Числа с плавающей точкой

Методы представления действительных чисел

С плавающей запятой.	• Наиболее распространённый вариант.
С фиксированной запятой.	• Используется масштабирование чтобы получить фиксированное число разрядов после запятой, $12345 \rightarrow 123.45$.
Рациональные числа.	• Хранятся числитель и знаменатель дроби.
Логарифмические системы счисления.	• Записываются логарифм значения и его знак.
С плавающей запятой без ограничения точности.	• Реализуются программно для повышения точности.
Символьное представление.	• Системы компьютерной алгебры, например Mathematica, могут работать с такими числами как $\sqrt{\pi}$ напрямую.

Числа с фиксированной запятой

- Пусть единица целого числа соответствует $1/16$:
  - $0000\ 0001_2 = 1 \leftrightarrow \frac{1}{16} = 0.625$
  - $0001\ 1000_2 = 24 \leftrightarrow \frac{24}{16} = 1.5$
- Запятая находится в фиксированной позиции.
- Сложение такое же, как для целых чисел:

$$\frac{a}{c} + \frac{b}{c} = \frac{a+b}{c}$$

- Умножение требует масштабирования после выполнения операции

$$\frac{a}{c} * \frac{b}{c} = \frac{(a*b)/c}{c}$$

$$1.5 * 2.5 = 3.75$$

$$0001\ 1000_2 * 0010\ 1000_2 = 0011\ \ 1100\ |\ 0000_2$$

  - Деление на 2^n можно реализовать быстро.

- Используется:
  - Для ускорения вычислений в играх (сейчас бессмысленно).
  - В финансовых вычислениях для обеспечения точности.
  - Может быть реализован на основе двоичной системы, BCD или десятичной (текст).

Экспоненциальная запись

$$0.314 \times 10^1$$

$$\mp M \times \beta^E$$

- M – мантисса (англ. significand, mantissa)
- β – основание (англ. base)
- E – порядок (англ. exponent)

- Нормальная форма: $0 \leq M < 1$
- Нормализованная форма (англ. normalized, scientific notation): $1 \leq M < \beta$

Числа с плавающей точкой

- Числа с плавающей точкой – экспоненциальная запись в двоичной системе.

$$2.625 \leftrightarrow 1.0101_2 \times 2^1$$

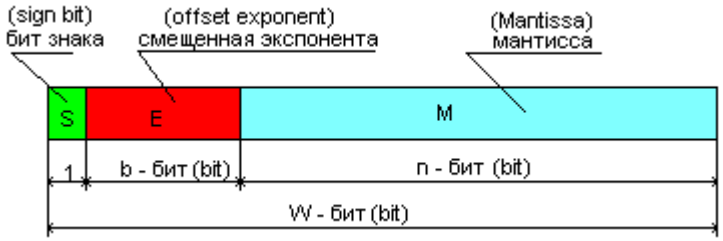
- В нормализованной форме старший разряд мантиссы всегда равен 1, его можно не хранить.
- Знак числа хранится в отдельном бите.
- Экспонента записывается в смещённом виде:

  $c = E + \beta^{b-1} - 1$, b – число бит для записи экспоненты.

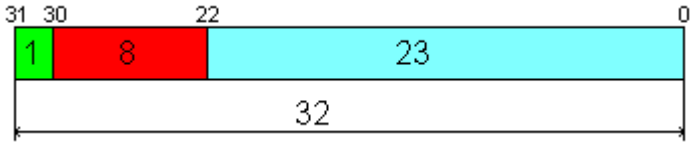
Экспонента	-127	…	-1	0	1	…	128
Код	00000000_2	…	01111110_2	01111111_2	10000000_2	…	11111111_2
Характеристика	0_{10}	…	126_{10}	127_{10}	128_{10}	…	255_{10}

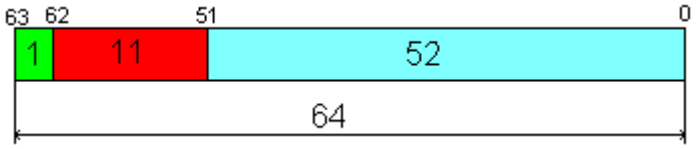
Обратное преобразование: $E = c - \beta^{b-1} + 1$.

