

Операции с целыми числами в информатике: от арифметики к программам

Целые числа — основа вычислений в памяти компьютера и «строительный материал» программ.



Почему целые числа важны в информатике



В отличие от школьной арифметики, компьютер хранит целые числа в ограниченной памяти и «уважает» строгие правила разрядности. Диапазон задаётся числом бит: ошибка понимания диапазона способна вызвать переполнение, неверный результат или сбой программы. Особенно часто целые числа нужны для счёта, сравнений, поиска максимумов/минимумов и организации циклов.

Понятие целого числа и его место среди типов данных

I

Целое число — это значение без дробной части: отрицательное, ноль или положительное. В информатике типы данных разделяют по тому, какие значения и представления они допускают: целые, вещественные, символьные, логические и строковые.

2

Целые типы выбирают, когда нужна точность без десятичных знаков: количество элементов, номер строки, число шагов, возраст в полных годах, количество ошибок. Здесь дробность не имеет смысла и только усложняет вычисления и проверку условий.

3

В языках программирования целые типы различаются размером и допустимым диапазоном, например `byte`, `short`, `int`, `long`. Этот выбор напрямую определяет, сколько памяти займёт число и какие значения будут корректными, а какие приведут к переполнению.

Двоичное представление целых чисел



Биты 0 и 1

Компьютер хранит целые числа в двоичном виде: используются только цифры 0 и 1, а каждый бит вносит вклад в итоговое значение.



Вес каждой позиции

Каждая двоичная позиция соответствует степени числа 2: поэтому итог зависит от расположения единиц и общего набора битов.



Разрядность задаёт диапазон

Количество доступных значений определяется разрядностью — числом бит, отведённых под число (например, 8, 16, 32 или 64).



Знак и дополнительный код

Для знаковых чисел знак часто учитывается специальным способом; отрицательные значения обычно представляются в дополнительном коде, что упрощает операции.

Диапазон целых чисел при разной разрядности

Увеличение разрядности расширяет диапазон не «на фиксированную величину», а примерно в 2 раза: растёт число возможных битовых комбинаций.

График показывает: при переходе к следующему типу по ширине (например, от 8 к 16 бит) доступные значения резко увеличиваются, что снижает риск переполнения.

Обобщение школьного курса информатики: представление целых чисел и диапазоны для 8/16/32/64 бит (знаковые и беззнаковые варианты).



Сложение целых чисел: правило и примеры

I

В математике сложение подчиняется тем же законам, а в компьютере оно реализуется через работу с битами. Если результат остаётся в допустимом диапазоне типа, вычисление проходит корректно и предсказуемо.

2

Когда знаки складываемых чисел различаются, компьютер по сути сравнивает их модули и определяет знак результата по большему по модулю слагаемому. Поэтому логика сложения остаётся связанной со знаками чисел.

3

Примеры: $7 + 5 = 12$, $-4 + 9 = 5$, $-6 + (-3) = -9$. В программах сложение применяют для накопления суммы, подсчёта элементов и обновления счётчиков в циклах.

Вычитание и связь с отрицательными числами

I

Вычитание удобно рассматривать через прибавление противоположного числа: $a - b = a + (-b)$. Это правило помогает понимать работу алгоритмов и объясняет, почему отрицательные значения играют ключевую роль в вычислениях.

2

Примеры: $10 - 3 = 7$, $5 - 8 = -3$, $-2 - 6 = -8$. В вычислительных системах вычитание строится на представлениях отрицательных чисел и операциях над их двоичными кодами.

3

Ошибки чаще возникают не из-за «математики», а из-за знака результата и порядка действий в программе. Если неверно учесть правила выражений, можно получить другой ответ или «сломать» логику условия.

