

Типы в языке C++

Фундаментальные типы

Это произведение доступно по лицензии
"Attribution-ShareAlike" ("Атрибуция — На тех же условиях") 4.0 Всемирная (CC BY-SA 4.0)
<http://creativecommons.org/licenses/by-sa/4.0/deed.ru>



February 1, 2022

Этапы изучения C++

- Фундаментальные типы данных
`int a = 0x1E; char f = 'z'; double t = .5e12;`
- Операторы и выражения
`c + f(e) -d<<7 + *i++;`
- Инструкции
`for(z=x; z&7; z<<2) {}; if(z) {};`
- Функции
`int f(const char *c, int &z);`
- Классы, пространства имён
`class A { ... }; namespace X { ... };`
- Исключения
`try {} catch(){};`
- Шаблоны (+STL)
`vector<double> c;`
`template<typename T> class X { ... };`

Классификация типов C++

Фундаментальные (встроенные типы).

Компилятор изначально знает эти типы и правила их обработки. Примеры: **int, char, double**.

Пользовательские типы

Пользователь сам задаёт их имена и правила работы с ними. Для определения используются ключевые слова **class, struct, enum, union**.

Производные типы

Модификации фундаментальных и пользовательских типов: указатели, ссылки, массивы, функции и их комбинации.

С помощью ключевого слова **typedef** (C++11 — **using**) определяются **СИНОНИМЫ** типов.

Интегральные типы данных

Интегральные (целочисленные) типы данных предназначены для хранения значений целых чисел.

- **char**
- **wchar_t**
- **int**
- **bool**

В C++11 введены новые интегральные фундаментальные типы:

char16_t, char32_t .

Синонимы, определенные в `<stdint.h>` (позволяют явно указать размер объектов данного типа):

**int8_t, int16_t, int32_t, int64_t,
uint8_t, uint16_t, uint32_t, uint64_t,
intmax_t, uintmax_t, intptr_t, std::size_t ...**

Интегральные типы данных

Тип char

Тип **char** предназначен для хранения символов (а точнее, номеров символов в какой-то из 8-битных кодовых таблиц).

Для записи непосредственных значений в программе используются **литералы**. Для типа **char** литералы записываются в прямых одинарных кавычках.

```
char a = 't';  
    // значение – символ 't'  
char b = '\n', b1 { '\ ' };  
    // значение b – спец. символ новой строки,  
    // b1 – одинарная кавычка  
char o1 = '\377';  
    // соответствует символу с кодом 255  
    // восьмеричное значение  
char x1 = '\xFF';  
    // шестнадцатеричное значение (=255)  
char8_t c8t { u8'Q' }; // C++20. // u8'Й' – error
```

C++20: введён тип **char8_t** (for UTF-8).

Многобайтные символы

wchar_t, char16_t, char32_t, uchar16_t ...

Для одновременного хранения символов разных национальных алфавитов используются символы, занимающие более 1 байта (и соответствующие литералы).

```
char16_t c16 { u'ї' }; // bad!  
// char16_t c16 = u'\u1234';  
char32_t c32 { U'я' };  
wchar_t wc = L'Ю';  
  
// cout << "c16= " << c16 << endl; // Error!  
cout << "c16=␣" << (unsigned)c16  
      << "␣c32=␣" << (unsigned)c32  
      << "␣wc=␣" << (unsigned)wc << endl;  
// c16= 1031 c32 = 1103 wc= 1070
```

C++20: if can not be represented with one byte — error.

Литералы символьных строк

Для записи литералов строк (массивов константных символов) используются литералы с двойными кавычками и raw-строки

```
cout << "abcd\nefdh" "AAAA" << endl;

cout << R"(_c:\nadya\tanya\fedya_)" << endl; // uR, UR
const char *s1 = R"!!!(c:\nnn\ttt)!!!";
cout << "s1=\"\" << s1 << '\"' << endl;

const char8_t *s2 = u8R"!!!(сыы:\няя\t)!!!"
cout << "s2=\"\" << s2 << '\"' << endl;
```

В конце массива автоматически добавляется нулевой символ '\000'. Попытка изменить такой массив приводит к неопределённому поведению.

Префиксы: "u8" - UTF8-строка, "u" - char16_t-строка, "U" - char32_t-строка, "L" - wchar_t-строка.

Тип **int** — основной для хранения целых чисел. Размер — наиболее удобный для данной архитектуры, но не менее 16 бит.

```
int am = -1234, ap = 32767;  
// для инициализации используем десятичные литералы  
// C++14: ac = 1'234'567;  
  
int bo = 0777;  
// если первый символ – '0' – восьмеричный литерал,  
// 1 цифра (0–7) = 3 бита.  
  
int bh = 0xAAAA5555;  
// '0x' – признак шестнадцатеричного литерала,  
// 1 цифра (0–F) = 4 бита.  
  
int bb = 0b010101010101;  
// C++11: '0b' – признак двоичного литерала,  
// 1 цифра (0–1) = 1 бит.
```

bool

Логический тип

Интегральный тип **bool** предназначен для хранения логических значений, и имеет 2 литерала:

false — ложь и

true — истина.

```
bool b1 = false;
```

```
bool b2 = 10 > 5;
```

```
bool b3 = b1 && b2;
```

Типы значение с плавающей точкой

float, double, long double

Для хранения дробных значений в C++ используется представление с плавающей точкой вида

$$x = a \cdot 10^n,$$

где a — мантисса, n — порядок. Стандарт: IEEE 754. Для хранения таких чисел используются типы **float**, **double**, **long double** и соответствующие литералы.

```
double a1 = -1.23e5; // = -123450.0000000;  
double b = 1e8; // = 1 * 10^8 = 100000000;  
float f1 = -2.1e-3f;  
long double g = 1.23456789012345e500L;  
double x = 0x1.AP2; // C++17 hexadecimal form
```

Между собой отличаются количеством знаков в мантиссе и максимальным значением порядка. Основной тип — **double**.

