C++

Башарин Егор

eaniconer@gmail.com

ЛЕКЦИЯ II

Fundamentals

тип

- свойство сущностей (функции, объектов, выражений)
- определяет множество операций над сущностями
- задает семантику последовательности битов

ФУНДАМЕНТАЛЬНЫЕ ТИПЫ

- `void`
- `std::nullptr_t`
- `sizeof`
- арифметические типы

BOOLEAN

- Тип: `bool`
- Значения: `true | false`
- `sizeof(bool) >= 1`

INTEGER TYPES

- Тип: `int`
- Модификаторы:
  - Знаковость: `signed`, `unsigned`
  - Размер: `short`, `long`, `long long`

при использовании модификатора `int` может быть опущен

`signed` используется по умолчанию

INTEGER TYPES

```
// Width in bytes by C++ standard:
sizeof(short int) == sizeof(short) >= 2

sizeof(int) >= 2

sizeof(long int) == sizeof(long) >= 4

sizeof(long long int) == sizeof(long long) >= 8
```

- Знаковость не влияет на размер
- Связь с моделью данных: LP32, ILP32, LLP64, LP64

FIXED WIDTH INTEGER TYPES

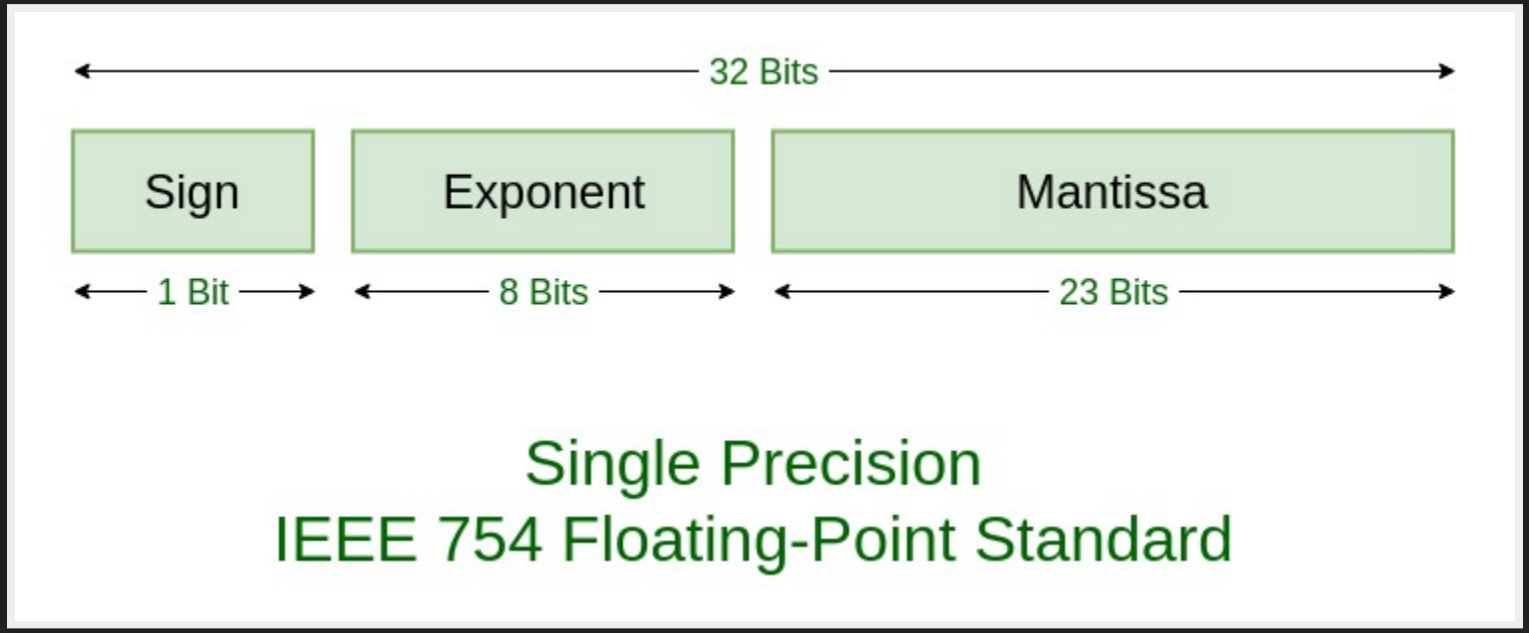
```cpp
// Требуется гарантия размера типа
#include <cstdint>

// signed
int8_t i8;   // -128..127
int16_t i16; // -32768..32767
int32_t i32;
int64_t i64;

// unsigned
uint8_t ui8;   // 0..255
uint16_t ui16; // 0..65535
uint32_t ui32;
uint64_t ui64;
```