Примеры вычислений с целыми числами

Выражение	Правило	Результат	Типичный смысл
$12 + 8$	сложение	20	сумма/итог
$-15 + 6$	сложение с разными знаками	-9	изменение счёта
$9 - 14$	вычитание	-5	разница/дефицит
$-7 - 5$	вычитание отрицательного	-12	убыток/уменьшение
3×4	умножение	12	масштаб
-3×4	умножение с разными знаками	-12	направление/коррекция
$20 \div 5$	целочисленное деление	4	количество групп
$-18 \div 3$	целочисленное деление	-6	счёт со знаком

В таблице показаны выражения, используемые арифметические правила и типичный смысл результата в программе.

Знак результата зависит от знаков операндов и вида операции; при программной реализации особенно важно отличать целочисленное деление от деления с остатком и от вещественного деления.

Школьный курс информатики: арифметические операции над целыми числами.

Умножение целых чисел и правило знаков



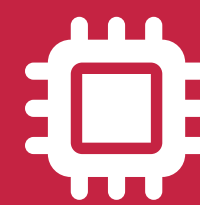
Правило знаков

При умножении знак результата определяется знаками множителей: одинаковые знаки дают положительный результат, разные — отрицательный. Это помогает быстро проверять ответы ещё до вычислений на экране.



Типовые применения

В информатике умножение встречается при масштабировании значений, вычислении площади, повторяющихся сдвигах и при формировании адресов в структуре данных, где важны целые величины.



Стоимость операции

На практике умножение может требовать больше вычислительных ресурсов, чем сложение и вычитание. Поэтому в некоторых алгоритмах используют более простые эквивалентные преобразования, сохраняя корректность.

$17 \div 5 = 3, \text{ остаток } 2$

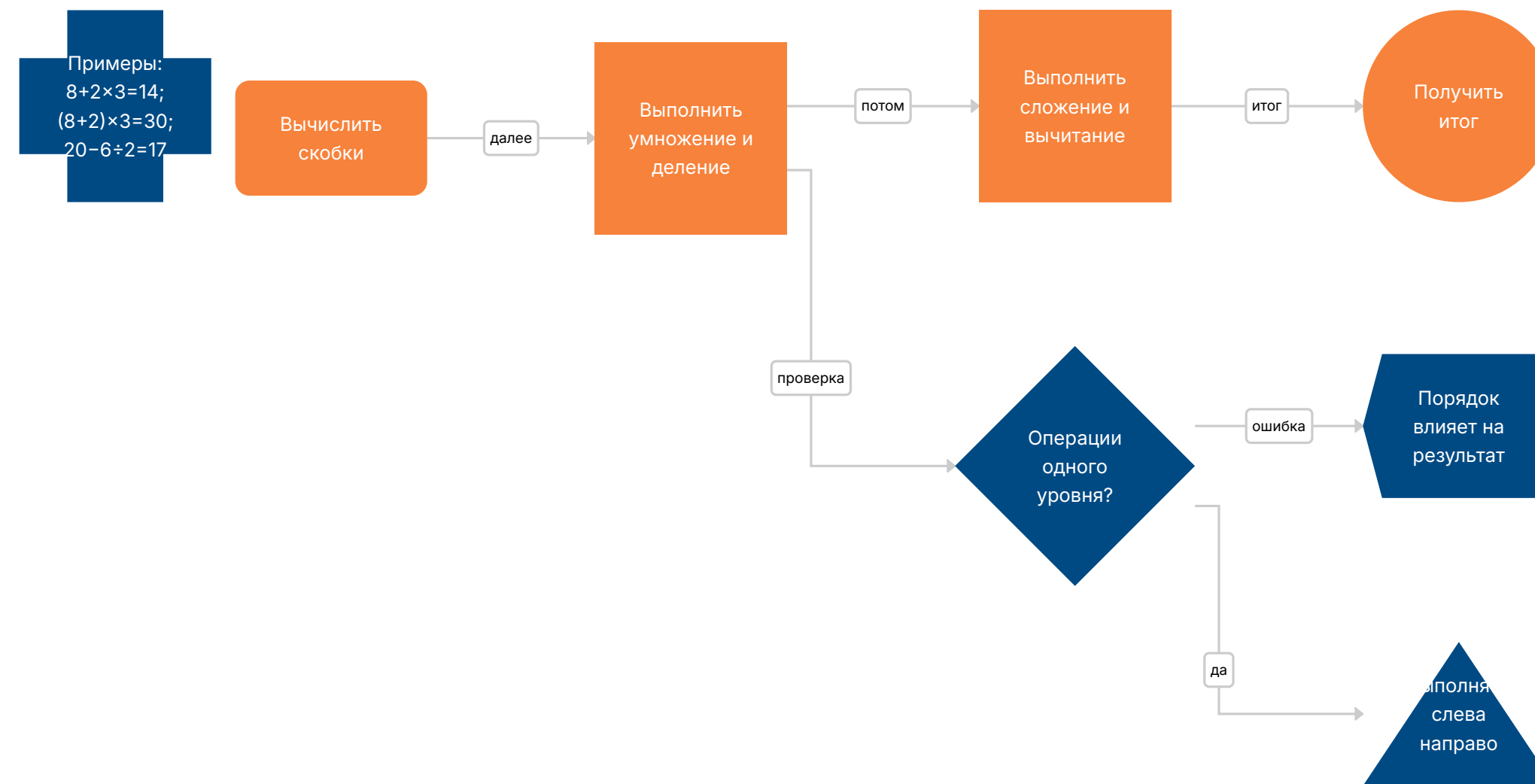
Остаток показывает, что после разбиения остаётся 2 единицы, которые не образуют полный делимый блок.

