Семинар 2

> инты, endian-ы, флоты/даблы, кодировки, арифметика над ними

Как хранить инты

Как хранить инты: беззнаковые

Тут всё максимально просто и без сюрпризов

```
print("%x\n", (uint8_t)(1));    // 1
print("%x\n", (uint8_t)(7));    // 7
print("%x\n", (uint8_t)(123));  // 7b
print("%x\n", (uint8_t)(255));  // ff

// Пы.Сы.: если вдруг кто забыл - char это int8
```

Как хранить инты: беззнаковые (skip that)

```
// a = 123
// a = 0b????????
//
// a = 123 = 64 + 59 = 64 + 32 + 27
// = 64 + 32 + 16 + 11
// = 64 + 32 + 16 + 8 + 3
// = 64 + 32 + 16 + 8 + 2 + 1
// = (1 << 6) + (1 << 5) + (1 << 4) + (1 << 3) + (1 << 1) + (1 << 0)
// = 0b01111011 = 0b0111 << 4 + 0b1011
// = (0x7 << 4) + b = 7b
```

Как хранить инты: знаковые

Как хранить инты: знаковые

Очевидно, надо где-то хранить знак (thx captan)

Если бы такая задача встала перед нами, мы бы написали что-то такое

```
struct signed_int {
    uint8_t _sign;
    uint8_t _absolute_value;
};
```

Как хранить инты: знаковые

Ну и операции над этим выглядели бы очевидно

```
struct signed_int {
    uint8_t _sign;
    uint8_t _absolute_value;
};

void sum(struct signed_int *out, struct signed_int *left, struct signed_int *right) {
    if (left->_sign == right->_sign) {
        // ...
    } else {
        // ...
    }
}

void prod(struct signed_int *out, struct signed_int *left, struct signed_int *right) {
    out->_sign = left->_sign ^ right->_sign;
    ...
}
```

Как хранить инты: знаковые

Потом, спустя пару ~~дней~~ часов мы бы догадались до такой оптимизации

```
struct signed_int {
    uint8_t _sign : 1;
    uint8_t _absolute_value : 7;
};
```

Теперь у нас `sizeof(struct signed_int) = sizeof(uint8_t) = 1`

Как хранить инты: знаковые

В таком представлении есть несколько проблем

1. У нас два нуля
2. Найдите

```
struct signed_int {
    uint8_t _sign : 1;
    uint8_t _absolute_value : 7;
};

// Викторина: что вам тут не нравится?)
void sum(struct signed_int *out, struct signed_int *left, struct signed_int *right) {
    if (left->_sign == right->_sign) {
        // ...
    } else {
        // ...
    }
}
```

Как хранить инты: знаковые

В таком представлении есть несколько проблем

1. У нас два нуля
2. Найдите

```
struct signed_int {
    uint8_t _sign : 1;
    uint8_t _absolute_value : 7;
};

// Викторина: что вам тут не нравится?)
void sum(struct signed_int *out, struct signed_int *left, struct signed_int *right) {
    if (left->_sign == right->_sign) {  // <-- маааааленький намёк
        // ...
    } else {
        // ...
    }
}
```

Как хранить инты: знаковые

IF - это плохо

процессор не очень любит ветвления, потому что он может не угадать ветвь

и из-за этого ему надо будет 'перезапускать конвеер'

а это оооооочень плохо, т.к. **операции сложения и вычитания сильно более частые, чем умножения**

Как хранить инты: знаковые

Давайте попробуем 'соптимизировать' именно сложение / вычитание

Вспоминаем теорию групп: 0

- $a + (b + c) = (a + b) + c$
- $\exists 0 : a + 0 = 0 + a = a$
- $\forall a \exists (-a) : a + (-a) = (-a) + a = 0$

Как хранить инты: знаковые

Давайте попробуем 'соптимизировать' именно сложение / вычитание

Вспоминаем теорию групп: 0

- ...
- ...
- $\forall a \exists (-a) : a + (-a) = (-a) + a = 0$

Как хранить инты: знаковые

Матаметика ~~- это моя жизнь~~ аксиоматически подарила нам тождество

Давайте для примера наёдем чило (-21) в $\mathbb{Z}_{256}$ (потому что 8 бит)

```
21 + (-21) = 0
```

```
21 = 0x15 = 0b00010101
```