FLOATING POINT TYPES

- `float`
- `double`
- `long double`

Special values: +INF, -INF, -0.0, NaN

FLOAT

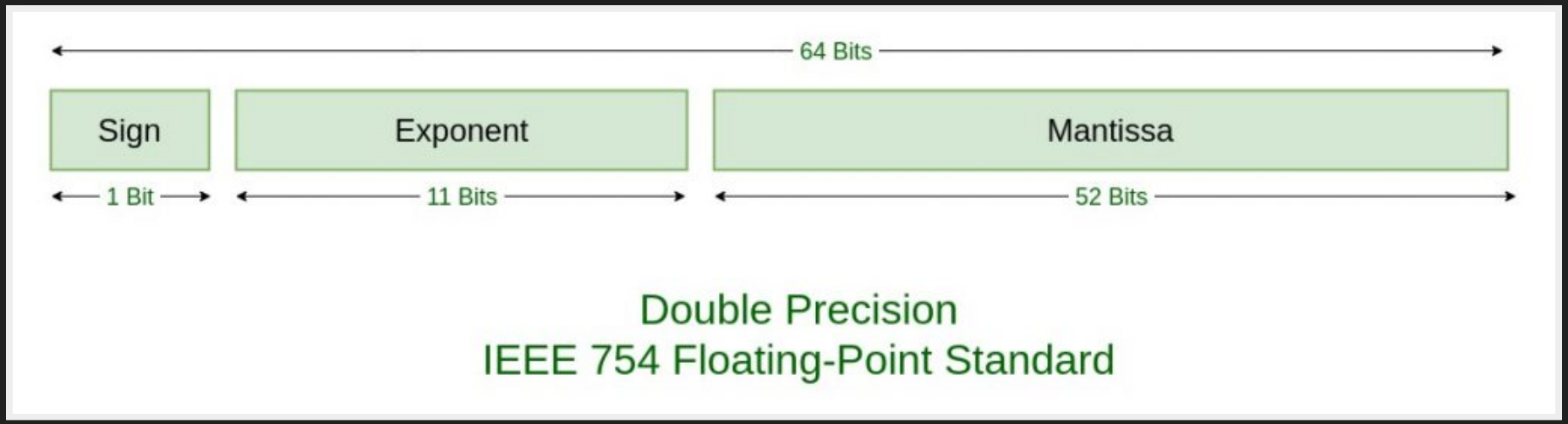


play with bits

FLOAT

```
dec to bin:      4.5 -> 100.1
normalization:   1.001 * 2^2

mantissa:                    001
(biased) exponent: 127 + 2 -> 10000001
```

DOUBLE


64 Bits
Sign
Exponent
Mantissa
1 Bit
11 Bits
52 Bits
Double Precision
IEEE 754 Floating-Point Standard

LONG DOUBLE

Read wiki

CHARACTER TYPES

- `char`
- `signed char`
- `unsigned char`
- `wchar_t`
- `char16_t` (C++11)
- `char32_t` (C++11)
- `char8_t` (C++20)

CHARACTER TYPES

`char`, `unsigned char`, `signed char` — разные типы

```cpp
#include <iostream>
#include <type_traits>

int main()
{
    std::cout
        << std::is_same_v<char, unsigned char>
        << std::is_same_v<char, signed char>;
}

// Output: 00
```

`char`

- эффективное представление символа (текста)
- знаковость зависит от платформы:
  - `unsigned` на ARM, PowerPC
  - `signed` на x86, x64
- `sizeof(char) == 1`

`unsigned char`

- используется для побайтового представления объекта в памяти

UTF

- `wchar_t` - UTF-16 on Windows
- `char16_t` - UTF-16
- `char32_t` - UTF-32
- `char8_t` - UTF-8

СВОЙСТВА АРИФМЕТИЧЕСКИХ ТИПОВ

std::numeric_limits<T> из <limits>

```cpp
#include <limits>

char maxChar = std::numeric_limits<char>::max();
double minDouble = std::numeric_limits<double>::min();
```

REFERENCES

en.cppreference.com/w/cpp/language/types

en.cppreference.com/w/cpp/types/is_fundamental

en.cppreference.com/w/cpp/types/numeric_limits

ЛИТЕРАЛЫ

литерал - запись в исходном коде, представляющая собой фиксированное значение определенного типа.

```
'a'          // char

10           // int
20u  20U     // unsigned int
10L  10l     // long
10UL 10ul    // unsigned long

12.2f        // float
12.2         // double

true false // bool

"abcd";      // c-строка

nullptr;     // nullptr_t
```

ОПЕРАТОРЫ

СРАВНЕНИЕ

```
int a = getA();
int b = getB();
```

a == b;

a != b;

a < b; a > b;

a <= b; a >= b;

АРИФМЕТИКА

```
int a = getA();
int b = getB();

+a;
-a;

a + b; a - b; a * b; a / b; a % b;

// bitwise
~a;     // NOT
a & b;  // AND
a | b;  // OR
a ^ b;  // XOR

a << b; a >> b; // SHIFT
```

ПРИСВАИВАНИЕ

```
int a = getA();
int b = getB();

a = b;

a += b; a -= b; a *= b; a /= b; a %= b;

a &= b; a |= b; a ^= b;

a <<= b; a >>= b;
```

ИНКРЕМЕНТ, ДЕКРЕМЕНТ

```
int a = 10;

// prefix
++a;
--a;

// postfix
a++;
a--;
```