Целочисленное деление возвращает результат без дробной части; остаток отдельно демонстрирует то, что не вошло в полные группы. В программировании это удобно для разбиения данных и индексов циклов.

Школьный курс информатики: целочисленное деление и операция остатка (mod).

Порядок вычислений: от скобок к результату

Приоритеты арифметических операций в выражениях и правило равного приоритета слева направо.



Переполнение разрядной сетки

1

Переполнение возникает, когда результат выходит за границы диапазона типа данных. Например, для 8-битного знакового числа $127 + 1$ не помещается и может перейти к -128 .

3

Последствия для кода серьёзные: неверные значения, сбои в вычислениях и логике, а иногда и уязвимости безопасности, если переполненный результат используется в дальнейших операциях.

2

В больших программах переполнение встречается при суммировании, подсчёте элементов, длинах массивов и обработке крупных объёмов данных, когда сумма становится больше максимально допустимой.

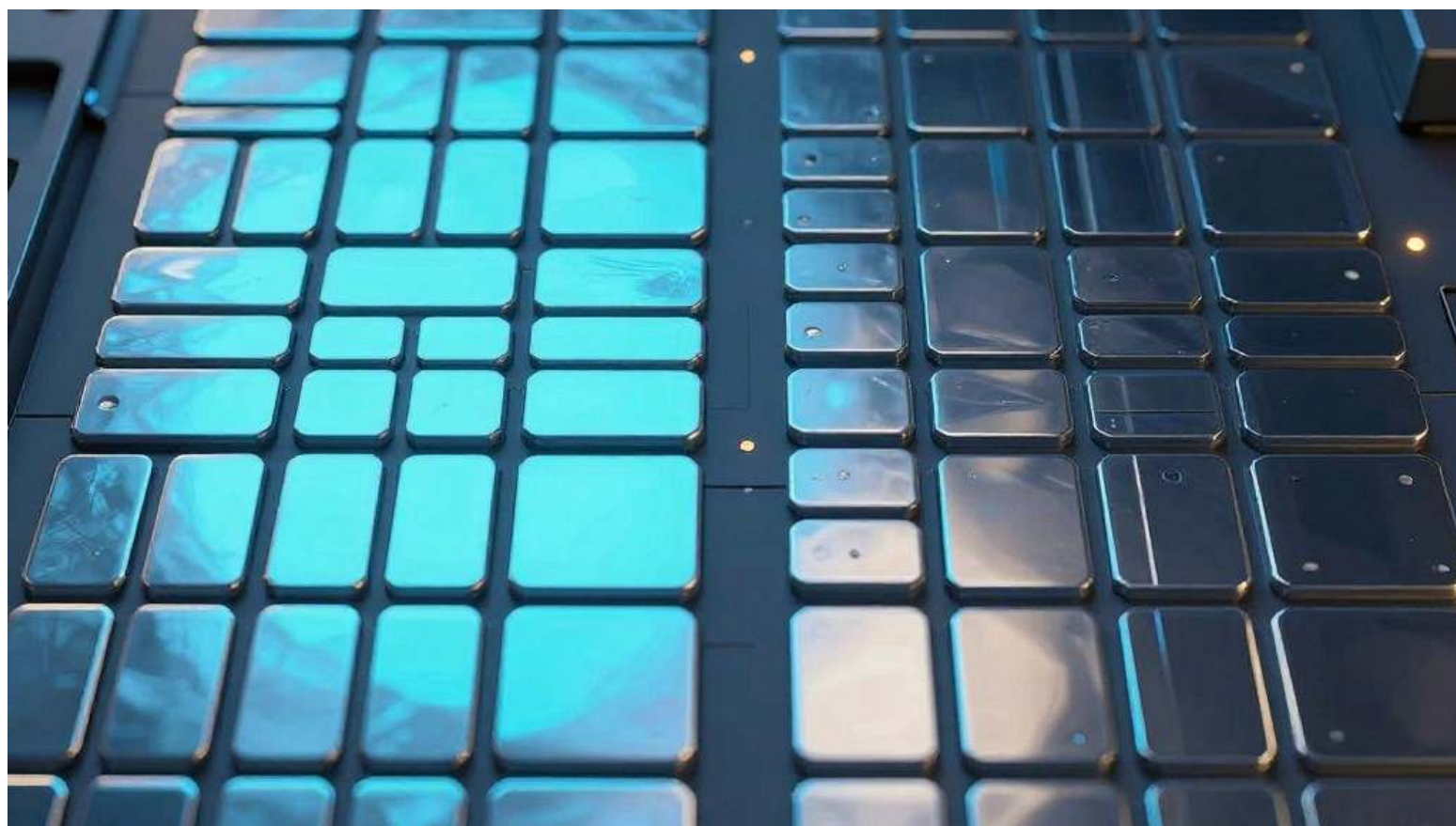
4

Надёжное программирование требует контроля диапазона: проверять границы перед операциями, использовать более широкий тип и предусматривать обработку граничных случаев.

Как компьютер «подружился» с отрицательными числами: дополнительный код

Инверсия и прибавление единицы как новый «язык» отрицательных

Когда нужно хранить знак и величину, компьютер чаще всего использует дополнительный код. Для перехода от положительного числа к представлению отрицательного инвертируют биты и прибавляют 1. Получается компактный формат, удобный для работы процессора.



Две записи для -1 и -2: мгновенное чтение без отдельных правил

В 8-битном представлении число -1 выглядит как 1111111 , а -2 — как 1111110 . Важно, что сложение и вычитание выполняются по единым правилам: процессору не требуется «особая схема» для отрицательных значений.



Целые числа в алгоритмах и циклах

I

Целые числа служат основой для счётчиков: в цикле `for` переменная последовательно принимает значения 0, 1, 2, 3 и далее, двигаясь на фиксированный шаг.

2

При поиске минимума, максимума и подсчёте элементов промежуточные результаты тоже удобно хранить в целых переменных, потому что сравнение и обновление значений не требуют дробной части.

3

В массивах индексы почти всегда целые, ведь это номера элементов. Неверный индекс из-за ошибки вычисления может привести к выходу за границы массива и нарушить логику программы.