Стандарт IEEE 754



Тип данных	Знак S	Поле экспоненты b	Поле мантиссы n	Всего W		Смещение экспоненты	Биты точности	Десятичные разряды
Half (IEEE 754-2008)	1	5	10	16		15	11	~3.3
Single	1	8	23	32		127	24	~7.2
Double	1	11	52	64		1023	53	~15.9
x86 extended precision	1	15	64	80		16383	64	~19.2
Quad	1	15	112	128		16383	113	~34.0





Стандарт IEE 754

- Для представления вещественных чисел используют следующий формат $v = (-1)^s \times 1.M \times 2^E$:

	Одинарная точность	Двойная точность
Знак s	1 бит	1 бит
Мантисса M	23 бит	52 бита
Экспонента (порядок) E	8 бит	11 бит
Точность (десятичные разряды)	≈ 7.2	≈ 15.9
Самое большое число	$\approx 3.404 \times 10^{38}$	$\approx 1.798 \times 10^{308}$
Самое маленькое число	$\approx 1.175 \times 10^{-38}$	$\approx 2.225 \times 10^{-308}$
Самое маленькое денормализованное число	$\approx 1.4 \times 10^{-45}$	$\approx 5 \times 10^{-324}$

- Есть коды для специальных значений: $+\infty, -\infty, NaN$

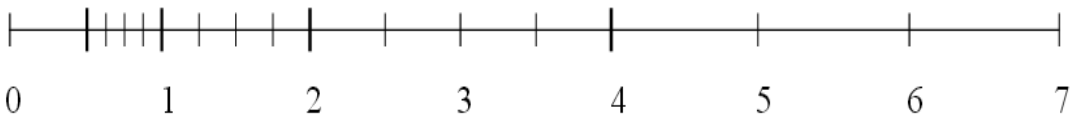
Числа с одинарной точностью IEEE 754

- Пределы нормализованных чисел:
  - самое большое: $1.11111\ldots1_2 \times 2^{127} \approx 3.404 \times 10^{38}$;
  - самое маленькое: $1.0000\ldots0_2 \times 2^{-126} \approx 1.175 \times 10^{-38}$.
- Можно представлять числа, меньшие 2^{-126}:
  - денормализованные числа хранят со значением 00_{16} в поле экспоненты.
  - скрытый бит считается равным 0.
  - значение $0000\ 0000_{16}$ мантиссы представляет 0.

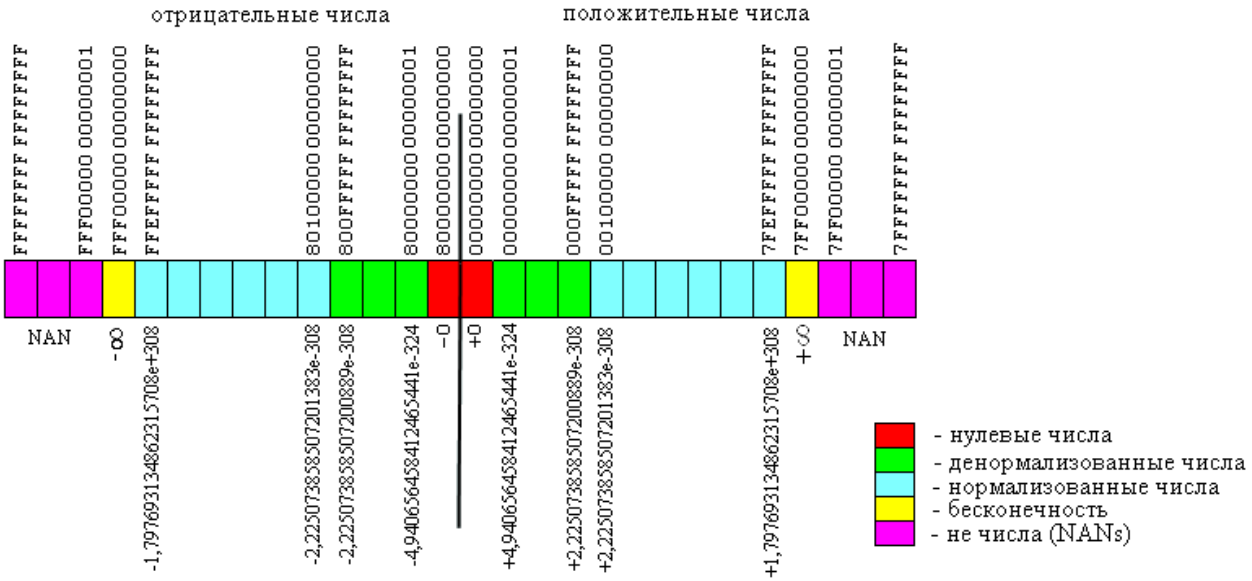
Экспонента	Мантисса	Значение	Описание
00_{16}	$= 0$	0	Ноль
00_{16}	$\neq 0$	$\mp 0.m \times 2^{-126}$	Денормализованные числа.
$01_{16} \ldots FE_{16}$		$\mp 1.m \times 2^{e-127}$	Нормализованные числа.
FF_{16}	$= 0$	$\mp\infty$	Бесконечности.
FF_{16}	$\neq 0$	NaN	«Не числа.»

