Floating point numbers

On GitHub: https://github.com/akalenuk/floating-point-experiments
On Medium (soon): https://medium.com/@okaleniuk

“One, two, many” counting system

- Our only numbers are “1”, “2” and “many”.
- There is no 0.
- There is no negative numbers.
- No rational and of course irrational numbers.

Isn’t it simple?

Addition is very simple indeed

	+1	+2	+many
1	2	many	many
2	many	many	many
many	many	many	many

“One, two, many” counting system

Subtraction raises so many questions

	-1	-2	-many
1	?	?	?
2	1	?	?
many	?	?	?

- What do we do with “1-1”?.
- What do we do with “1-2”?
- What’s “many-many”?
- What’s “many-1”?

“One, two, many” counting system

Since 0 isn’t part of our counting system, “1-1” is not a number.

So isn’t “1-2”.

“Many-many” could be anything. It may be a number, or not.

“Many-1” is polemic.

By tshrinivasan (Own work) [CC BY-SA 3.0], via Wikimedia Commons

By Balaram Mahalder (Own work) [CC BY-SA 3.0 or GFDL], via Wikimedia Commons

Need for a standardization

We have to balance applicability, so it would work for most.
But different applications require different properties.
Therefore we have to agree on simplifications.

Subtraction after “standardization”

	-1	-2	-many
1	NaN	NaN	NaN
2	1	NaN	NaN
many	many	Undef	Undef

Incorrect, but convenient.
(Also the guy on the right has a knife)

Floating point arithmetics

- Standardized by IEEE 754. Not C++ specific, but universal.
- Covers finite grids of rational numbers.
- Treats overflows as “`+infinity`”, “`-infinity`”*.
- Treats underflows as “+0” “-0”**.
- Has a “`not a number`” special value***.

* - actually bear a semantic of “still a number, we just can’t count it”.
`+infinity + 1 = +infinity`.

** - It also has subnormal numbers, which are numbers with the least possible exponent and trailing zeros in the significant. These can be cast to zero automagically to avoid performance overhead.

*** - Which can be quiet of signaling, so there are two of them****.

**** - technically, there is a whole class.

Examples

```
0    - 0 00000000 00000000000000000000000
-0   - 1 00000000 00000000000000000000000
1    - 0 01111111 00000000000000000000000
-1   - 1 01111111 00000000000000000000000
```

```
15         - 0 10000010 11100000000000000000000
15.9375    - 0 10000010 11111110000000000000000
0.9375     - 0 01111110 11100000000000000000000
```

```
inf        - 0 11111111 00000000000000000000000
-inf       - 1 11111111 00000000000000000000000
-nan       - 1 11111111 10000000000000000000000 (1./0. * 0.)
-nan       - 1 11111111 10000000000000000000000 (0./0.)
-nan       - 1 11111111 10000000000000000000000 (std::sqrt(-1.))
```