4

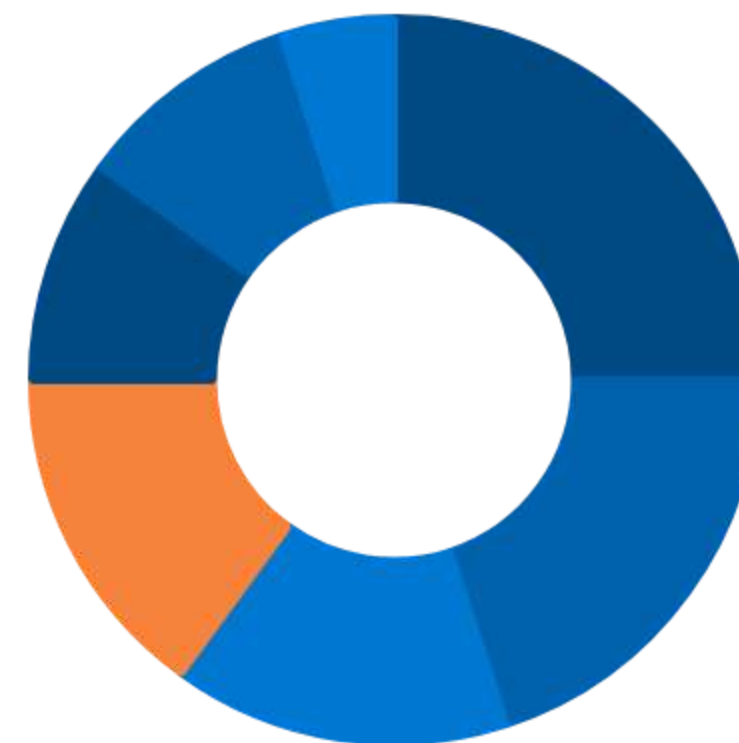
Потеря целочисленности или неверный порядок вычислений иногда ломает цикл: он может завершиться слишком рано, уйти в бесконечные повторения или посчитать не те элементы.

Где встречаются операции с целыми числами в повседневных программах

Даже когда пользователь видит «картинку» или «форму», внутри приложения часто считаются индексы, суммы, длительности и коды состояний — то есть используются целые числа.

Самые большие фрагменты диаграммы приходятся на счётчики и индексы: они обслуживают почти все алгоритмы, потому что требуют точных дискретных значений.

Обобщённая структура типичных учебных и прикладных задач; точные статистические доли зависят от конкретного приложения и набора сценариев.



Типичные ошибки при работе с целыми числами



Потеря знака

Код может неожиданно считать отрицательное число как большое беззнаковое. Тогда сравнения и условия ветвления дают неверный результат, а цикл может начать «двигаться» в противоположную сторону.



Переполнение

Если операция даёт число вне диапазона типа, результат «оборачивается». Это приводит к неверным подсчётам, ошибкам в адресации массива и иногда к уязвимостям, если значение участвует в безопасности.



Деление на ноль

В целочисленных вычислениях деление на ноль недопустимо и вызывает ошибку выполнения. В программах с вводом пользователя всегда нужно проверять делитель до операции.



Неверный порядок операций

Без учёта приоритетов выражение вроде $8 + 2 \times 3$ может быть вычислено иначе, чем ожидается. Такие ошибки часто «прячутся» в сложных формулах и проявляются только на тестах.



Путаница типов деления

Важный нюанс: целочисленное деление отбрасывает дробную часть, а вещественное — нет. Ошибка в выборе операции ломает логику, особенно при вычислении среднего и индексов.

Сравнение целых типов данных в языках программирования

Размер и диапазон зависят от языка и платформы. Но общий принцип один: чем шире тип, тем больше допустимые значения и тем больше памяти он занимает.

Выбор типа — это баланс между нужным диапазоном и экономией памяти: для больших сумм и длинных массивов нужен более широкий тип, иначе растёт риск переполнения.

Учебные материалы по базовым целочисленным типам; точные границы различаются в зависимости от языка.

Тип	Размер	Пример диапазона	Область применения
byte	1 байт	0...255 или −128...127	малые значения
short	2 байта	−32 768...32 767	умеренные счётчики
int	4 байта	примерно −2,1 млрд...2,1 млрд	общие вычисления
long	8 байт	очень большой диапазон	крупные суммы и файлы

Практические примеры из информатики и реальной жизни

I

Целые числа встречаются в подсчёте количества файлов, отображении номера страницы и вычислении возраста в полных годах, потому что эти величины дискретны и не требуют дробных значений.