Какие числа можно представить в виде числа с плавающей точкой

- Нормализованные числа для $\beta = 2, n = 3,\ b = 2,$ $E_{min} = -1,\ E_{max} = 2$.



- Диапазон чисел формата двойной точности.



Преобразование числа в формат с плавающей точкой

Алгоритм преобразования:

1. Запишите число в двоичной системе счисления.
    1. Целая часть – как обычно.
    2. Дробная часть:
        1. Взять дробную часть числа, умножить на 2.
        2. Выдать в качестве следующего бита целую часть результата.
        3. Если дробная часть результата $\neq 0$ перейти к а.
2. Допишите $\times 2^0$.
3. Нормализуйте число, сдвинув запятую в позицию после первой 1 и скорректировав экспоненту.
4. Вычислите характеристику $c = E + 2^{b-1} - 1$.
5. Запишите знак числа и характеристику в отведённые им битовые поля.
6. Запишите часть мантиссы, находящуюся после запятой, в отведённое ей поле.

Режимы округления

- Округление используется в случаях, когда число бит результата операции превышает доступное для хранения.

- Round To Nearest, Ties to Even – округление в сторону ближайшего, к чётному в случае конфликтов.
  - Режим по умолчанию в IEEE 754.

  $01.1\mathbf{01}_2 \to 01.1_2, 01.1\mathbf{10}_2 \to 10.0_2, 01.1\mathbf{11}_2 \to 10.0_2, -01.1\mathbf{01}_2 \to -01.1_2$
- Round Up - округление в сторону $+\infty$;

  $01.1\mathbf{01}_2 \to 10.0_2, 01.1\mathbf{10}_2 \to 10.0_2, 01.1\mathbf{11}_2 \to 10.0_2, -01.1\mathbf{01}_2 \to -01.1_2$
- Round Down – округление в сторону $-\infty$;

  $01.1\mathbf{01}_2 \to 01.1_2, 01.1\mathbf{10}_2 \to 01.1_2, 01.1\mathbf{11}_2 \to 01.1_2, -01.1\mathbf{01}_2 \to -10.0_2$
- Round Towards Zero – округление в сторону 0;
  - Для реализации достаточно отбросить «лишние» биты.

  $01.1\mathbf{01}_2 \to 01.1_2, 01.1\mathbf{10}_2 \to 01.1_2, 01.1\mathbf{11}_2 \to 01.1_2, -01.1\mathbf{01}_2 \to -01.1_2$

Сложение чисел с плавающей точкой

Пусть $f_0 = M_0 \times 2^{E_0}$, $f_1 = M_1 \times 2^{E_1}$, $E_0 \geq E_1$.

Тогда $f_0 + f_1 = (M_0 + M_1 * 2^{E_1 - E_0}) \times 2^{E_0}$.

- Алгоритм сложения:
  1. Сдвинуть точку в меньшем числе, чтобы совпали экспоненты.
  2. Сложить (вычесть) мантиссы.
  3. Нормализовать сумму.
  4. Проверить переполнение.
  5. Округлить до доступного числа бит.
  6. Нормализовать в случае необходимости.

Умножение чисел с плавающей точкой

Пусть $f_0 = M_0 \times 2^{E_0}$, $f_1 = M_1 \times 2^{E_1}$.