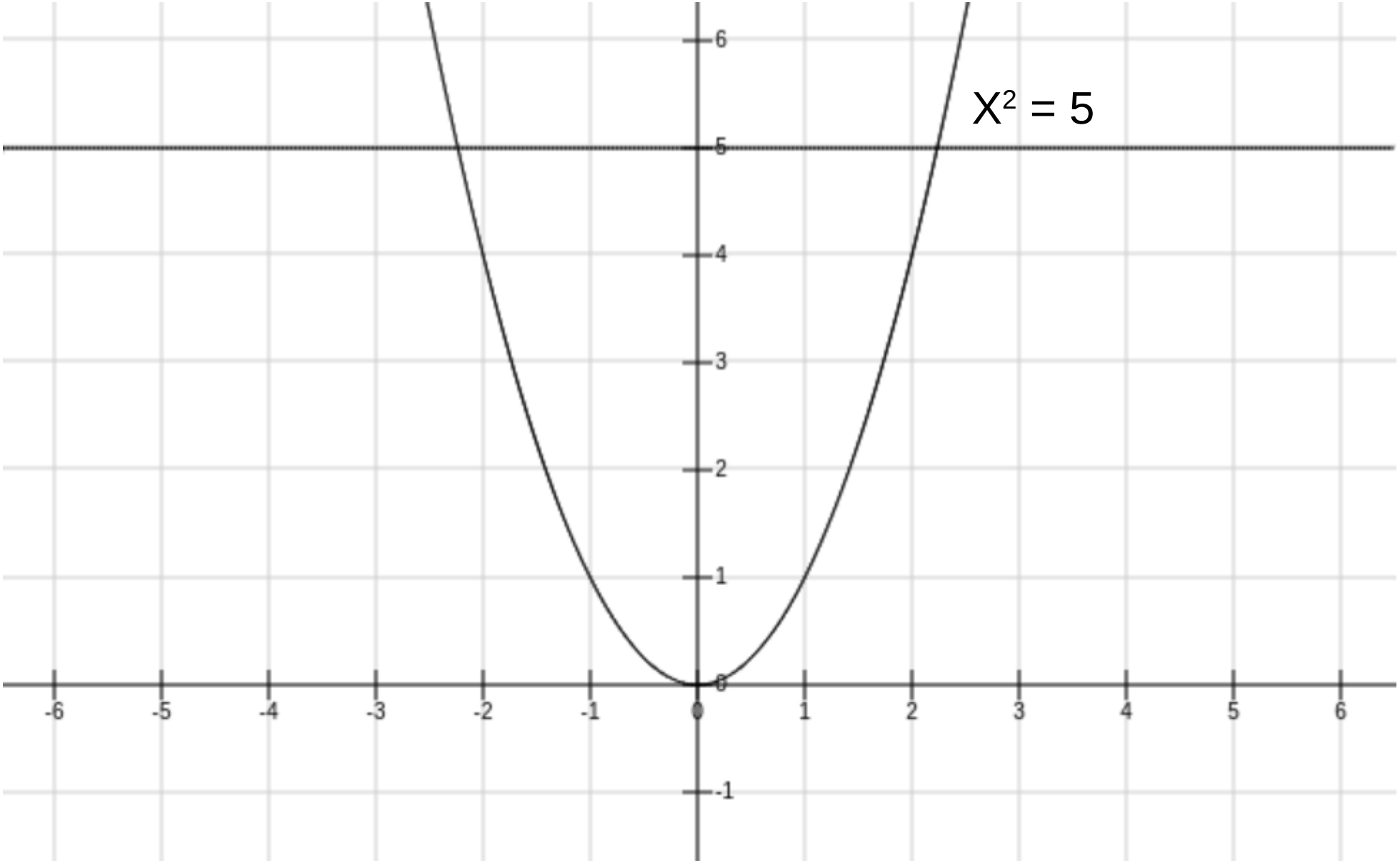
Prepare for the big boring table!..

