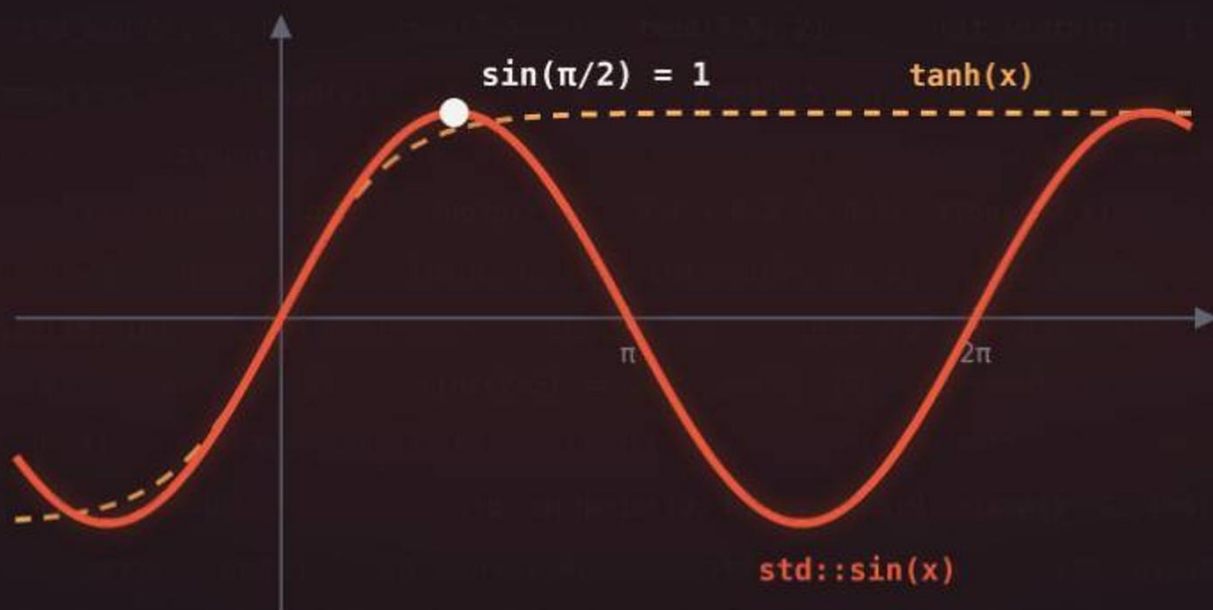


— УЧЕБНИК С ЗАДАЧАМИ И ОТВЕТАМИ

C++

и математика без соплей



Все математические функции C++ и C:
от классики до C++23 — с примерами,
шутками и задачами

Алексей Примаков

СОДЕРЖИТ
НЕЦЕНЗУРНУЮ
БРАНЬ

18+

C++ · C++23 · C

Алексей Примак

**С++/С и математика
без соплей код**

«Автор»

2026

Примак А.

С++/С и математика без соплей код / А. Примак — «Автор»,
2026

Учебник, после которого математика в С++ перестаёт быть страшной. Десять глав проводят по всем математическим функциям языка: от банального `abs` до гамма-функции, от округления до случайных чисел. Каждая тема разобрана на живых примерах кода, с ловушками, на которых спотыкаются даже опытные программисты: почему $5 / 2$ равно 2, почему $0.1 + 0.2$ не равно 0.3 и почему $\sin(90)$ — это совсем не единица. Каждая глава показывает тему с трёх сторон: классический С++, современный С++20/23 (`format`, `print`, `ranges`, концепты, `constexpr`) и чистый С. В конце глав — задачи, в конце книги — ответы, шпаргалка и таблица «С против С++». Весь код проверен компилятором. Объясняю просто, шучу много и иногда ругаюсь — потому что баги того заслуживают. Книга подойдёт студентам, начинающим программистам и всем, кто хочет наконец разобраться, что творится внутри `cmath`.