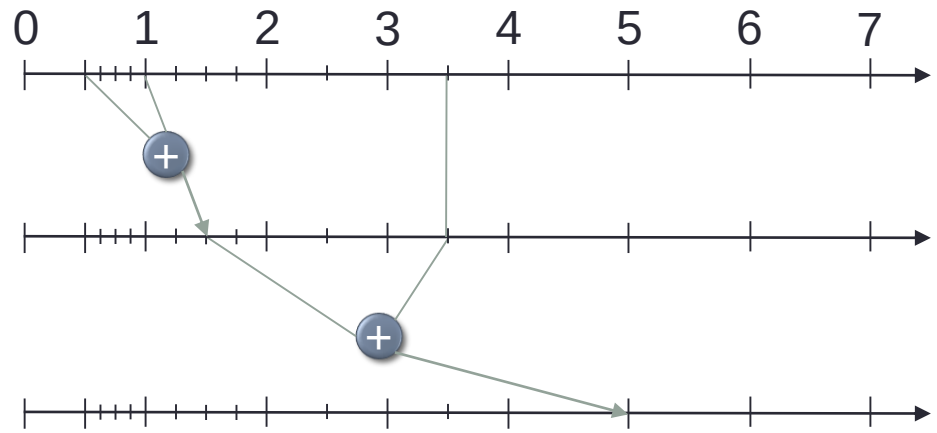
Тогда $f_0 * f_1 = (M_0 * M_1) \times 2^{E_0 + E_1}$.

- Алгоритм умножения:
  1. Сложить экспоненты.
  2. Умножить мантиссы и определить знак результата.
  3. Нормализовать результат.
  4. Проверить переполнение.
  5. Округлить до доступного числа бит.
  6. Нормализовать в случае необходимости.

Проблемы, вызванные точностью представления чисел

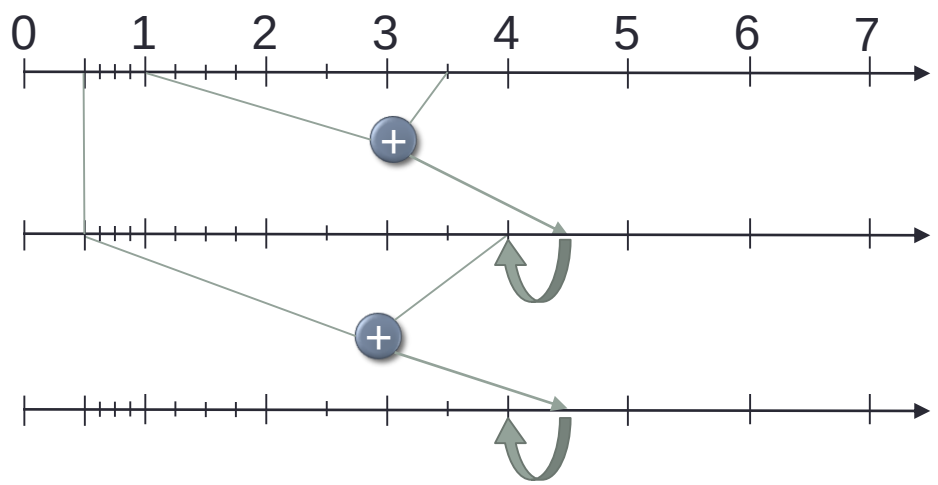
- Порядок вычислений может влиять на результат и его точность:
  - выполняется коммутативность: $a + b = b + a$, $a * b = b * a$;
  - иногда не выполняется ассоциативность: $a + (b + c) \neq (a + b) + c$;

(0.5 + 1.0) + 3.5



(0.5 + 1.0) + 3.5 = 5

0.5 + (1.0 + 3.5)



0.5 + (1.0 + 3.5) = 4

- иногда не выполняется дистрибутивность: $a * (b + c) \neq a * b + b * c$;

Проблемы, вызванные точностью представления чисел

- Не все числа могут быть представлены.
  - Математика: $\tan\left(\frac{\pi}{2}\right) = \infty$, single: −22877332.0,double:16331239353195370.0.
  - $(single)12346789.123457 - (double)123456789.123457 = 2.87654320895672.$
- Преобразование в целые числа: $63.0/9.0 \to 7$, $0.63 \ / \ 0.09 \to 6$.
- Преобразование числа в текст и обратно происходит с округлением, многие числа нельзя ввести точно (вводить лучше в экспоненциальном формате).
- Ограниченный размер экспоненты может привести к переполнению или атни-переполнению.
- Затруднено сравнение на равенство.
- Затруднена проверка безопасности деления: делитель, не равный нулю, может привести к бесконечному результату.