Кажется очевидно, что

```
0b00010101 + ~0b00010101 = 0b00010101 + 0b11101010 = 0b11111111
```

Теперь, чтоб получить `0`, остается добавить `1`

```
0b00010101 + ~0b00010101 + 1 = 0
```

Т.е. `-21 = ~(21) + 1`

Как хранить инты: знаковые

Т.е. если у нас есть число `a`, то `-a := ~a + 1`

(и в обратную сторону это тоже работает)

- `0 = -0` (внезапно)
- `-1 = 0xff -> reinterpret_cast<unsinged char> -> 255`
- `-2 = 0xfe -> reinterpret_cast<unsinged char> -> 254`
- `-3 = 0xfd -> reinterpret_cast<unsinged char> -> 253`
- `...`

Ну и получаем какой-то такой ряд

`-128..-1,0,1..127`

Как хранить инты: знаковые

Ну и `-a := ~a + 1` обладает рядом плюсов

- первый бит отвечает за знак (познайте дзен, это почти очевидно)
- простая операция умножения на -1
- операции `сумма` / `разность` не меняются
- умножение можно перезаписать, используя свойства кольца
  - `b * (-a) = b * (0 - a) = b * (0xff + 1 - a) = b * (0xff - a) + b`

Как хранить инты: endian

> как хранить инты НЕ в регистрах (в RAM)

Как хранить инты: endian

```c
#include <stdio.h>
#include <stdint.h>

int main() {
    union {
        int32_t number;
        uint8_t bytes[4];
    } u;

    u.number = 0x89abcdef;
    for (int i = 0; i < 4; ++i)
        printf("%x ", u.bytes[i]);
}
```

```
$ ./a.out
? что тут будет ?
```

Как хранить инты: endian

```c
#include <stdio.h>
#include <stdint.h>

int main() {
    union {
        int32_t number;
        uint8_t bytes[4];
    } u;

    u.number = 0x89abcdef;
    for (int i = 0; i < 4; ++i)
        printf("%x ", u.bytes[i]);
}
```

```
$ ./a.out
ef cd ab 89
```

Как хранить инты: endian

```c
#include <stdio.h>
#include <stdint.h>

int main() {
    union {
        int32_t number;
        uint8_t bytes[4];
    } u;

    u.number = 0x89abcdef;
    for (int i = 0; i < 4; ++i)
        printf("%x ", u.bytes[i]);
}
```

```
$ ./a.out
ef cd ab 89
```

what ???

Как хранить инты: endian

```
// т.е. мы написали
u.number = 0x89abcdef;

// и получили
u.bytes[4] = { 0xef, 0xcd, 0xab, 0x89 };

// хотя (возможно) хотели что-то такое
u.bytes[4] = { 0x89, 0xab, 0xcd, 0xef };
```

есть идеи, почему так и отчего это зависит?

Как хранить инты: endian

Правильный ответ - от архитектуры компьютера

(правда вы сейчас не найдете комьютеры с big-endian)

```
/* Little Endian
/  0x89abcdef  -->  0xef 0xcb 0xab 0x89
/
/  Big Endian
/  0x89abcdef  -->  0x89 0xab 0xcd 0xef
*/
```

Как хранить инты: endian

А почему так нелогично?...

```
/* Little Endian
/  0x89abcdef  -->  0xef 0xcb 0xab 0x89
/
/  Big Endian
/  0x89abcdef  -->  0x89 0xab 0xcd 0xef
*/
```

Как хранить инты: endian

А почему так нелогично?...

На самом деле это очень полездно, вот пример:

```
uint32_t u32 = 0x89abcdef

uint8_t u8 = *((uint8_t*)(&u32))
// u8 = ???
```

Как хранить инты: endian

А почему так нелогично?...

На самом деле это очень полездно, вот пример:

```
uint32_t u32 = 0x89abcdef

uint8_t u8 = *((uint8_t*)(&u32))
// u8 = 0x89
```

Как хранить инты: endian

А почему так нелогично?...

На самом деле это очень полездно, вот пример:

```
uint32_t u32 = 0x89abcdef

uint8_t u8 = *((uint8_t*)(&u32))
// u8 = 0xef
```

А зачем?)