1	- 0 01111111 00000000000000000000000
2	- 0 10000000 00000000000000000000000
4	- 0 10000001 00000000000000000000000
8	- 0 10000010 00000000000000000000000
16	- 0 10000011 00000000000000000000000
32	- 0 10000100 00000000000000000000000
64	- 0 10000101 00000000000000000000000
128	- 0 10000110 00000000000000000000000
256	- 0 10000111 00000000000000000000000
512	- 0 10001000 00000000000000000000000
1024	- 0 10001001 00000000000000000000000
2048	- 0 10001010 00000000000000000000000
4096	- 0 10001011 00000000000000000000000
8192	- 0 10001100 00000000000000000000000
16384	- 0 10001101 00000000000000000000000
32768	- 0 10001110 00000000000000000000000
65536	- 0 10001111 00000000000000000000000
131072	- 0 10010000 00000000000000000000000
262144	- 0 10010001 00000000000000000000000
524288	- 0 10010010 00000000000000000000000
1.04858e+06	- 0 10010011 00000000000000000000000
2.09715e+06	- 0 10010100 00000000000000000000000
4.1943e+06	- 0 10010101 00000000000000000000000
8.38861e+06	- 0 10010110 00000000000000000000000
1.67772e+07	- 0 10010111 00000000000000000000000
3.35544e+07	- 0 10011000 00000000000000000000000
6.71089e+07	- 0 10011001 00000000000000000000000
1.34218e+08	- 0 10011010 00000000000000000000000
2.68435e+08	- 0 10011011 00000000000000000000000
5.36871e+08	- 0 10011100 00000000000000000000000
1.07374e+09	- 0 10011101 00000000000000000000000
2.14748e+09	- 0 10011110 00000000000000000000000
4.29497e+09	- 0 10011111 00000000000000000000000
8.58993e+09	- 0 10100000 00000000000000000000000
1.71799e+10	- 0 10100001 00000000000000000000000
3.43597e+10	- 0 10100010 00000000000000000000000
6.87195e+10	- 0 10100011 00000000000000000000000
1.37439e+11	- 0 10100100 00000000000000000000000
2.74878e+11	- 0 10100101 00000000000000000000000
5.49756e+11	- 0 10100110 00000000000000000000000
1.09951e+12	- 0 10100111 00000000000000000000000
2.19902e+12	- 0 10101000 00000000000000000000000
4.39805e+12	- 0 10101001 00000000000000000000000
8.79609e+12	- 0 10101010 00000000000000000000000
1.75922e+13	- 0 10101011 00000000000000000000000
3.51844e+13	- 0 10101100 00000000000000000000000
7.03687e+13	- 0 10101101 00000000000000000000000
1.40737e+14	- 0 10101110 00000000000000000000000
2.81475e+14	- 0 10101111 00000000000000000000000
5.6295e+14	- 0 10110000 00000000000000000000000
1.1259e+15	- 0 10110001 00000000000000000000000
2.2518e+15	- 0 10110010 00000000000000000000000
4.5036e+15	- 0 10110011 00000000000000000000000
9.0072e+15	- 0 10110100 00000000000000000000000
1.80144e+16	- 0 10110101 00000000000000000000000
3.60288e+16	- 0 10110110 00000000000000000000000
7.20576e+16	- 0 10110111 00000000000000000000000
1.44115e+17	- 0 10111000 00000000000000000000000
2.8823e+17	- 0 10111001 00000000000000000000000
5.76461e+17	- 0 10111010 00000000000000000000000
1.15292e+18	- 0 10111011 00000000000000000000000
2.30584e+18	- 0 10111100 00000000000000000000000
4.61169e+18	- 0 10111101 00000000000000000000000
9.22337e+18	- 0 10111110 00000000000000000000000
1.84467e+19	- 0 10111111 00000000000000000000000
3.68935e+19	- 0 11000000 00000000000000000000000