Вычисление приближенного значения числа π методом Архимеда

$$t_0 = 1/\sqrt{3},$$

$$t_{i+1} = \left(\sqrt{t_i^2 + 1} - 1\right)/t_i \qquad t_{i+1} = t_i/\left(\sqrt{t_i^2 + 1} + 1\right)$$

0	3.4641016151377543863	3.4641016151377543863
1	3.2153903091734710173	3.2153903091734723496
2	3.1596599420974940120	3.1596599420975006733
3	3.1460862151314012979	3.1460862151314352708
	…	
11	3.1415927256228504127	3.1415927220386148377
12	3.1415926717412858693	3.1415926707019992125
13	3.1415926189011456060	3.1415926578678454728
14	3.1415926717412858693	3.1415926546593073709
15	3.1415919358822321783	3.1415926538571730119
16	3.1415926717412858693	3.1415926536566394222
17	3.1415810075796233302	3.1415926536065061913
18	3.1415926717412858693	3.1415926535939728836
19	3.1414061547378810956	3.1415926535908393901
20	3.1405434924008406305	3.1415926535900560168
	…	
25		3.1415926535897962246
26		3.1415926535897962246
27		3.1415926535897962246
28		3.1415926535897962246
	Истинное значение	3.1415926535897932384626433832 7...

Проблемы на практике

25 февраля 1991 г., 28 солдат США погибли от иракской ракеты, пропущенной системой Patriot

- В следствии потери точности при расчётах времени за 100 часов дежурства системные часы ушли на 0.34 секунды, что дало ошибку определения цели в 600 метров.

4 июня 1996 г., взрыв ракетоносителя Ariane

- Из-за исключения при переполнении в двух бортовых компьютерах ракета потеряла управление и была взорвана системой самоуничтожения.

30 декабря 2010 г. Ошибка №53632 PHP

- Попытка вывода числа 2.2250738585072011e-308, находящегося на границе нормализованных и ненормализованных чисел вызывает зависание процесса веб-сервера со 100% загрузкой ЦПУ.

Алгоритм Кэхэна

- При сложении большого количества слагаемых возможны значительные погрешности, если сумма станет намного больше слагаемых (рост ошибки может достигать $O(n)$) .
- Циклическая яма – сумма становится так велика, что дальнейшее добавление слагаемых никогда её не изменяет.
- Алгоритм Кэхэна позволяет вычислить сумму с погрешностью, ограниченной только точностью представления чисел ($O(1)$).

```
function KahanSum(input):
  sum = 0
  c = 0
  for i in input:
    y = i - c
    t = sum + y           # При сложении теряются младшие биты суммы.
    c = (t - sum) - y     # (t - sum) - та часть, которую удалось
                          # добавить.
                          # -((t - sum) - y) - «потерянная часть»
    sum = t
    # «Потерянная часть» будет добавлена на следующем шаге.
  return sum
```

Алгоритм Кэхэна – пример.