© Примак А., 2026

© Автор, 2026

Содержание

Предисловие: какого хрена мы тут собрались	6
Что нужно	7
Шаблон каждой программы	8
Если ты пишешь на C	9
Что такое «современный C++»	10
Три правила, которые спасут тебе жопу	11
Глава 1. Арифметика, или где тебя наебёт компьютер	12
1.1. Модуль: abs и fabs	13
1.2. Деление: целое против дробного	14
1.3. Остаток: %, fmod, remainder	15
1.4. Переполнение	16
1.5. Современный C++: честные сравнения и проверка диапазона	17
1.6. На чистом C	18
Задачи к главе 1	19
Глава 2. Степени и корни: pow, sqrt, cbrt, hypot	20
2.1. pow — возведение в степень	21
2.2. sqrt и cbrt — корни	22
2.3. hypot — гипотенуза без переполнения	23
Пример: расстояние между двумя точками	24
2.4. Современный C++: степень на этапе компиляции	25
2.5. На чистом C	26
Задачи к главе 2	27
Глава 3. Логарифмы и экспонента: log, exp и их родня	28
3.1. Что такое логарифм на пальцах	29
Конец ознакомительного фрагмента.	30

Алексей Примак

С++/С и математика без соплей код

С++ и математика без соплей

Все математические функции С++ и С: от классики до С++23, с примерами, шутками и задачами

Table of Contents

Предисловие: какого хрена мы тут собрались

Эта книга научит тебя всем математическим функциям стандартного C++ — от банального `abs` до гамма-функции, — и в конце каждой главы будут задачи, от которых не отвертишься.

Я буду шутить, иногда материться и разжёвывать так, чтобы дошло даже в четыре утра. Математика в программировании — это не страшно. Страшно, когда ты сравниваешь два `double` через `==` и потом три дня ищешь баг.

Что нужно

- Компилятор с поддержкой C++20: g++ 10+, clang 12+ или MSVC 2019+.
- Голова. Желательно своя.
- Терпение на задачи. Ответы в конце, но подглядывать — это как читать последнюю страницу детектива: можно, но ты лох.

Шаблон каждой программы

Почти все функции живут в заголовке `<cmath>`. Целочисленные штуки — в `<numeric>`, константы — в `<numbers>`, рандом — в `<random>`.

```
#include <iostream>
#include <cmath>
#include <numeric>
#include <numbers>
#include <iomanip>

int main() {
    std::cout << std::fixed << std::setprecision(4);
    std::cout << std::sqrt(2.0) << '\n'; // 1.4142
}
```

Компилируем:

```
g++ -std=c++20 -Wall -Wextra main.cpp -o main && ./main
```

Флаги `-Wall -Wextra` — это твоя бесплатная бабушка, которая ворчит на каждую подозрительную строчку. Слушай бабушку.

Если ты пишешь на C

Почти все функции из этой книги есть и в чистом C: они живут в `<math.h>`. В конце каждой главы есть раздел «На чистом C», где показано, чем C отличается и что там приходится писать руками.

```
gcc -std=c2x -Wall -Wextra main.c -o main -lm && ./main
```

Флаг `-lm` подключает математическую библиотеку. Забудешь его — получишь ошибку линковки `undefined reference to sqrt` и полчаса недоумения. Флаг `-std=c2x` включает свежий стандарт C23 (в новых компиляторах можно писать `-std=c23`).

Что такое «современный C++»

Каждая глава заканчивается разделом «Современный C++»: там то же самое, но с фишками C++20 и C++23 — `std::format`, `std::print`, `ranges`, концепты, `constexpr`. Так сейчас пишут в нормальных командах. Для C++23 компилируй с флагом `-std=c++23` (g++ 14+ или clang 18+).

Три правила, которые спасут тебе жопу

Всегда пиши `std::` перед функцией. Без него `abs(-2.5)` может вызвать целочисленную версию из `C` и вернуть 2. Охренеть, да? Вот поэтому.

Тригонометрия работает в радианах, а не в градусах. Запомни это раз и навсегда.