Как хранить инты: endian

Ладно, вот хороший пример

```
// u32 = 0x000000ef
uint32_t u32 = 0x89abcdef - 0x89abcd00

uint8_t u8 = *((uint8_t*)(&u32))
```

Немного про кодировки

Немного технологий мамонтов, дошедших до наших дней

Немного про кодировки

Немного технологий мамонтов, дошедших до наших дней

Unicode

Немного про кодировки

Немного технологий мамонтов, дошедших до наших дней

Unicode

- появился давно
  - вот динозавры вымерли и сразу появился Unicode (~1990г)
- использует для хранения знаков 2 байта
- совместим с *основными символами* ASCII (старший байт = 0)

Немного про кодировки

Ну и во времена мамонтов были компьютеры с разной архитектурой процессора

Был и Big-endian и Little-endian

Вопрос:

> как читать на компьютере Big-endian фалй, записаный на компьютере с Little-endian (и наоборот)?

Немного про кодировки

Вопрос:

> как читать на компьютере Big-endian фалй, записаный на компьютере с Little-endian (и наоборот)?

Ответ:

> файл в кодировке Unicode начинался с символа `0xfffe` (или `0xfeff`) - aka Byte Order Mark (BOM)

Немного про кодировки

Вопрос:

> как читать на компьютере Big-endian фалй, записаный на компьютере с Little-endian (и наоборот)?

Ответ:

> файл в кодировке Unicode начинался с символа `0xfffe` (или `0xfeff`)
> aka Byte Order Mark (BOM)

(5 секунд на осознание)

Немного про кодировки

> А как сейчас дела?

Сейчас стандарты кодировок (с длиной символа > 1 байта) тоже имеют такие маркировки в начале
По этим маркировкам

- можно понять, что это за файл (тип файла)
- на какой архитектуре процессора оно записывалось

Немного про кодировки

> А как сейчас дела?

Сейчас стандарты кодировок (с длиной символа > 1 байта) тоже имеют такие маркировки в начале
По этим маркировкам

- можно понять, что это за файл (тип файла)
- на какой архитектуре процессора оно записывалось

Но Little-endian сейчас правит миром, поэтому почти всегда можно забить))

И что, теперь Big-endian нигде не используется?

И что, теперь Big-endian нигде не используется?

Используется в разных сетевых протаколах (будем проходить позже)

И для этого байты в регистре разворачивают

И что, теперь Big-endian нигде не используется?

Используется в разных сетевых протаколах (будем проходить позже)

И для этого байты в регистре 'разворачивются'

'разворачивются' не данные, а всякая фигню для заголовков пакетов

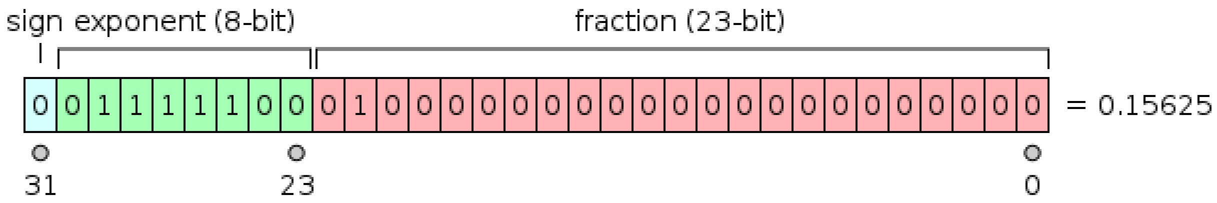
флоты/даблы

флоты/даблы

если вдруг помните перевод дробных чисел в двоичную систему счисления - забудьте)

