

# Внутреннее представление данных

Юрий Литвинов  
yurii.litvinov@gmail.com

29.09.2020

# Побитовые операции

- ▶ `&` — побитовое “И”
- ▶ `|` — побитовое “ИЛИ”
- ▶ `~` — побитовое “НЕ”
- ▶ `1 & 2 == false`, но `1 && 2 == true`
- ▶ `<<`, `>>` — битовый сдвиг
  - ▶ `int x = 1 << 3`
- ▶ `sizeof` — размер типа в байтах
  - ▶ `int s = sizeof(int) * 8`
- ▶ Обратите внимание, что ВСЁ хранится как набор бит
  - ▶ “3” — литерал, лишь удобная форма записи 00...0011 в коде

Маски

&	1	1	0	1	1	0	1	0
	0	0	0	0	0	0	0	1
	0	0	0	0	0	0	0	0

&	1	1	0	1	1	0	1	0
	0	0	0	0	0	0	1	0
	0	0	0	0	0	0	1	0

# Работа с масками

```
char x = 5;
```

```
int bit = 0b10000000;
```

```
for (int j = 0; j < 8; ++j)
```

```
{  
    printf((x & bit) ? "1" : "0");
```

```
    bit = bit >> 1;
```

```
}
```

# Целые числа

- ▶ Прямой код
  - ▶ 5 — 00000101, -5 — 10000101
- ▶ Обратный код
  - ▶ 5 — 00000101, -5 — 11111010
- ▶ Дополнительный код
  - ▶ 5 — 00000101, -5 — 11111011
  - ▶  $-x$  представляется как  $2^n - x$ , поэтому и дополнительный
    - ▶  $n$  — разрядность регистра

# Арифметические действия

- ▶ В обратном коде единица переноса в старшем разряде прибавляется к младшему разряду
- ▶ В дополнительном коде единица переноса в старшем разряде отбрасывается

# Формат записи

## ► Литералы

► **int** hexadecimal = 0x35FF;

► **int** octal = 03567;

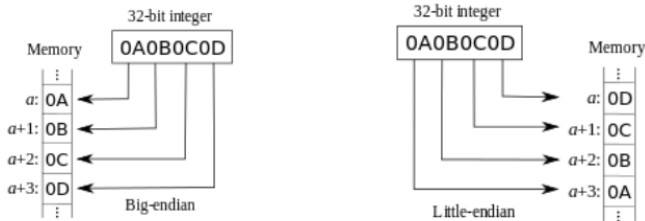
► **int** binary = 0b00100111;

► 0xFF == 255

► **int** x = 239;

**unsigned char** \*b = (**unsigned char**\*)&x;

printf("0x%02X%02X%02X%02X\n", b[0], b[1], b[2], b[3]);

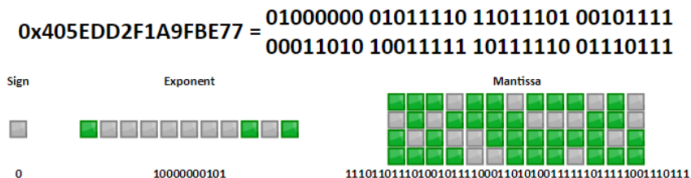


# Вещественные числа

- ▶ IEEE 754 — международный стандарт
- ▶  $x = (+-)m * p^q$ 
  - ▶  $p$  — основание системы счисления
  - ▶  $q$  — порядок числа (целое число)
  - ▶  $m$  — мантисса числа (правильная  $p$ -ичная дробь, у которой первая цифра после запятой не равна 0)
    - ▶ Часто используют нормализованную запись,  $m \in [1, p)$
  - ▶ Например:
    - ▶  $3,1415926 = 0,31415926 * 10^1$
    - ▶  $1000 = 0,1 * 10^4$
    - ▶  $0,123456789 = 0,123456789 * 10^0$
    - ▶  $0,0000107_8 = 0,107_8 * 8^{-4}$
    - ▶  $1000,0001_2 = 0,10000001_2 * 2^4$
    - ▶  $0 = 0,0 * 10^0$

# Внутреннее представление

- ▶ 123.456
- ▶ Наиболее точное представление (IEEE 754 Double, 64 бит):  
1.234560000000000003069544618484E2



- ▶ <http://www.binaryconvert.com/>



# Смещённый порядок

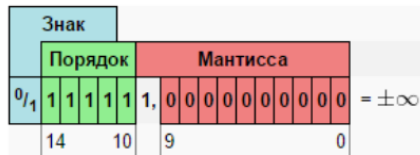
- ▶  $123.456 : q = 10000000101_2 ???$
- ▶ Смещённый порядок  $= 2^{a-1} - 1 + \text{<истинный порядок>}$ 
  - ▶  $a$  — количество разрядов, отводимых под порядок
  - ▶ Чтобы не хранить знак ещё и порядка числа
- ▶  $123.456 \approx 1111011.01110100101111 = 1.11101101110100101111 * 2^6$
- ▶ Смещённый порядок  $= 2^{10} - 1 + 6 = 1029_{10} = 10000000101_2$

# Специальные числа

## ► Неопределённость (NaN):



## ► Бесконечности:



```
double y = 0.0;  
double x = 239.0 / y;  
printf("%f", x);
```

# Строки

Строка как последовательность символов (их кодов) — таблица символов

- ▶ ASCII (American Standard Code for Information Interchange)
  - ▶ 8 бит на символ (0 – 255), 0 – 127 стандартны, 128 – 255 — для локальных алфавитов
  - ▶ Кодовые страницы
    - ▶ cp866
    - ▶ cp1251
    - ▶ koi8-r
    - ▶ ...
- ▶ Unicode

Строка как последовательность байт — кодировка

- ▶ UCS-16BE, UCS16-LE, UTF-8

# Зачем

- ▶ Локализация — перевод программы на другой язык (и под другую культуру)
- ▶ Интернационализация — сделать так, чтобы программу было можно локализовать
- ▶ У однобайтовых кодировок некоторые проблемы с иероглифическими языками
  - ▶ Shift JIS и прочие странные вещи

# Юникод

- ▶ UCS, universal character set
  - ▶ Кодовые позиции — целые числа (U+0000 – U+007F, ...)
  - ▶ Порядка 110 000 кодовых позиций
- ▶ UTF, Unicode transformation format
  - ▶ Кодировки — битовое представление кодов из UCS
- ▶ UTF-8
  - ▶ 0x00000000 – 0x0000007F: 0xxxxxxx
  - ▶ 0x00000080 – 0x000007FF: 110xxxxx 10xxxxxx
  - ▶ 0x00000800 – 0x0000FFFF: 1110xxxx 10xxxxxx 10xxxxxx
  - ▶ 0x00010000 – 0x001FFFFF: 11110xxx 10xxxxxx 10xxxxxx 10xxxxxx
  - ▶ В точности совпадает с ASCII для первых 127 символов
- ▶ BOM (Byte Order Mark)
  - ▶ FE FF, FF FE, EF BB BF