`double` — это приближение. $0.1 + 0.2 \neq 0.3$, и это не баг, это физика двоичных дробей.

Глава 1. Арифметика, или где тебя наебёт компьютер

Главный вывод главы: $5 / 2$ в C++ равно 2, а остаток от деления отрицательного числа бывает отрицательным. Сейчас разберём почему.

1.1. Модуль: `abs` и `fabs`

Модуль — это «расстояние до нуля». Минус выкидывается, как бывший из жизни.

```
std::cout << std::abs(-7)      << '\n'; // 7 (int)
std::cout << std::abs(-7.5)    << '\n'; // 7.5 (double, из <cmath>)
std::cout << std::fabs(-7.5)  << '\n'; // 7.5 (всегда double)
std::cout << std::llabs(-9000000000LL) << '\n'; // для long long
```

Подвох: `std::abs(INT_MIN)` — неопределённое поведение. У `int` минимум `-2147483648`, а максимум `2147483647`. Положительной пары у минимума просто нет, и программа может сделать что угодно, хоть стихи напечатать.

1.2. Деление: целое против дробного

Если оба операнда целые — дробная часть отрезается нахрен. Не округляется, а именно отрезается к нулю.

```
std::cout << 7 / 2 << '\n'; // 3
std::cout << -7 / 2 << '\n'; // -3 (не -4!)
std::cout << 7.0 / 2 << '\n'; // 3.5
std::cout << static_cast<double>(7) / 2 << '\n'; // 3.5
```

Классическая ошибка новичка: `double avg = sum / count;`, где обе переменные `int`. Деление уже целое, а в `double` кладётся огрызок. Лечится приведением хотя бы одного операнда.

1.3. Остаток: %, fmod, remainder

Оператор % работает только с целыми. Знак результата совпадает со знаком делимого.

```
std::cout << 7 % 3 << '\n'; // 1
std::cout << -7 % 3 << '\n'; // -1 (сюрприз, сука)
```

Для дробных чисел — std::fmod (знак как у делимого) и std::remainder (ближайшее к нулю, может быть отрицательным):

```
std::cout << std::fmod(7.5, 2.0) << '\n'; // 1.5
std::cout << std::remainder(7.5, 2.0) << '\n'; // -0.5 (7.5 = 4*2 - 0.5)
```

Хочешь «математический» остаток, всегда от 0 до m-1? Пиши так:

```
int mod(int a, int m) { return ((a % m) + m) % m; }
// mod(-7, 3) == 2
```

Это пригодится для дней недели, кругового буфера и всего, что крутится по кругу.

1.4. Переполнение

`int` не резиновый. Сложи два больших — получишь отрицательное число, а для знаковых типов это вообще неопределённое поведение.

```
int a = 2'000'000'000;  
int b = a + a;           // UB, на практике мусор  
long long c = 1LL * a + a; // 4000000000, всё ок
```

Правило: если результат может быть большим — считай в `long long` с самого начала, а не после.

1.5. Современный C++: честные сравнения и проверка диапазона

Сравнение знакового и беззнакового числа — мина под ногами. Перед сравнением `-1` молча превращается в `4294967295`, и `-1 < 1u` оказывается ложью. C++20 даёт функции, которые сравнивают по-человечески:

```
#include <utility>
#include <cstdint>

int a = -1;
unsigned b = 1;
std::cout << (a < b) << '\n';           // 0 – минус один стал огромным
std::cout << std::cmp_less(a, b) << '\n'; // 1 – C++20, честное сравнение
std::cout << std::in_range<std::uint8_t>(300) << '\n'; // 0 – в байт не влезет
```

Правило: если в сравнении встретились `int` и `unsigned` (например, `i < v.size()`), либо приводи типы явно, либо используй `std::cmp_less`.

1.6. На чистом C

В C нет std:: и нет перегрузки функций. Поэтому у модуля четыре разных имени: abs для int, labs для long, llabs для long long (все три в <stdlib.h>) и fabs для double (в <math.h>). Перепутаешь — компилятор молча обрежет дробь.