2

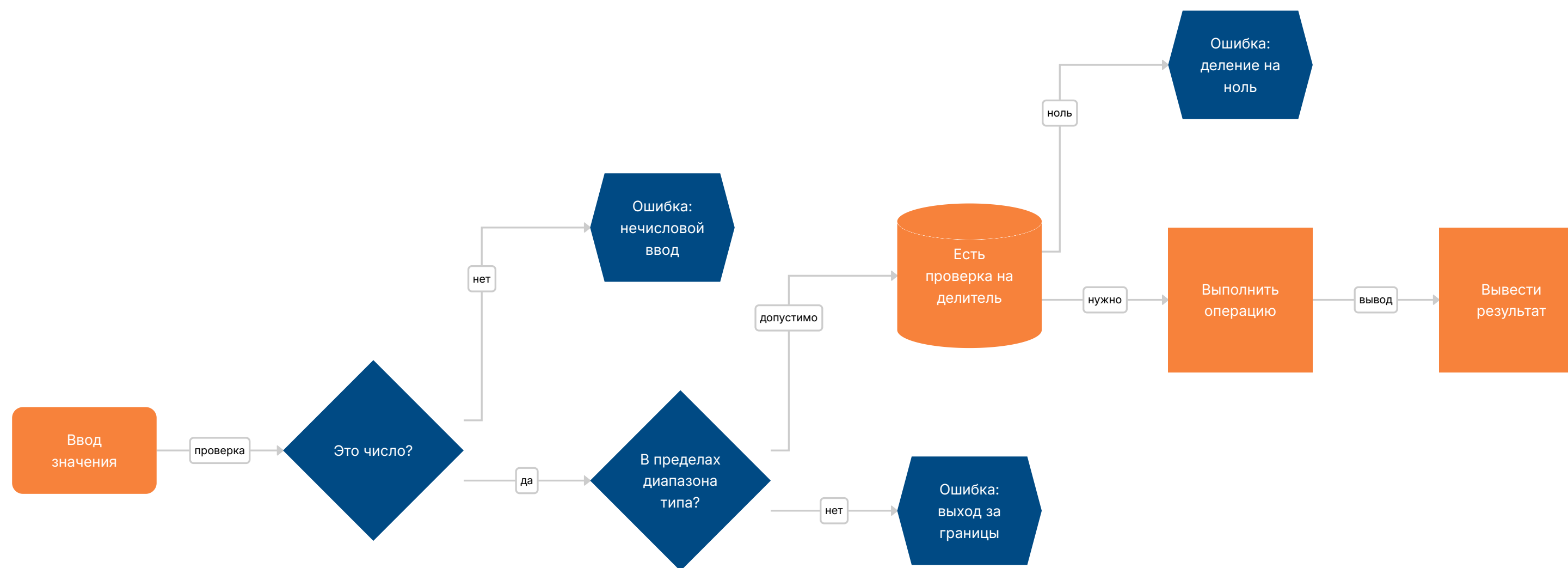
В играх они управляют баллами, а на «клеточном» поле — координатами элементов. В электронных таблицах и приложениях для оценок целые числа удобны для быстрых сумм и промежуточных результатов.

3

Системы посещений, лайков, шагов и сообщений используют целые счётчики: так обеспечиваются точные обновления и понятная логика условий, где дробность не имеет смысла.

Алгоритм проверки целого числа перед вычислениями

Безопасная последовательность: сначала валидация ввода и границ, затем операция.



Итоги: что важно знать об операциях с целыми числами

Целые числа — фундаментальный тип данных: их двоичное представление, ограниченный диапазон и правила для знаковых операций влияют на результат почти всех вычислений. Важны разрядность, дополнительный код, порядок операций и переполнение.