7.3787e+19	- 0 11000001 00000000000000000000000
1.47574e+20	- 0 11000010 00000000000000000000000
2.95148e+20	- 0 11000011 00000000000000000000000
5.90296e+20	- 0 11000100 00000000000000000000000
1.18059e+21	- 0 11000101 00000000000000000000000
2.36118e+21	- 0 11000110 00000000000000000000000
4.72237e+21	- 0 11000111 00000000000000000000000
9.44473e+21	- 0 11001000 00000000000000000000000
1.88895e+22	- 0 11001001 00000000000000000000000
3.77789e+22	- 0 11001010 00000000000000000000000
7.55579e+22	- 0 11001011 00000000000000000000000
1.51116e+23	- 0 11001100 00000000000000000000000
3.02231e+23	- 0 11001101 00000000000000000000000
6.04463e+23	- 0 11001110 00000000000000000000000
1.20893e+24	- 0 11001111 00000000000000000000000
2.41785e+24	- 0 11010000 00000000000000000000000
4.8357e+24	- 0 11010001 00000000000000000000000
9.67141e+24	- 0 11010010 00000000000000000000000
1.93428e+25	- 0 11010011 00000000000000000000000
3.86856e+25	- 0 11010100 00000000000000000000000
7.73713e+25	- 0 11010101 00000000000000000000000
1.54743e+26	- 0 11010110 00000000000000000000000
3.09485e+26	- 0 11010111 00000000000000000000000
6.1897e+26	- 0 11011000 00000000000000000000000
1.23794e+27	- 0 11011001 00000000000000000000000
2.47588e+27	- 0 11011010 00000000000000000000000
4.95176e+27	- 0 11011011 00000000000000000000000
9.90352e+27	- 0 11011100 00000000000000000000000
1.9807e+28	- 0 11011101 00000000000000000000000
3.96141e+28	- 0 11011110 00000000000000000000000
7.92282e+28	- 0 11011111 00000000000000000000000
1.58456e+29	- 0 11100000 00000000000000000000000
3.16913e+29	- 0 11100001 00000000000000000000000
6.33825e+29	- 0 11100010 00000000000000000000000
1.26765e+30	- 0 11100011 00000000000000000000000
2.5353e+30	- 0 11100100 00000000000000000000000
5.0706e+30	- 0 11100101 00000000000000000000000
1.01412e+31	- 0 11100110 00000000000000000000000
2.02824e+31	- 0 11100111 00000000000000000000000
4.05648e+31	- 0 11101000 00000000000000000000000
8.11296e+31	- 0 11101001 00000000000000000000000
1.62259e+32	- 0 11101010 00000000000000000000000
3.24519e+32	- 0 11101011 00000000000000000000000
6.49037e+32	- 0 11101100 00000000000000000000000
1.29807e+33	- 0 11101101 00000000000000000000000
2.59615e+33	- 0 11101110 00000000000000000000000
5.1923e+33	- 0 11101111 00000000000000000000000
1.03846e+34	- 0 11110000 00000000000000000000000
2.07692e+34	- 0 11110001 00000000000000000000000
4.15384e+34	- 0 11110010 00000000000000000000000
8.30767e+34	- 0 11110011 00000000000000000000000
1.66153e+35	- 0 11110100 00000000000000000000000
3.32307e+35	- 0 11110101 00000000000000000000000
6.64614e+35	- 0 11110110 00000000000000000000000
1.32923e+36	- 0 11110111 00000000000000000000000
2.65846e+36	- 0 11111000 00000000000000000000000
5.31691e+36	- 0 11111001 00000000000000000000000
1.06338e+37	- 0 11111010 00000000000000000000000
2.12676e+37	- 0 11111011 00000000000000000000000
4.25353e+37	- 0 11111100 00000000000000000000000
8.50706e+37	- 0 11111101 00000000000000000000000
1.70141e+38	**- 0 11111110 00000000000000000000000**