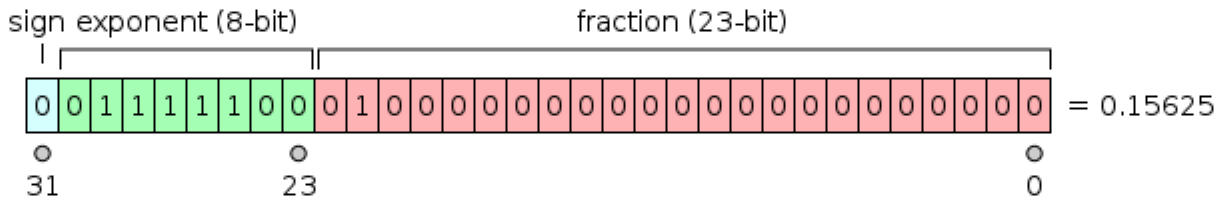
флоты/даблы: как хранить

Если немного погуглить, то вы найдете такую картинку



флоты/даблы: как хранить

Если немного погуглить, то вы найдете такую картинку



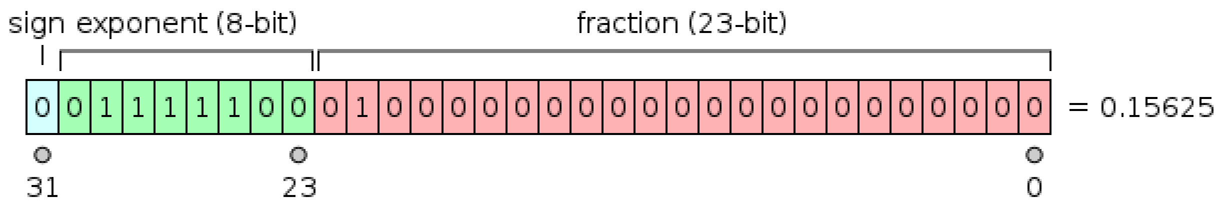
Вот эта штука регулируется стандартом **IEEE 754**

В Си мы можем использовать

- `float` - **IEEE 754 Single Precision**
- `double` - **IEEE 754 Double Precision**

флоты/даблы: как хранить

Если немного погуглить, то вы найдете такую картинку



Вот эта штука регулируется стандартом **IEEE 754**

В Си мы можем использовать

- `float` - **IEEE 754 Single Precision**
- `double` - **IEEE 754 Double Precision**

Хотя можно найтий `float16`, `float32`, `float64`, `float128`, `float256`

флоты/даблы: как хранить

НЛО (небольшое лирическое отступление)

`3.1415926` - десятичная форма записи

`31415926 * 10^(-7)` - степенная(?) форма записи

`31415926` - мантисса

`-7` - экспонента

`31415926e-7` - экспоненциальная форма записи

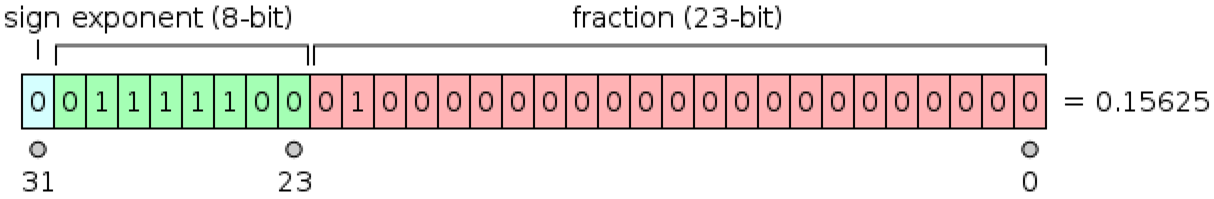
`31415926e-7` - экспоненциальная форма записи

А ещё основание экспоненты может быть любым

`0b1101 * 2^3`
`0b1101 << 3`

флоты/даблы: как хранить

Ладно, давайте разбираться на примере `float32` aka `float`



- `S` - sign
- `E` - exponent
- `M` - mantissa

Ну и общая формула какая-то такая (конкретизируем позже)

флоты/даблы: как хранить

Пример

`5 = 0b101 << 0`

`5 = 0b1010 << -1`

`5 = 0b10100 << -2`

`5 = 0b101000 << -3`

...

флоты/даблы: как хранить

Пример

`5 = 0b101 << 0`

`5 = 0b1010 << -1`

`5 = 0b10100 << -2`

`5 = 0b101000 << -3`

...

и что делать?

флоты/даблы: как хранить

`5 = 0b101 << 0`

`5 = 0b1010 << -1`

`5 = 0b10100 << -2`

`5 = 0b101000 << -3`

...

Вариантов на самом деле всего два

1. это всё равнозначные представления числа `5f`
2. какое-то из них является 'стандартным'

флоты/даблы: как хранить

`5 = 0b101 << 0`

`5 = 0b1010 << -1`

`5 = 0b10100 << -2`

`5 = 0b101000 << -3`

...