Зато в C23 появился заголовок <stdckdint.h> с проверкой переполнения (нужен gcc 14+ или clang 18+). Функция возвращает true, если результат не влез:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <stdckdint.h> // C23

int main(void) {
    printf("%d %.1f\n", abs(-7), fabs(-7.5)); // 7 7.5
    printf("%d\n", -7 % 3);                  // -1, как и в C++

    int r;
    if (ckd_add(&r, 2000000000, 2000000000))
        puts("Переполнение! Хорошо, что проверили.");
    return 0;
}
```

Задачи к главе 1

Напиши программу, которая читает два целых числа и выводит их среднее арифметическое как дробное. Проверь на 3 и 4 — должно быть 3.5, а не 3.

Дано количество секунд (например, 100000). Выведи его в формате дни:часы:минуты:секунды, используя только / и %.

Сегодня среда (день 2, если понедельник — 0). Какой день недели будет через n дней и был n дней назад? Используй функцию `mod` из раздела 1.3, проверь на $n = 100$.

Почему `std::abs(-2.7)` без `#include <cmath>` может вернуть 2? Объясни своими словами и проверь на своём компиляторе.

Посчитай сумму чисел от 1 до 100000 в `int` и в `long long`. Сравни результаты и объясни разницу.

Глава 2. Степени и корни: `pow`, `sqrt`, `cbrt`, `hypot`

Главный вывод: `pow` — универсальный, но медленный и капризный; для квадрата пиши `x * x`, для корня — `sqrt`, для длины вектора — `hypot`.

2.1. pow — возведение в степень

В C++ нет оператора $^$ для степени. $2 \wedge 3$ — это побитовое XOR, и оно равно 1. Сколько людей на этом погорело — не сосчитать.

```
std::cout << std::pow(2.0, 10) << '\n'; // 1024
std::cout << std::pow(9.0, 0.5) << '\n'; // 3 (корень)
std::cout << std::pow(2.0, -1) << '\n'; // 0.5
std::cout << std::pow(-8.0, 1.0/3) << '\n'; // nan! отрицательное в дробную степень
```

Подвох номер два: pow возвращает double. (int)std::pow(10, 2) на некоторых старых компиляторах давал 99, потому что получалось 99.9999999. Для целых степеней пиши свою функцию:

```
long long ipow(long long base, int exp) {
    long long result = 1;
    while (exp > 0) {
        if (exp & 1) result *= base; // нечётная степень — домножаем
        base *= base;                // возводим основание в квадрат
        exp >>= 1;                   // делим степень пополам
    }
    return result;
}
// ipow(3, 13) == 1594323, всего ~4 умножения вместо 13
```

Это называется быстрое возведение в степень. Выучи его — на собеседованиях спрашивают постоянно.

2.2. sqrt и cbrt — корни

```
std::cout << std::sqrt(16.0) << '\n'; // 4
std::cout << std::sqrt(2.0) << '\n'; // 1.41421
std::cout << std::sqrt(-1.0) << '\n'; // nan — мнимых чисел тут нет
std::cout << std::cbrt(-27.0) << '\n'; // -3 — кубический корень из минуса можно!
```

Запомни: cbrt спокойно жрёт отрицательные числа, а pow(x, 1.0/3) — нет. Для кубического корня всегда cbrt.

Проверка «является ли число полным квадратом» — тоже ловушка из-за погрешности:

```
bool isSquare(long long n) {
    long long r = std::llround(std::sqrt((double)n));
    return r * r == n; // проверяем целыми, а не дробными!
}
```

2.3. hypot — гипотенуза без переполнения

Теорема Пифагора в одну функцию:

```
std::cout << std::hypot(3.0, 4.0) << '\n'; // 5
std::cout << std::hypot(1.0, 2.0, 2.0) << '\n'; // 3 (C++17, трёхмерная)
```

Почему не `sqrt(a*a + b*b)`? Потому что при $a = 1e200$ квадрат улетает в бесконечность, а `hypot` считает аккуратно. Плюс читается красивее.

Пример: расстояние между двумя точками

