Числовые типы — конспект темы

Целочисленные типы

Все числа, записанные в коде в виде цифр, по умолчанию имеют тип `int`, если они достаточно маленькие. Если написать `auto x = 1`, переменная `x` автоматически получит тип `int`. Такие типы называются **целочисленными**. Они делятся на:

- знаковые (signed) — положительные и отрицательные числа и 0.
- безнаковые (unsigned) — положительные числа и 0.

Целочисленные типы C++

Тип	Количество байтов	Количество бит	Диапазон значений
int	4	32	от -2 147 483 648 до 2 147 483 647
unsigned int	4	32	от 0 до 4 294 967 295
int8_t	1	8	от -128 до 127
uint8_t	1	8	от 0 до 255
int16_t	2	16	от -32 768 до 32 767
uint16_t	2	16	от 0 до 65 535
int32_t	4	32	от -2 147 483 648 до 2 147 483 647
uint32_t	4	32	от 0 до 4 294 967 295
int64_t	8	64	от -9 223 372 036 854 775 808 до 9 223 372 036 854 775 807
uint64_t	8	64	от 0 до 18 446 744 073 709 551 615

Как выбрать подходящий целочисленный тип данных:

- Если вам достаточно размера и диапазона значений стандартного типа `int`, используйте его.
- Если хотите хранить числа порядка триллиона, нужен `int64_t`.
- Если нужно экономить память на числах, берите типы меньшей размерности: `int8_t`, `int16_t`.

- Если пишете программу, которая будет запускаться на неизвестной архитектуре, и нуждаетесь в 32-битном типе — используйте `int32_t`.

Чтобы узнать размер типа или выражения в байтах, примените оператор `sizeof`. Результат вызова `sizeof` имеет беззнаковый тип `size_t`.

Чтобы вывести минимальное и максимальное значение любого целочисленного типа, подключите `<limits>`.

Ограниченность памяти и переполнение

Переполнение происходит, когда значение переменной выходит из диапазона значений указанного типа. Код с переполнением может в любой момент дать сбой.

Как избежать переполнения:

- Заранее продумывать каждый шаг, в том числе промежуточные вычисления и неявные преобразования типов.
- Выбирать подходящие типы.
- Явно преобразовывать типы оператором `static_cast`.
- Настроить компилятор так, чтобы предупреждения он считал ошибками.

Операции с целочисленными типами

Чтобы произвести арифметическую или логическую операцию над разными целочисленными типами, компилятор неявно преобразует их к единому типу.

Правила преобразования типов:

- Все типы меньше `int` компилятор приводит к `int`.
- Если размер целочисленных типов больше `int`, меньший тип приводится к большему.
- Если размер типов одинаковый, но один из них беззнаковый, знаковый приводится к беззнаковому.

Если вы забыли правила, вызовите связанную с типом ошибку компиляции. В сообщении об ошибке увидите, как в вашем случае происходит преобразование.

Сравнение знакового и беззнакового числа:

```
cout << (-1 < 1) << endl;  //сравниваем два числа напрямую
```

Чтобы воспроизвести предыдущий пример и сделать единицу беззнаковой, добавьте суффикс `u`:

```
cout << (-1 < 1u) << endl;
```

Суффикс `u` (или `U`) показывает, что литерал в коде относится к типу `unsinged int`. То есть:

- тип литерала `1` — `int`;
- тип литерала `1u` — `unsigned int`.

Техника безопасности

Переполнение может произойти при итерации по вектору циклом `for`, когда компилятор сравнивает знаковый и беззнаковый тип.

Как избежать переполнения:

- Используйте только беззнаковые типы;
- Приводите беззнаковые типы к знаковым оператором `static_cast` и следите за размером вектора.
- Проверяйте, что код работает в крайних случаях.
- Не вычитайте из беззнаковых типов или будьте внимательны при вычитании.

Два подхода к выбору типа

1. Следовать семантике значений. Если у переменной по смыслу не бывает отрицательных значений — объявлять её беззнаковой.

Минус: придётся помнить все опасности преобразования знаковых и беззнаковых типов.

2. Приводить все беззнаковые типы к знаковым оператором `static_cast`.

Минус: `static_cast` заполонит ваш код.

Перечислимые типы

Перечислимый (перечисляемый) — тип данных с конечным числом упорядоченных именованных значений (перечислителей). Объявляется ключевыми словами `enum class`.

В зависимости от позиции в наборе перечислителям присваиваются целочисленные
значения:

```
enum CatBreed {
  RUSSIAN_BLUE,        //присваивается значение 0
  MAINE_COON,         //присваивается значение 1
  BRITISH_SHORTHAIR, //присваивается значение 2
  SIBERIAN          //присваивается значение 3
};

enum DogBreed {
  SIBERIAN_HUSKY,
  GOLDEN_RETRIEVER
  POMERANIAN,
  SAMOYED
};
```

Каждый `enum class` считается уникальным типом. Поэтому компилятор не будет проводить операции с перечислителями из разных наборов. Значения одного перечислимого типа сравнивают друг с другом операторами `==`, `!=`, `<` и `>`.

Значения перечислимых типов могут быть элементами множеств или ключами словарей. Порядок между значениями соответствует порядку их определения при объявлении типа.

Оператор switch

Оператор `switch` — компактный аналог `if`, а ветка `default` — компактный аналог `else`. `switch` может сравнить заданную переменную или результат выражения с конкретными значениями и выполнить действия в зависимости от того, с каким значением произошло совпадение. `default`-ветка выполнится, если не подошла ни одна `case`-ветка:

```
void ProcessRequest(
        set<int>& numbers,
        RequestType request_type,
        int request_data) {
```

```
    switch (request_type) {
    case RequestType::ADD:
        numbers.insert(request_data);
        break;
    case RequestType::REMOVE:
        numbers.erase(request_data);
        break;
    case RequestType::NEGATE:
        if (numbers.count(request_data) == 1) {
            numbers.erase(request_data);
            numbers.insert(-request_data);
        }
        break;
    default:
        cout << "Unknown request"s << endl;
    }
}
```

Оператор `break` означает выход из оператора `switch`.

Двойное двоеточие

Двойное двоеточие — оператор разрешения области видимости. У него несколько сфер применения.

Перечисление

Оператор `::` позволяет делать значения `enum`-типа неуникальными в рамках всей программы. Это одно из преимуществ `enum class`: все имена значений типа «спрятаны» внутрь его имени.

Обращение к сущностям внутри класса

Оператор `::` применяют, чтобы снаружи класса обратиться к полю, методу или типу внутри класса.

Пространство имён

Если не написать в начале программы `using namespace std`, все имена из этого пространства имён нужно употреблять с префиксом `std::`.