Фундаментальные типы данных C++

Концепция типа данных

Тип данных - свойство программных сущностей, которое определяет:

1. Множество допустимых значений
2. Допустимые операции
3. Низкоуровневое представление (интерпретация битов)

```
// int - частный случай типа данных (целое число)
int main() {
  return 0;
}
```

Фундаментальные типы

В C++ существует большое количество типов данных. Более того, можно создавать и свои типы.

Однако каждый тип данных состоит из, или является производным от одного из фундаментальных типов:

- "Пустой" тип: `void`
- Целые числа: `int`, `short`, `long`, `long long`
- Числа с плавающей точкой: `float`, `double`, `long double`
- Символьный тип: `char`, `char8_t`, `char16_t`, `char32_t`
- Логический тип: `bool`
- `std::nullptr_t`

Целые типы: `int`

Основным целочисленным типом является `int`.

Стандарт C++ гарантирует, что размер `int` **как минимум** 16 бит (2 байта*).

Это значит, что значения типа `int` могут лежать в диапазоне $[-32'768, 32'767]$.

На практике, как правило, `int` занимает 32 бита (4 байта), то есть принимает значения из $[-2'147'483'648, 2'147'483'647]$.

```
// литералы типа int
0 1 2 -1 -2 100 123 -123456789 987'654'321
```

*: Далее предполагаем, что 1 байт = 8 бит

Целые типы: `long` (`long int`)

Стандарт C++ гарантирует, что размер `long` **как минимум** 32 бита (4 байта), но **не меньше** размера `int`.

Это значит, что значения типа `long` могут лежать в диапазоне $[-2'147'483'648, 2'147'483'647]$.

```
// литералы типа long
0l 1L 2l -1L -2l 100L 123l -123456789L 987'654'321l
```

Целые типы: `long long` (`long long int`)

Стандарт C++ гарантирует, что размер `long long` **как минимум** 64 бита (8 байт), но **не меньше** размера `long`.

Это значит, что значения типа `long long` могут лежать в диапазоне $[-9'223'372'036'854'775'808, 9'223'372'036'854'775'807]$.

```
// литералы типа long long
0ll 1LL 2ll -1LL -2ll 100LL 123ll -123456789LL 987'654'321ll
```

Целые типы: `short` (`short int`)

Стандарт C++ гарантирует, что размер `short` **как минимум** 16 бит (2 байта), но **не больше** размера `int`.

На практике, как правило, `short` занимает 16 бит (2 байта), то есть принимает значения из $[-32'768, 32'767]$.

`short` не имеет собственных литералов.

Целые типы: модификаторы `signed` / `unsigned`

По умолчанию все целые типы *знаковые*, то есть могут хранить как положительные, так и отрицательные числа.

Чтобы это подчеркнуть к названиям типов можно приписывать модификатор `signed`:

```
int == signed int == signed

long      == signed long      == long int      == signed long int
long long == signed long long == long long int == signed long long int
short     == signed short     == short int     == signed short int
```

Целые типы: модификаторы `signed` / `unsigned`

Если предполагается, что значение должно хранить только неотрицательные числа, то можно к имени типа добавить `unsigned`.

В этом случае допустимые значения меняются следующим образом:

- 2 байта: $[0, 65'535]$
- 4 байта: $[0, 4'294'967'295]$
- 8 байт: $[0, 18'446'744'073'709'551'615]$

```
unsigned int       == unsigned
unsigned long      == unsigned long int
unsigned long long == unsigned long long int
unsigned short     == unsigned short int
```

```
// беззнаковые литералы
1u, 2UL, 3ull
```

Целые типы: операции

Над целыми числами (кроме `short`!) можно выполнять все арифметические операции:

```
+1, -1, 2 + 2, 3 - 2, 4 * 5, 7 / 2, 13 % 5, 6 & 1, ...
```

- Результатом операции является значение того же типа, что и у операндов.
- При переполнении беззнаковых чисел выполняется арифметика по модулю.
- В остальных ситуациях переполнение - это *Undefined Behavour (неопределенное поведение)*.
- Деление на 0 - *UB*.
- Битовые сдвиги отрицательных чисел тоже могут приводить к *UB*.

Целые типы: операции

А что если операнды имеют разные типы (или тип `short`)?

```
1 + 3ul, 2l + 10ll, ...
```

Порядок действий следующий:

- (`signed` / `unsigned`) `short` приводится к `int`
- Менее широкий тип приводится к более широкому (`int` -> `long` -> `long long`)
- Знаковый тип приводится к беззнаковому

Целые типы: упражнение

Какой тип будет иметь результат в каждом из случаев?

```
0 + 0l;      // ??
0ll + 0;     // ??
0l + 0ll;    // ??
0u + 0;      // ??
0 + 0ul;     // ??
0ul + 0ll;   // ??
```

Целые типы: упражнение

Какой тип будет иметь результат в каждом из случаев?

```
0 + 0l;      // long
0ll + 0;     // long long
0l + 0ll;    // long long
0u + 0;      // unsigned int
0 + 0ul;     // unsigned long
0ul + 0ll;   // unsigned long или long long(зависит от ширины long)
```

Целые типы фиксированной ширины

На разных системах целые типы могут иметь разные размеры, что осложняет жизнь.