```
struct Point { double x, y; };  
  
double dist(Point a, Point b) {  
    return std::hypot(b.x - a.x, b.y - a.y);  
}  
// dist({0,0}, {6,8}) == 10
```

2.4. Современный C++: степень на этапе компиляции

Функцию `ipow` можно сделать `constexpr` — тогда компилятор посчитает её ещё до запуска программы. А концепт `std::integral` не даст сунуть туда `double`:

```
#include <concepts>

template <std::integral T>
constexpr T ipow(T base, unsigned exp) {
    T result = 1;
    while (exp > 0) {
        if (exp & 1) result *= base;
        exp >>= 1;
        if (exp > 0) base *= base;    // не возводим лишний раз — меньше риск переполнения
    }
    return result;
}

static_assert(ipow(2, 10) == 1024);    // проверено компилятором, а не тобой
constexpr long long big = ipow(3LL, 30); // 205891132094649, посчитано заранее
// ipow(2.5, 3); // ошибка компиляции: концепт integral не пустит double
```

`static_assert` — это тест, который выполняется при компиляции. Сломал функцию — программа даже не соберётся. В C++26 обещают сделать `constexpr` и большую часть `<cmath>`, включая `sqrt` и `pow`.

2.5. На чистом C

Без перегрузки у каждой функции три версии: `sqrtf` для `float`, `sqrt` для `double`, `sqrtl` для `long double`. Чтобы не помнить суффиксы, есть `<tgmath.h>` — «перегрузка для бедных»: макросы сами выбирают нужную версию по типу аргумента.

```
#include <stdio.h>
#include <tgmath.h> // включает <math.h> и добавляет типовые макросы

int main(void) {
    float f = 2.0f;
    printf("%f\n", sqrtf(f));           // явно float-версия
    printf("%f\n", sqrt(f));           // с <tgmath.h> сам выберет sqrtf
    printf("%f\n", hypot(3.0, 4.0));    // 5.000000
    printf("%f\n", cbrt(-27.0));        // -3.000000
    printf("%f\n", pow(2.0, 10));       // 1024.000000, оператора ^ для степени нет и тут
    return 0;
}
```

Задачи к главе 2

Выведи 2^3 и `std::pow(2, 3)`. Объясни, почему результаты разные.

Реализуй `ipow` и посчитай 2 в 62-й степени. Сравни с `std::pow(2.0, 62)` — совпадает ли до последней цифры?

Решение квадратного уравнения: по a , b , c выведи корни через дискриминант. Обработай три случая: два корня, один, ни одного.

Даны три точки на плоскости. Найди периметр треугольника с помощью `hypot`.

Выведи все полные квадраты от 1 до 1000 с помощью функции `isSquare`. Сколько их?

Глава 3. Логарифмы и экспонента: `log`, `exp` и их родня

Главный вывод: логарифм отвечает на вопрос «в какую степень возвести основание, чтобы получить число», а `log` в C++ — это натуральный логарифм, а не десятичный.

3.1. Что такое логарифм на пальцах

Если $2^{10} = 1024$, то логарифм 1024 по основанию 2 равен 10. Всё. Логарифм — это «сколько раз надо умножить», вывернутое наизнанку.

На практике логарифм считает «сколько шагов». Сколько раз можно поделить 1000000 пополам? $\log_2(1000000) \approx 20$. Вот почему бинарный поиск такой быстрый.

Конец ознакомительного фрагмента.

Текст предоставлен ООО «Литрес».

Прочитайте эту книгу целиком, [купив полную легальную версию](#) на Литрес.

Безопасно оплатить книгу можно банковской картой Visa, MasterCard, Maestro, со счета мобильного телефона, с платежного терминала, в салоне МТС или Связной, через PayPal, WebMoney, Яндекс.Деньги, QIWI Кошелек, бонусными картами или другим удобным Вам способом.