And now for even bigger!..

1	**- 0 01111111 00000000000000000000000**	8.88178e-16	- 0 01001101 00000000000000000000000	7.88861e-31	- 0 00011011 00000000000000000000000
0.5	- 0 01111110 00000000000000000000000	4.44089e-16	- 0 01001100 00000000000000000000000	3.9443e-31	- 0 00011010 00000000000000000000000
0.25	- 0 01111101 00000000000000000000000	2.22045e-16	- 0 01001011 00000000000000000000000	1.97215e-31	- 0 00011001 00000000000000000000000
0.125	- 0 01111100 00000000000000000000000	1.11022e-16	- 0 01001010 00000000000000000000000	9.86076e-32	- 0 00011000 00000000000000000000000
0.0625	- 0 01111011 00000000000000000000000	5.55112e-17	- 0 01001001 00000000000000000000000	4.93038e-32	- 0 00010111 00000000000000000000000
0.03125	- 0 01111010 00000000000000000000000	2.77556e-17	- 0 01001000 00000000000000000000000	2.46519e-32	- 0 00010110 00000000000000000000000
0.015625	- 0 01111001 00000000000000000000000	1.38778e-17	- 0 01000111 00000000000000000000000	1.2326e-32	- 0 00010101 00000000000000000000000
0.0078125	- 0 01111000 00000000000000000000000	6.93889e-18	- 0 01000110 00000000000000000000000	6.16298e-33	- 0 00010100 00000000000000000000000
0.00390625	- 0 01110111 00000000000000000000000	3.46945e-18	- 0 01000101 00000000000000000000000	3.08149e-33	- 0 00010011 00000000000000000000000
0.00195312	- 0 01110110 00000000000000000000000	1.73472e-18	- 0 01000100 00000000000000000000000	1.54074e-33	- 0 00010010 00000000000000000000000
0.000976562	- 0 01110101 00000000000000000000000	8.67362e-19	- 0 01000011 00000000000000000000000	7.70372e-34	- 0 00010001 00000000000000000000000
0.000488281	- 0 01110100 00000000000000000000000	4.33681e-19	- 0 01000010 00000000000000000000000	3.85186e-34	- 0 00010000 00000000000000000000000
0.000244141	- 0 01110011 00000000000000000000000	2.1684e-19	- 0 01000001 00000000000000000000000	1.92593e-34	- 0 00001111 00000000000000000000000
0.00012207	- 0 01110010 00000000000000000000000	1.0842e-19	- 0 01000000 00000000000000000000000	9.62965e-35	- 0 00001110 00000000000000000000000
6.10352e-05	- 0 01110001 00000000000000000000000	5.42101e-20	- 0 00111111 00000000000000000000000	4.81482e-35	- 0 00001101 00000000000000000000000
3.05176e-05	- 0 01110000 00000000000000000000000	2.71051e-20	- 0 00111110 00000000000000000000000	2.40741e-35	- 0 00001100 00000000000000000000000
1.52588e-05	- 0 01101111 00000000000000000000000	1.35525e-20	- 0 00111101 00000000000000000000000	1.20371e-35	- 0 00001011 00000000000000000000000
7.62939e-06	- 0 01101110 00000000000000000000000	6.77626e-21	- 0 00111100 00000000000000000000000	6.01853e-36	- 0 00001010 00000000000000000000000
3.8147e-06	- 0 01101101 00000000000000000000000	3.38813e-21	- 0 00111011 00000000000000000000000	3.00927e-36	- 0 00001001 00000000000000000000000
1.90735e-06	- 0 01101100 00000000000000000000000	1.69407e-21	- 0 00111010 00000000000000000000000	1.50463e-36	- 0 00001000 00000000000000000000000
9.53674e-07	- 0 01101011 00000000000000000000000	8.47033e-22	- 0 00111001 00000000000000000000000	7.52316e-37	- 0 00000111 00000000000000000000000
4.76837e-07	- 0 01101010 00000000000000000000000	4.23516e-22	- 0 00111000 00000000000000000000000	3.76158e-37	- 0 00000110 00000000000000000000000
2.38419e-07	- 0 01101001 00000000000000000000000	2.11758e-22	- 0 00110111 00000000000000000000000	1.88079e-37	- 0 00000101 00000000000000000000000
1.19209e-07	- 0 01101000 00000000000000000000000	1.05879e-22	- 0 00110110 00000000000000000000000	9.40395e-38	- 0 00000100 00000000000000000000000
5.96046e-08	- 0 01100111 00000000000000000000000	5.29396e-23	- 0 00110101 00000000000000000000000	4.70198e-38	- 0 00000011 00000000000000000000000
2.98023e-08	- 0 01100110 00000000000000000000000	2.64698e-23	- 0 00110100 00000000000000000000000	2.35099e-38	- 0 00000010 00000000000000000000000
