

Язык C++. Типы данных и константы

Фундаментальные арифметические типы

В языке C++ существует строго определённый набор базовых (встроенных) типов данных, которые не являются объектами классов из стандартной библиотеки. Эти типы принято разделять на две основные категории: **целочисленные** и **типы с плавающей точкой**.

Целочисленные типы:

- `int` (основной целочисленный тип)
- `short` (короткое целое)
- `long` (длинное целое)
- `long long` (очень длинное целое)
- `char` (символьный тип)
- `bool` (логический тип)

Модификаторы: Все эти типы могут иметь модификаторы `signed` (знаковый) и `unsigned` (беззнаковый).

Типы с плавающей точкой:

- `float` (одинарная точность)
- `double` (двойная точность)
- `long double` (расширенная точность)

Важно: Типы `string`, `vector` и `array` являются типами стандартной библиотеки (STL) и не относятся к фундаментальным типам языка.

Объявление констант с помощью `const`

Ключевое слово `const` указывает, что значение переменной не может быть изменено после её инициализации.

Особенности:

- Порядок написания: `const double` и `double const` эквивалентны
 - Обязательная инициализация при объявлении
 - Невозможно объявить константу без имени
-

Символьные литералы типа `char`

Литерал — явно заданное значение в исходном коде программы.

Синтаксис: Всегда заключаются в одинарные кавычки: `'A'`, `'1'`, `'\n'`

Различия:

- `'A'` — символьный литерал (тип `char`)
 - `"A"` — строковый литерал (тип `const char[2]`)
 - `'65'` — ошибка: в одинарных кавычках только один символ
-

Суффиксы числовых литералов

Суффиксы явно задают тип числового литерала:

Целочисленные суффиксы:

- `U` или `u` — `unsigned int`
- `L` или `l` — `long`
- `LL` или `ll` — `long long`
- `ULL` или `ull` — `unsigned long long`

Комбинации: Допустимы (например `ul`), но `ulong` — нестандартный

Использование ключевого слова `auto`

`auto` — спецификатор типа для автоматического определения типа на этапе компиляции.

Особенности:

- Тип фиксируется при инициализации
 - Не является динамическим типом
 - Упрощает сложные объявления
 - Повышает универсальность кода
-

Ключевое слово `constexpr`

`constexpr` (`constant expression`) — значение вычисляется на этапе компиляции.

Свойства:

- Неявно является `const`
 - Гарантирует `compile-time` вычисления
 - Размещение в `read-only` памяти
 - Широкие возможности оптимизации
 - Используется для размеров массивов, параметров шаблонов
-

Особенности типа `void`

`void` — специальный тип, представляющий отсутствие значения.

Применение:

1. Возвращаемый тип функции: `void func()`
2. Универсальный указатель: `void*` (требуется явное приведение)
3. В объявлениях функций без параметров (устаревший стиль)

Ограничение: Нельзя объявить переменную типа `void`

Неявное преобразование типов (Integer Promotion)

Integer Promotion — автоматическое преобразование мелких целочисленных типов к `int`.

Применяется для:

- Арифметических операций с `short`, `char`, `bool`
- Присваивания значений мелких типов переменным `int`
- Побитовых операций

Цель: Повышение эффективности вычислений

Правила арифметических преобразований

Основной принцип: Тип с меньшим диапазоном → тип с большим диапазоном

Особые случаи:

- `unsigned + signed → unsigned`

- Типы с плавающей точкой > целочисленные типы
 - Приоритет размеров: long > int > short > char
-

Восьмеричные литералы

Формат: Начинаются с 0, затем цифры 0-7

Примеры:

- 0777 — восьмеричное 511
- 0 — восьмеричный ноль

Ошибки:

- 8080 — десятичный (начинается не с 0)
 - 0o777 — только C++14+
 - 089 — недопустимая цифра 9
-

Преимущества использования const-переменных

1. **Читаемость:** MaxUsers ВМЕСТО 100
 2. **Поддержка:** Изменение в одном месте
 3. **Безопасность:** Защита от случайного изменения
 4. **Оптимизация:** Подстановка значения в код
-

Логические литералы типа bool

Литералы:

- true — логическая истина
- false — логическая ложь

Важно:

- TRUE/FALSE — могут быть макросами
 - 0 → false, любое ненулевое → true
-

Целочисленное деление

Особенности:

- Дробная часть отбрасывается (усечение к нулю)
 - Остаток: оператор %
 - Для математического деления: хотя бы один операнд с плавающей точкой
 - $-5 / 2 = -2$
-

Uniform Initialization (Инициализация с фигурными скобками)

Преимущества:

- Универсальность работы со всеми типами
- Запрет сужающих преобразований
- Единообразие синтаксиса
- Безопасность (предотвращение потерь данных)

Примеры: `int x{5};`, `std::vector<int> v{1, 2, 3};`

Статические константы с модификатором `static`

`static` в глобальной области → внутренняя компоновка (internal linkage)

Преимущества:

- Использование одинаковых имён в разных файлах
 - Соккрытие реализации модуля
 - Управление областью видимости
 - Оптимизация доступа
-

Размер типа `int` на разных платформах

Стандартные гарантии:

- `sizeof(char) <= sizeof(short)`
- `sizeof(short) <= sizeof(int)`

- `sizeof(int) <= sizeof(long)`

На практике: На большинстве систем `int` — 4 байта

Перечисления (enum)

Перечисление — пользовательский тип с набором именованных констант.

Особенности:

- Именованные целочисленные константы
- Явное или автоматическое назначение значений
- Улучшение безопасности типов
- Самодокументируемый код

Пример: `enum Status { Ok = 0, Error = 1, Waiting = 2 };`

Явное приведение типов с `static_cast`

`static_cast` — безопасное явное преобразование с проверкой на этапе компиляции.

Преобразование `float` → `int`:

- Усечение (дробная часть отбрасывается)
 - Без округления: `4.99` → `4`, `-3.14` → `-3`
-

Спецификатор типа `decltype`

`decltype(expression)` — возвращает тип выражения без его вычисления.

Применение:

- Шаблонное программирование
- Создание псевдонимов сложных типов
- Метапрограммирование
- Работа с лямбда-выражениями

Работает: Только на этапе компиляции

Создание псевдонимов типов с помощью using

Синтаксис: `using NewTypeName = ExistingType;`

Преимущества перед typedef:

- Более читаемый синтаксис
- Лучшая работа с шаблонами
- Возможность создания шаблонных псевдонимов
- Естественный порядок записи

Ошибки:

- `typedef using ...` — смешивание ключевых слов
- `using byte unsigned char;` — пропущен `=`
- Несоответствие типов