer (e.g., 10ths of a second)
  - Binary Coded Decimal (BCD) + radix point
  - Fixed point (e.g., value *256)

- **Handle special values**
  - NaN is especially tricky to get right

- **Manage and handle roundoff error**
  - Doubles give more bits to work with (53-bit mantissa)
    - But fundamentally, all problems are still there
  - Don’t use floating point as an iterator, including time!

- **Comparisons are especially problematic (NaN, roundoff)**

<table>
<thead>
<tr>
<th>24 bits</th>
<th>8 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTEGER</td>
<td>FRACTION</td>
</tr>
<tr>
<td>+</td>
<td></td>
</tr>
<tr>
<td>INTEGER</td>
<td>FRACTION</td>
</tr>
<tr>
<td></td>
<td>24 8</td>
</tr>
<tr>
<td>Fixed Point Addition (uses normal integer addition CPU hardware)</td>
<td></td>
</tr>
</tbody>
</table>
Hey, check it out: $e^{\pi} - \pi$ is 19.999099979. That's weird.

Yeah. That's how I got kicked out of the ACM in college.

... What?

During a competition, I told the programmers on our team that $e^{\pi} - \pi$ was a standard test of floating-point handlers -- it would come out to 20 unless they had rounding errors.

That's awful.

Yeah, they dug through half their algorithms looking for the bug before they figured it out.

https://xkcd.com/217/