1.49012e-08	- 0 01100101 00000000000000000000000	1.32349e-23	- 0 00110011 00000000000000000000000	1.17549e-38	- 0 00000001 00000000000000000000000
7.45058e-09	- 0 01100100 00000000000000000000000	6.61744e-24	- 0 00110010 00000000000000000000000	**5.87747e-39**	**- 0 00000000 10000000000000000000000**
3.72529e-09	- 0 01100011 00000000000000000000000	3.30872e-24	- 0 00110001 00000000000000000000000	**2.93874e-39**	**- 0 00000000 01000000000000000000000**
1.86265e-09	- 0 01100010 00000000000000000000000	1.65436e-24	- 0 00110000 00000000000000000000000	**1.46937e-39**	**- 0 00000000 00100000000000000000000**
9.31323e-10	- 0 01100001 00000000000000000000000	8.27181e-25	- 0 00101111 00000000000000000000000	**7.34684e-40**	**- 0 00000000 00010000000000000000000**
4.65661e-10	- 0 01100000 00000000000000000000000	4.1359e-25	- 0 00101110 00000000000000000000000	**3.67342e-40**	**- 0 00000000 00001000000000000000000**
2.32831e-10	- 0 01011111 00000000000000000000000	2.06795e-25	- 0 00101101 00000000000000000000000	**1.83671e-40**	**- 0 00000000 00000100000000000000000**
1.16415e-10	- 0 01011110 00000000000000000000000	1.03398e-25	- 0 00101100 00000000000000000000000	**9.18355e-41**	**- 0 00000000 00000010000000000000000**
5.82077e-11	- 0 01011101 00000000000000000000000	5.16988e-26	- 0 00101011 00000000000000000000000	**4.59177e-41**	**- 0 00000000 00000001000000000000000**
2.91038e-11	- 0 01011100 00000000000000000000000	2.58494e-26	- 0 00101010 00000000000000000000000	**2.29589e-41**	**- 0 00000000 00000000100000000000000**
1.45519e-11	- 0 01011011 00000000000000000000000	1.29247e-26	- 0 00101001 00000000000000000000000	**1.14794e-41**	**- 0 00000000 00000000010000000000000**
7.27596e-12	- 0 01011010 00000000000000000000000	6.46235e-27	- 0 00101000 00000000000000000000000	**5.73972e-42**	**- 0 00000000 00000000001000000000000**
3.63798e-12	- 0 01011001 00000000000000000000000	3.23117e-27	- 0 00100111 00000000000000000000000	**2.86986e-42**	**- 0 00000000 00000000000100000000000**
1.81899e-12	- 0 01011000 00000000000000000000000	1.61559e-27	- 0 00100110 00000000000000000000000	**1.43493e-42**	**- 0 00000000 00000000000010000000000**
9.09495e-13	- 0 01010111 00000000000000000000000	8.07794e-28	- 0 00100101 00000000000000000000000	**7.17465e-43**	**- 0 00000000 00000000000001000000000**
4.54747e-13	- 0 01010110 00000000000000000000000	4.03897e-28	- 0 00100100 00000000000000000000000	**3.58732e-43**	**- 0 00000000 00000000000000100000000**
2.27374e-13	- 0 01010101 00000000000000000000000	2.01948e-28	- 0 00100011 00000000000000000000000	**1.79366e-43**	**- 0 00000000 00000000000000010000000**
1.13687e-13	- 0 01010100 00000000000000000000000	1.00974e-28	- 0 00100010 00000000000000000000000	**8.96831e-44**	**- 0 00000000 00000000000000001000000**
5.68434e-14	- 0 01010011 00000000000000000000000	5.04871e-29	- 0 00100001 00000000000000000000000	**4.48416e-44**	**- 0 00000000 00000000000000000100000**
2.84217e-14	- 0 01010010 00000000000000000000000	2.52435e-29	- 0 00100000 00000000000000000000000	**2.24208e-44**	**- 0 00000000 00000000000000000010000**
1.42109e-14	- 0 01010001 00000000000000000000000	1.26218e-29	- 0 00011111 00000000000000000000000	**1.12104e-44**	**- 0 00000000 00000000000000000001000**
7.10543e-15	- 0 01010000 00000000000000000000000	6.31089e-30	- 0 00011110 00000000000000000000000	**5.60519e-45**	**- 0 00000000 00000000000000000000100**
3.55271e-15	- 0 01001111 00000000000000000000000	3.15544e-30	- 0 00011101 00000000000000000000000	**2.8026e-45**	**- 0 00000000 00000000000000000000010**
1.77636e-15	- 0 01001110 00000000000000000000000	1.57772e-30	- 0 00011100 00000000000000000000000	**1.4013e-45**	**- 0 00000000 00000000000000000000001**