Для решения этой проблемы в C++11 появились типы *фиксированной ширины*:

```
int8_t, int16_t, int32_t, int64_t
uint8_t, uint16_t, uint32_t, uint64_t
```

Они имеют **в точности** тот размер, который указан в названии.

Чтобы использовать их, необходимо подключить заголовочный файл `<cstdint>`

```
#include <cstdint>
```

Символьные типы: `char`

Тип `char` используется для хранения символов.

Символ представляется некоторым 1 байтовым целым числом согласно ASCII таблице (https://www.asciitable.com/).

```
// символьные литералы
'a', '1', '@', '\n', ...
```

Так как `char` - целое число, оно может быть как знаковым, так и беззнаковым (какой из типов используется по умолчанию - зависит от реализации):

```
signed char, unsigned char
```

Символьные типы: `wchar_t`, `char16_t`, `char32_t`

Широкие символьные типы используются для хранения символов из кодировок `UTF-16` и `UTF-32`.

https://en.cppreference.com/w/cpp/language/character_literal

Символьные типы

Так как символы представляются целыми числами, тип `char` можно считать 8-битным целым числом и использовать все арифметические операции

```
'E' + ('a' - 'A') == 'e'
```

Напоминание: при выполнении арифметики числа, у которых тип имеет ранг меньший `int`, приводятся к `int`.

Упражнение

Какой тип имеет результат выражения?

```
'a' + 0;    // ?
'a' + 0l;   // ?
'a' + 0ll;  // ?
'a' + 'a';  // ?
```

Упражнение

Какой тип имеет результат выражения?

```
'a' + 0;    // int
'a' + 0l;   // long
'a' + 0ll;  // long long
'a' + 'a';  // int
```

Логический тип: `bool`

Объекты логического типа могут принимать всего 2 значения: `true` / `false`.

Имеет размер в 1 байт.

Обычно используется для хранения результата сравнения:

```
5 > 6;   // false
5 < 6;   // false
5 >= 6;  // false
5 <= 6;  // false
5 == 6;  // false
5 != 6;  // false
```

Логический тип: `bool`

Объекты типа `bool` могут выступать операндами логических операций:

```
5 > 6 && 5 < 6;  // false (логическое "и")
5 > 6 || 5 < 6;  // true (логическое "или")
!(5 > 6);        // true (логическое "не")
```

`&&` и `||` - особенные операции:

1. Гарантируется, что выражение слева будет вычислено до выражения справа.
2. Если слева значение `true`, то правая часть `||` вычисляться не будет.
3. Если слева значение `false`, то правая часть `&&` вычисляться не будет.

```
5 < 6 || ...;  // что бы ни стояло справа, оно не вычисляется (совсем)
5 > 6 && ...;  // что бы ни стояло справа, оно не вычисляется (никак)
```

Логический тип: `bool`

Логический тип в C++ является разновидностью целового типа и может быть использован в арифметических выражениях. При этом `true == 1`, а `false == 0`:

```
true + 5;      // int: 6
10l * false;   // long: 0
```

Верно и обратное: при подстановке в логическую операцию ненулевое значение интерпретируется как `true`, нулевое - `false`:

```
5 && 1;  // true
0 || 0;  // false
!-1;     // false
```

Числа с плавающей точкой: `float`, `double`, `long double`

Числа с плавающей точкой используются для хранения рациональных чисел.

`float` - числа с одинарной точностью (4 байта, примерно $[\pm 10^{-38}, \pm 10^{38}]$)

`double` - числа с двойной точностью (8 байт, примерно $[\pm 10^{-308}, \pm 10^{308}]$)

`long double` - числа с расширенной точностью (16 байт, примерно $[\pm 10^{-4932}, \pm 10^{4932}]$)

```
// дробные литералы
0., 1.5, 3.14159;    // double
0f, 1.5F, 3.14159f;  // float
0l, 1.5L, 3.14159l;  // long float

123.456e10;   // 123.456 * 10^10
123.456e-10;  // 123.456 * 10^(-10)
```

Числа с плавающей точкой: особенности

- Можно применять те же арифметические операции, что и к целым числам (кроме битовых операций и взятия остатка):

```
0.1 + 0.2 * 5.67 / 0.9  // деление дробное
```

- Следует помнить, что дробные числа имеют ограниченную точность при вычислениях:

```
0.1 + 0.2 != 0.3;
```

- Числа с плавающей точкой всегда знаковые.
- Имеются специальные значения: `+inf`, `-inf`, `nan`:

```
1. / 0 /* inf */;    -1. / 0 /* -inf */;    0. / 0 /* nan */;
```

Числа с плавающей точкой: операции с целыми

- Если типы чисел с плавающей точкой не совпадают, то менее широкий аргумент приводится к более широкому:

```
5.0 + 1.5f;  // double
5.0 + 1.5l;  // long double
```

- При выполнении арифметической операции над целым и дробным числом целое число приводится к типу дробного:

```
1 + 0.0f;  // float
```

Пустой тип: `void`

Тип `void` - тип с пустым множеством значений.

Главное применение - сообщить о том, что выражение ничего не возвращает (результата нет).