Специальный тип **void** предназначен для обозначения отсутствия какого-либо типа в данном месте. Также используется для построения производных типов.

```
void f1( int a ); // функция ничего не возвращает  
int g1( void ); // функция ничего не получает  
void *z = malloc( 512 ); // z указывает на нечто.
```

Литералов не имеет.

std::nullptr_t

Тип для нулевого указателя

Специальный тип **std::nullptr_t** предназначен для литерала **nullptr_t**, который обозначает особое значение указателя, по которому не могут размещаться обычные объекты.

```
void *p = nullptr;
```

Формально определён в <cstddef>.

Модификаторы размера

short, long, long long

Для типа **int** можно управлять размером типа с помощью модификаторов **short**, **long**, **long long**. При этом сам тип **int** можно опускать. Для литералов есть суффиксы 'l', 'L', 'll', 'LL'.

```
long int la = 123L;  
long la1 = 5;  
  
short s = 123;  
  
long long z = -1LL;  
auto lz = 1234LL;  
auto sz = 512uz; // std::size_t since C++23
```

В C++11 можно использовать синонимы типов `int8_t`, `int16_t`, `int32_t`, `int64_t`, `uint8_t`, `uint16_t`, `uint32_t`, `uint64_t` — для точного контроля размера.

Модификаторы знаковости

signed, unsigned

Для типов **char**, **int** можно управлять возможностью представления отрицательных значений с помощью модификаторов **signed**, **unsigned**.

По умолчанию **int** — знаковый тип, **char** — неопределённого знака.

```
unsigned u = 65536u;
```

```
signed char c = -128;
```

```
unsigned char uc = 255;
```

Компилятор может дать предупреждение, если в одном выражении используются знаковые и беззнаковые объекты.

Для внутреннего представления отрицательных чисел используется **дополнительный код** (C++20).

Модификаторы постоянства и волатильности

const, volatile

Объект, помеченный модификатором **const** не изменяет своё состояние с момента создания до момента уничтожения.

Объект, помеченный модификатором **volatile** может изменять своё состояние независимо от хода выполнения программы. Компилятор не должен делать предположения о значении такого объекта.

```
unsigned const char cc = 'Z';  
    // без инициализации – ошибка  
const int mask1 = 0xAAAA;  
const unsigned long long zz = 54LLU;  
volatile char t;
```

Непротиворечивые модификаторы можно комбинировать. В C++11 введен модификатор **constexpr**. В C++20 — **constexpr**, **consteval**, **constexpr**.

Из фундаментальных и пользовательских типов можно создавать производные, непротиворечивым способом комбинируя

- **массивы** ([]);
- **указатели** (*);
- **ссылки** (&, &&);
- **функции.**

Размер (в символах) объекта, типа или выражения можно определить оператором **sizeof**.

Производные типы

Массивы

Массив — это производный тип, объекты которого содержат несколько элементов базового типа, при этом элементы массива расположены в памяти последовательно. Доступ к одному элементу возможен с помощью оператора `[]`. Индексация начинается с **0**! Значения индексов не проверяются!

```
int m1[8];
int m2[] = { 0, 1, 2, 3, 4, 5 };
const int nm = 5;
double v[nm];
char s1[] = "ptnhlo";

m1[0] = m2[5] + 2;
v[nm-1] = 1.7e-5;
```

В современном C++ рекомендуется вместо массивов использовать шаблонные контейнеры STL:

vector, array, list, deque ...

Производные типы

Указатели

Указатель содержит адрес, по которому в памяти находится объект (или нечто). В определении типа используется символ “*”. Для получения объекта по указателю используется унарный оператор “*” (разыменование). Для взятия адреса используется унарный оператор “&”.

```
int a = 5, b = 4;
int *pa = &a;
*pa = 12;
pa = &b; *pa = 42;
void *v = malloc( 123 ); free( v );
char *ps = new char[512]; delete[] ps;
const char *sc = "some_string";
double *b; // bad, points to somewhere!
```

Литерал “**nullptr**” (нулевой указатель) обозначает адрес, по которому не могут располагаться объекты.

В современном C++ рекомендуется использовать

unique_ptr, shared_ptr

Производные типы

Ссылки

Ссылка — другое имя объекта. В определении типа используется символ “&”. Для реализации ссылок часто используется адрес, но правила — другие! Ссылка всегда указывает на объект, и всегда только на один — переставить ссылку на другой объект или не инициализировать **нельзя!** Доступ к объекту, на который ссылаемся — непосредственный, без операторов.

```
int a = 5;
int &ra = a;
ra = 7;           // a = 7;
int *pa1 = &ra; // address of 'a' !
int &zz;         // Error !!!!!!!!!
```

Чаще всего используются при построении специальных функций (конструктор копирования, операторы ...). В C++11 есть rvalue-ссылки (&&). На самом деле всё ещё сложнее ...