PREFIX VS POSTFIX

```
int a = 10;
int b = a++; // postfix
assert(a == 11);
assert(b == 10);

int c = 10;
int d = ++c; // prefix
assert(c == 11);
assert(d == 11);
```

LOGIC

```
bool a = getA();
bool b = getB();

bool c = !a;
bool d = a && b; // Short-circuit evaluation
bool e = a || b; // Short-circuit evaluation
```

POINTERS

```c
int main() {
    double pi = 3.1415;

    double* ptrToPi = &pi;
}
```

POINTER VALUES

- pointer to object/function
- null pointer
- invalid pointer

POINTER TO OBJECT

Представляет адрес первого байта памяти, в которой расположен объект

```
int object = 3;

int* ptr = &object // address-of operator

*ptr = 4; // indirection operator

assert(object == 4);

// * see also:
// https://en.cppreference.com/w/cpp/memory/addressof
```

POINTERS TO VOID

```
int i = 1;

int* pi = &i;

void* vi = pi; // implicit conversion from any type

int* pi2 = static_cast<int*>(vi); // explicit cast required
```

NULL POINTERS

- Специальное значение, присущее указателю любого типа
- Используется, чтобы указать на отсутствие объекта
- Разыменовывание ведет к UB

INVALID POINTERS

Указатель может стать недействительным, если он указывает на локальный объект, который был уничтожен.

```
int* makeInvalidPtr() {
    int i = 0;
    return &i;
}

void f() {
    int* ptr = makeInvalidPtr(); // invalid pointer
}
```

КОНСТАНТНОСТЬ

```
const int i = 32;
i = 10; // error: cannot assign to variable
```

КОНСТАНТНОСТЬ И УКАЗАТЕЛИ

```
int i = 32;

int* p = &i; // pointer to int

int j = 1;
p = &j;     // OK
*p = 100;   // OK
```

КОНСТАНТНОСТЬ И УКАЗАТЕЛИ

```
int i = 32;

const int* p = &i; // pointer to const int

int j = 1;
p = &j;     // OK
*p = 100;   // error
```

КОНСТАНТНОСТЬ И УКАЗАТЕЛИ

```
int i = 32;

int* const p = &i; // const pointer to int

int j = 1;
p = &j;     // error
*p = 100;   // OK
```

КОНСТАНТНОСТЬ И УКАЗАТЕЛИ

```
int i = 32;

const int* const p = &i; // const pointer to const int

int j = 1;
p = &j;     // error
*p = 100;   // error
```

КОНСТАНТНОСТЬ И УКАЗАТЕЛИ

Все, что слева от `*` относится к типу

Все, что справа от `*` относится к указателю

МАССИВЫ

- Непрерывная последовательность объектов определенного типа: `T a[N]`
- индексация от `0` до `N-1` с помощью `operator[]`

```
int arr1[10]; // not initialized
int arr2[3] = {1, 2, 3};
int arr3[3] = {}; // init with zeros
int arr4[] = {1, 2, 3};

int a = arr[0]; // subscript operator

int mat[3][2] = {{1,1}, {1,1}, {1,1}}; // multidimensional
int b = mat[0][0]; // subscript operator
```

SIZEOF

```c
int arr3[3] = {};
assert(sizeof(arr3) == 3 * sizeof(int));

int arr4[] = {1, 2, 3};
assert(sizeof(arr4) == 3 * sizeof(int));
```

ARRAYS AND POINTERS

Array-to-pointer decay

```
int a[3] = {1, 2, 3};
int* p = a; // implicit cast
```

C-СТРОКИ

`const char*` - тип строкового литерала

```
const char* hello = "hello"; // immutable string
```

C-СТРОКИ

```
const char hello[] = "hello";
std::cout << sizeof(hello); // what is expected?
```

C-СТРОКИ

```
const char hello[] = "hello";

// lifehack: compilation error to get out the type
hello = 3;
// error: cannot assign to variable 'hello'
// with const-qualified type 'const char [6]'
```

C-СТРОКИ

```
const char* hello = "hello";
```

C-строки имеют терминирующий нуль `\0`

Чтобы найти длину строки, нужно посчитать количество символов до `\0`

C-СТРОКИ

```c
char hello[] = "hello"; // mutable string is legal?
```

АРИФМЕТИКА С УКАЗАТЕЛЯМИ

```
int arr[100]{};
int* p = arr;    // p points to arr[0]

// adding an integer
int* q = p + 2;  // q points to arr[2]
int* s = 4 + p;  // s points to arr[4]

// subtract a number
int* a = s - 2;  // a points to arr[2]

// Undefined behavior
int* ub1 = p + 1000; // out of arr
int* ub2 = p - 1;    // out of arr
```

АРИФМЕТИКА С УКАЗАТЕЛЯМИ

```
int arr[20]{};
const int I = 3;
const int J = 10;
int* p = arr + I;
int* q = arr + J;

ptrdiff_t diff1 = p - q; // I - J
ptrdiff_t diff2 = q - p; // J - I

// ptrdiff_t - signed integer type (e.g.: long)

/* Undefined behavior:
1. p and q указывают на объекты из разных массивов
2. результат разности не помещается в ptrdiff_t
```