Equality problem



Equality problem solution #1

```cpp
#include <iostream>
#include <cmath>

static const double delta = 1e-5;
static const double epsilon = 1e-5;

bool almost_equal(double x, double y){
        return std::abs(x-y) < epsilon;
}

int main(){
        for(double x = 2.; x < 3.; x += delta)
                if(almost_equal(5., x*x)){
                        std::cout << x << std::endl;
                        return 0;
                }
        return 1;
}
```

We can check grid determined by delta.

If $|5 - x^2| < epsilon$, then x is kind of solution

Let's say delta is the algorithm precision.
What's epsilon then?

How do we determine one?

For every big enough ***epsilon*** there is big enough ***y***
$y = x^2$, so $(x+delta)^2-x^2 > epsilon$ although $x < y < x+delta$.

Equality problem solution #2

```cpp
#include <iostream>
#include <cmath>

static const double delta = 1e-5;

bool change_sign(double x, double y){
        return x*y < 0;
}

int main(){
        for(double x = 2.; x < 3.; x += delta){
                double x_delta = x + delta;
                        if(change_sign(x_delta*x_delta - 5., x*x - 5.)){
                                std::cout << x << std::endl;
                                return 0;
                        }
                }
        return 1;
}
```

Still a grid.

But use sign change instead of ***epsilon***.

Now the answer has to be between ***x*** and ***x+delta***.

Unless ***y = 0***.

We can use properties such as continousnes or unimodality, but this limits the applicability.

Algebraic properties (that floats don't hold)

```
void assert_associativity()
{
        // associativity
        double a = 1.;
        double b = 9007199254740992.; // 2 ^ 53
        double c = -9007199254740992.;
        double d = (a + b) + c;
        double e = a + (b + c);
        assert(d != e);
}
```

Without asociativity, we can't have robust parallel algorithms.

```
void assert_identity()
{
        // identity element
        double a = 0.;
        double b = -0.;
        assert(a == b);
        // 0. != -0.
        auto a_string = std::to_string(a);
        auto b_string = std::to_string(b);
        assert(a_string != b_string);
}
```

No real null, just "small enough"

```
void assert_integers()
{
        // not a superset on integers
        uint64_t a = std::numeric_limits<uint64_t>::max();
        uint64_t b = std::numeric_limits<uint64_t>::max() - 1;
        double double_a = static_cast<double>(a);
        double double_b = static_cast<double>(b);
        assert(sizeof(a) == sizeof(double_a));
        assert(a == b + 1);
        assert(double_a == double_b); // lost one
}
```

No interchangeability.
(unless in a very limited subset)

Algebraic properties (that compilers rely on)

Floating point algebra reductions:									
a+b = b+a	x	-	x	x	x	x	-	-	x
a*b = b*a	x	-	x	x	x	x	-	-	x
a+b+c = a+(b+c)	x	-	x	x	-	-	-	-	-
(a+b)+c = a+(b+c)	-	-	x	x	-	-	-	-	-
a*b*c = a*(b*c)	x	-	-	x	-	-	-	-	-
a+b+c+d = (a+b)+(c+d)	-	-	-	x	-	-	-	-	-
a*b+a*c = a*(b+c)	x	-	-	-	x	-	-	-	x
a*x*x*x + b*x*x + c*x + d = ((a*x+b)*x+c)*x+d	x	-	x	x	x	-	-	-	-
$x*x*x*x*x*x*x*x = ((x^2)^2)^2$	-	-	-	x	-	-	-	-	-
a+a+a+a = a*4	x	-	-	x	x	-	-	-	-
-(-a) = a	-	-	x	x	x	x	x	x	-
a-(-b) = a+b	-	-	-	x	x	x	-	x	-
a+0 = a	x	-	x	x	x	x	x	x	-
a*0 = 0	-	-	x	x	x	x	-	x	x
a*1 = a	x	-	x	x	x	x	x	-	x
(-a)*(-b) = a*b	-	-	-	x	x	x	-	-	-
a/a = 1	-	-	-	-	-	-	-	-	x
a/1 = a	x	-	x	x	x	-	x	-	-
0/a = 0	-	-	-	x	x	-	-	x	x
(-a == -b) = (a == b)	-	-	-	x	x	-	-	-	-
(-a > -b) = (a < b)	-	-	-	x	x	-	-	-	x
Divide by constant = multiply by reciprocal	x	x	-	x	x	-	-	x	-

http://www.agner.org/optimize/optimizing_cpp.pdf

Integers

```cpp
#include <array>
#include <iostream>

template <typename T>
struct Box
{
        std::array<T, 3> m_side;
        double Volume()
        {
                return m_side[0]*m_side[1]*m_side[0];
        }
};

int main()
{
        Box<int> test_box;
        test_box.m_side = {100, 200, 300};
        std::cout << test_box.Volume() << std::endl;
        test_box.m_side = {1000, 2000, 3000};
        std::cout << test_box.Volume() << std::endl;
        test_box.m_side = {10000, 20000, 30000};
        std::cout << test_box.Volume() << std::endl;
}
```

```
./integer_overflow
2e+06
2e+09
-1.45476e+09
```

Not specified by the universal standard.
Hardly specified by C or C++ standard.
Overflows are eihter silent or undefined;
conversions are obscure.