Вариантов на самом деле всего два

1. это всё равнозначные представления числа `5f`
2. какое-то из них является 'стандартным

Короче ответ (2), но не совсем (поговорим про вычисления позже)

флоты/даблы: как хранить

Правильно число 5 будем хранить так

`5 = 0b101 << 0`

флоты/даблы: как хранить

Правильно число 5 будем хранить так

`5 = 0b101 << 0`

Но смотрите ещё на один фокус

`6 = 0b110 << 0`

`7 = 0b111 << 0`

`8 = 0b1000 << 0`

`9 = 0b1001 << 0`

флоты/даблы: как хранить

Правильно число 5 будем хранить так

`5 = 0b101 << 0`

Но смотрите ещё на один фокус

`6 = 0b110 << 0`

`7 = 0b111 << 0`

`8 = 0b1000 << 0`

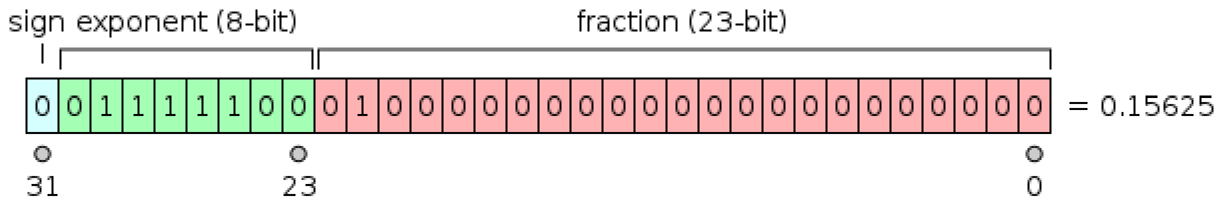
`9 = 0b1001 << 0`

У нас в каждом числе $\neq 0$ есть первый $\neq 0$ бит (логично)

Так давайте *не будем хранить этот бит, т.к. он всегда единица*

флоты/даблы: как хранить

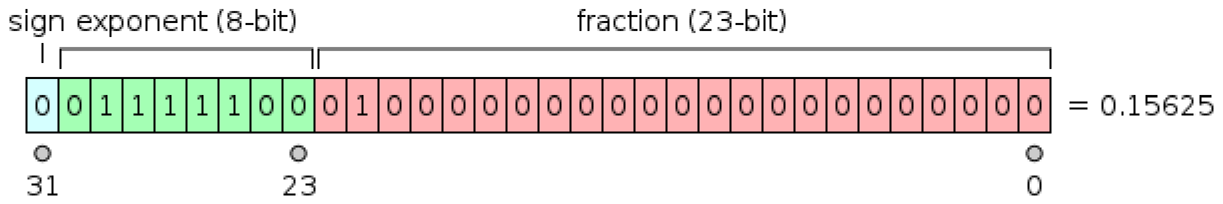
Теперь записываем ультимативную настоящую формулу



$$x = (-1)^{b_{31}} \times 2^{(b_{30}b_{29}\ldots b_{23})_2 - 127} \times (1b_{22}b_{21}\ldots b_0)_2$$

флоты/даблы: как хранить

Теперь записываем ультимативную настоящую формулу



$$x = (-1)^{b_{31}} \times 2^{(b_{30}b_{29}\dots b_{23})_2 - 127} \times (1b_{22}b_{21}\dots b_0)_2$$

А откуда взялось `-127`?

флоты/даблы: арифметика

> сейчас всё объясню за один слайд

флоты/даблы: арифметика

> сейчас всё объясню за один слайд
>
> готовы

флоты/даблы: арифметика

$31415 \times 10^{-4} + 27 \times 10^{-1}$ = ?

флоты/даблы: арифметика

$31415 \times 10^{-4} + 27 \times 10^{-1}$ =

$31415 \times 10^{-4} + 27000 \times 10^{-4}$ =

$(31415 + 27000) \times 10^{-4}$ =

58415×10^{-4}

флоты/даблы: специальные значения

type	sign	E	M
QNaN	x	11...1	1xx...
SNaN	x	11...1	0xx...
+inf	0	11...1	0
-inf	1	11...1	0
+zero	0	0	0
-zero	1	0	0
normal	x	not all 1 or 0	xx...x
denormalized	x	00...0	xx...x

флоты/даблы: специальные значения

type	sign	E	M
QNaN	x	11...1	1xx...
SNaN	x	11...1	0xx...
+inf	0	11...1	0
-inf	1	11...1	0
+zero	0	0	0

кажется у вас есть задача на это, поэтому

- гугл в помощь
- math.h вам тоже поможет)

Всё)