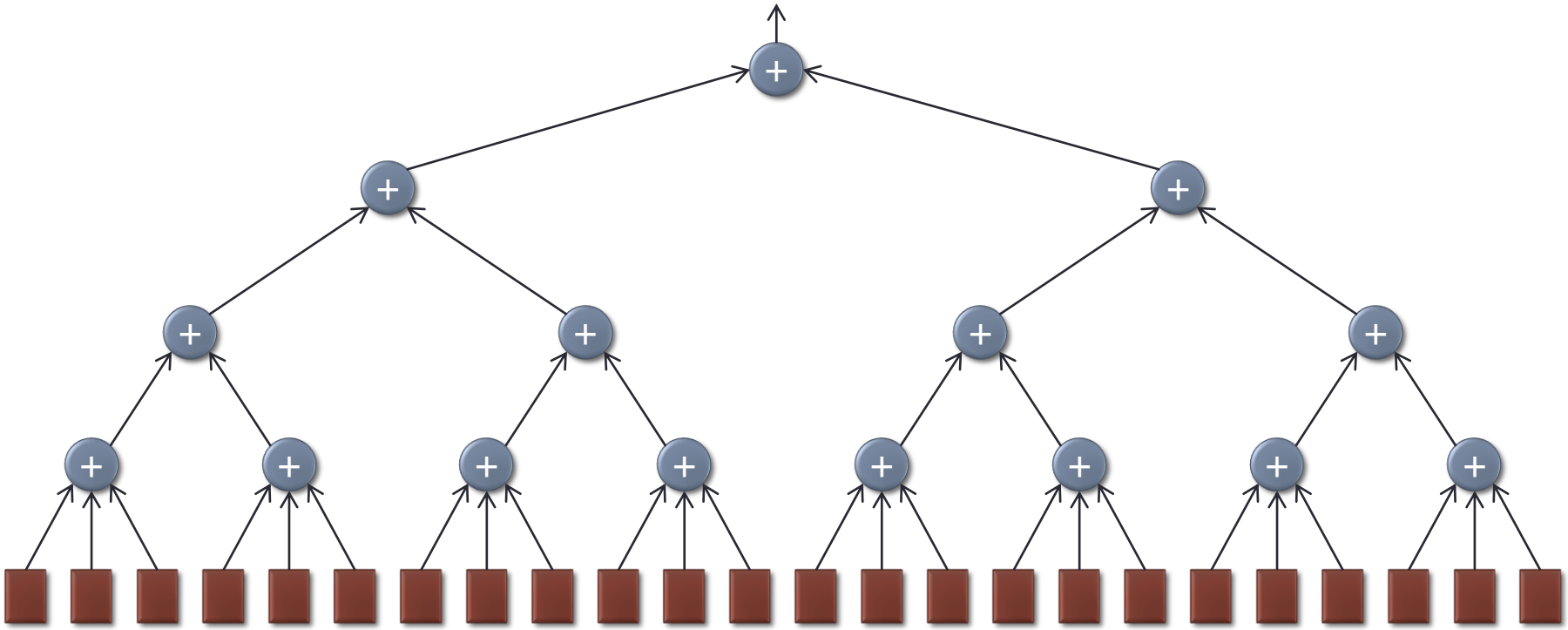
- Сложение $10000.0 + 3.14159 + 2.71828$ используя арифметику с числами с 6 десятичными разрядами.
- Точный ответ: 10005.85987, округлённо $\cong 10005.9$.
- Обычное сложение: $(10000.0 + 3.14159) + 2.71828 =$
  $= 10003.1 + 2.71828 = 10005.8$.
- Алгоритм Кэхэна:

Шаг 1	Шаг 2

`c = 0`
`y = 3.14159 – 0`
`t = 10000.0 + 3.14159 ≅ 10003.1`
`c = (10003.1 - 10000.0) - 3.14159`
`  = 3.10000 - 3.14159 = -.0415900`
`sum = 10003.1`
	`y = 2.71828 - -.0415900 = 2.75987`
	`t = 10003.1 + 2.75987 =`
	`  = 10005.85987 ≅ 10005.9`
	`c = (10005.9 - 10003.1) - 2.75987`
	`  = 2.80000 - 2.75987 = .040130`
	`sum = 10005.9`

Альтернатива – попарное сложение

- Рост ошибки $O(\log n)$.
- Не увеличивает число операций, может выполняться параллельно.



Как следует работать с числами с плавающей точкой

- Проводите вычисления, используя один тип данных.
- Избегайте «лишних» преобразований данных.
- При сравнении чисел с плавающей точкой вместо $x == y$ используйте $abs(x - y) \leq \epsilon$.
- Проверяйте результаты floor(), ceil() и приведения к целому типу.
- Избегайте сложения (вычитания) чисел, экспоненты которых сильно отличаются.
  - При попадании в циклическую дыру используйте алгоритм Кэхэна или аналоги.
- Избегайте вычитания близких чисел, полученных в ходе предварительных вычислений.
  - Попробуйте переписать выражение, чтобы избежать вычитания.
    - Корни квадратного уравнения $ax^2 + bx + c = 0$:

$$b < 0\text{: } r_1 = -\frac{b + \sqrt{b^2 - 4ac}}{2a}, r_2 = \frac{2c}{-b + \sqrt{b^2 - 4ac}};$$

$$b \gg ac\text{: } r_1 = \frac{2c}{-b - \sqrt{b^2 - 4ac}}, r_2 = -\frac{b - \sqrt{b^2 - 4ac}}{2a}.$$

    - Вместо $x^2 - y^2$ лучше использовать $(x - y)(x + y)$.
- С функциями exp(), log(), sin(),… тоже есть свои особенности.
- Прочитайте David Golberg, What Every Computer Scientist Should Know About Floating-Point Arithmetic – 94 страницы.