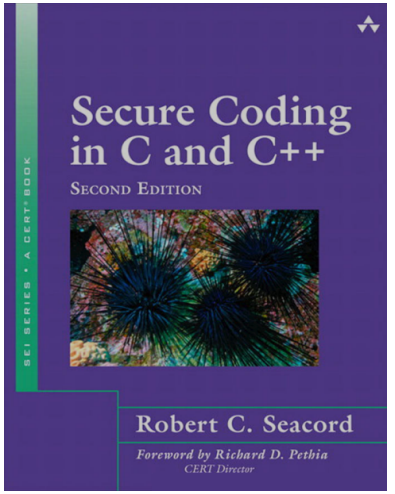
Integers are inherently unsafe

Things to know

- Integer Conversion Rank
- Integer Promotions
- Usual Arithmetic Conversions

Things to follow

- MSC14-C. Do not introduce unnecessary platform dependencies.
- DCL03-C. Use a static assertion to test the value of a
- constant expression.
- INT07-C. Use only explicitly signed or unsigned char type for numeric values.
- INT32-C. Ensure that operations on signed integers do not result in overflow.
- INT34-C. Do not shift a negative number of bits or more bits than exist in the operand.
- INT13-C. Use bitwise operators only on unsigned operands.
- INT01-C. Use rsize_t or size_t for all integer values representing the size of an object.
- and many more...



Algebraic properties (that compilers rely on too)

Integer algebra reductions:									
a+b = b+a	x	(x)	x	x	x	x	-	x	x
a*b = b*a	x	(x)	x	x	x	x	-	x	x
(a+b)+c = a+(b+c)	x	-	x	x	-	-	x	x	-
a+b+c = c+b+a	x	-	-	x	-	-	-	-	-
a+b+c+d = (a+b)+(c+d)	-	-	x	x	-	-	-	-	-
a*b+a*c = a*(b+c)	x	-	x	x	x	-	-	-	x
a*x*x*x + b*x*x + c*x + d = ((a*x+b)*x+c)*x+d	x	-	x	x	x	-	-	-	x
x*x*x*x*x*x*x*x = $((x^2)^2)^2$	-	-	x	-	-	-	-	-	-
a+a+a+a = a*4	x	-	x	x	-	-	-	-	x
-(-a) = a	x	-	x	x	x	x	x	x	-
a-(-b) = a+b	x	-	x	x	x	x	-	x	-
a-a = 0	x	-	x	x	x	x	x	x	x
a+0 = a	x	x	x	x	x	x	x	x	x
a*0 = 0	x	x	x	x	x	x	x	-	x
a*1 = a	x	x	x	x	x	x	x	x	x
(-a)*(-b) = a*b	x	-	x	x	x	-	-	-	-
a/a = 1	-	-	-	-	x	-	-	-	x
a/1 = a	x	x	x	x	x	x	x	x	x
0/a = 0	-	-	-	x	-	-	-	x	x
(-a == -b) = (a == b)	-	-	-	x	x	-	-	-	-
(a+c == b+c) = (a == b)	-	-	-	-	x	-	-	-	-
!(a < b) = (a >= b)	x	x	x	x	x	x	x	x	x
(a<b && b<c && a<c) = (a<b && b<c)	-	-	-	-	-	-	-	-	-
Multiply by constant = shift and add	x	x	x	x	-	x	x	x	-
Divide by constant = multiply and shift	x	-	x	x	x	(-)	x	-	-

Exercises

Validate or disprove algebraic properties compilers rely on.
Brute force example:

```
#include <iostream>
#include <limits>

int main()
{
	for(uint32_t i = 0u; i < std::numeric_limits<uint32_t>::max(); ++i)
	{
		const float& f = *(reinterpret_cast<float*>(&i));
		if(f != f)
			{
			std::cout << f << " != " << f << std::endl;
			return 0;
			}
	}
}
```