# Семинар #7: Память и бинарные файлы.

## Системы счисления

Мы привыкли пользоваться десятичной системой счисления и не задумываемся, что под числом в десятичной записи подразумевается следующее:

$$123.45_{10} = 1 \cdot 10^2 + 2 \cdot 10^1 + 3 \cdot 10^0 + 4 \cdot 10^{-1} + 5 \cdot 10^{-2}$$

Конечно, в числе 10 нет ничего сильно особенного с математической точки зрения. Оно было выбрано исторически, скорей всего по той причине, что у человека 10 пальцев. Компьютеры же работают с двоичными числами, потому что оказалось что процессоры на основе двоичной логики сделать проще. В двоичной системе счисления есть всего 2 цифры: `0` и `1`. Под записью числа в двоичной системе подразумевается примерно то же самое, что и в десятичной:

$$101.01_2 = 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2} = 5.25_{10}$$

При работе с компьютером на низком уровне имеет смысл использовать двоичную систему за место десятичной. Но человеку очень сложно воспринимать числа в двоичной записи, так как они получаются слишком длинными. Поэтому популярность приобрели восьмеричная и шестнадцатиричная системы счисления. В шестнадцатиричной системе счисления есть 16 цифр: `0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f`.

$$1a.8_{16} = 1 \cdot 16^1 + 10 \cdot 16^0 + 8 \cdot 16^{-1} = 26.5$$
$$1ab_{16} = 1 \cdot 16^2 + 10 \cdot 16^1 + 11 = 427$$
$$ff.c_{16} = 15 \cdot 16^1 + 15 \cdot 16^0 + 12 \cdot 16^{-1} = 255.75$$

### Шестнадцатиричная, восьмеричная и бинарная системы счисления в языке `C`

Язык `C` поддерживает шестнадцатиричные, восьмеричные и бинарные литералы. Чтобы получить шестнадцатиричное число нужно написать `0x` перед числом. Чтобы получить восьмеричное число нужно написать `0` перед числом. Чтобы получить бинарное число нужно написать `0b` перед числом.

```c
#include <stdio.h>
int main()
{
    int a = 123;      // Десятичная система
    int b = 0x123;    // Шестнадцатиричная система
    int c = 0123;     // Восьмеричная система
    int d = 0b10101;  // Бинарная система
    printf("%i %i %i %i\n", a, b, c, d);
}
```

Также, можно печатать и считывать числа в других системах счисления с помощью спецификаторов `%x` (для шестнадцатеричной – he**x**adecimal) и `%o` (для восьмеричной системы – **o**ctal). Спецификатор `%d` можно использовать для десятичной системы – **d**ecimal (получается, что спецификатор `%d` это то же самое, что и `%i`). Для отображения адресов при печати с помощью спецификатора `%p` используется шестнадцатеричная система счисления.

```c
#include <stdio.h>
int main()
{
    int a;
    scanf("%d", &a);    // Считаем число в десятичной системе
    printf("%x\n", a);  // Напечатает это же число в шестнадцатеричной системе

    printf("%p\n", &a); // Для адресов используется шестнадцатеричная система
}
```

# Представление чисел в памяти

## Представление целых чисел в памяти

Положительные числа представляются в памяти в соответствии с их записью в бинарной системе счисления. Для представления отрицательных чисел в памяти используется способ, который называется дополнительный код. Однобайтовые числа представляются в памяти следующим образом:

| unsigned char | | signed char | |
|---|---|---|---|
| 0 | 00000000 | 0 | 00000000 |
| 1 | 00000001 | 1 | 00000001 |
| 2 | 00000010 | 2 | 00000010 |
| 3 | 00000011 | ... | |
| 4 | 00000100 | 126 | 01111110 |
| 5 | 00000101 | 127 | 01111111 |
| 6 | 00000110 | -128 | 10000000 |
| ... | | -127 | 10000001 |
| 253 | 11111101 | ... | |
| 254 | 11111110 | -2 | 11111110 |
| 255 | 11111111 | -1 | 11111111 |

Целые числа большего размера представляются в памяти аналогичным образом.

## Представление чисел с плавающей точкой в памяти

Разберём как числа с плавающей точкой хранятся в памяти на примере. Пусть у нас есть число 123.456 типа `float`. Как это число хранится в двоичном коде? Для начала переведём это число из десятичной системы счисления в двоичную:

$$123.456_{10} = 1111011.01110100101111000111_2$$

Затем представим это число в научной записи:

$$1111011.01110100101111000111 = 1.11101101110100101111000111 \cdot 2^6$$

Части этой записи числа и хранятся в памяти. Число типа `float` имеет размер 4 байта или 32 бита. Из них:

- 1 бит приходится на знак числа. 0 – для положительных и 1 для отрицательных.
- 8 бит приходится на степень двойки. К степени двойки в двоичной научной записи прибавляется число 127, а затем это число хранится в этих битах. В нашем примере будет храниться число $6+127 = 133 = 10000101_2$.
- 23 бита приходится на мантиссу. В двоичной научной записи это просто 23 знака после точки. В нашем примере это 11101101110100101111001 (округляем последний бит).

Таким образом число 123.456 типа `float` будет хранится в памяти как:

$$0 \quad 10000101 \quad 11101101110100101111001$$

Разобъем эту запись на кусочки по 8 бит:

$$01000010 \quad 11110110 \quad 11101001 \quad 01111001$$

Переведём каждый из кусочков в шестнадцатеричную систему:

$$42 \quad F6 \quad E9 \quad 79$$

Это значения которые будут иметь байты числа типа `float` при записи в него числа 123.456. Числа типа хранятся аналогичным образом, но для хранения степени и мантиссы используется 11 и 52 бита соответственно.

## Побитовые операторы

### Печать битового представления числа

```
#include <stdio.h>
#define PRINT_BINARY(prefix, x) {\
    printf("%s", prefix);\
    for (int i = 8 * sizeof(x) - 1; i >= 0; --i)\
    {\
        printf("%llu", (unsigned long long)((x) >> i) & 1);\
        if (i % 8 == 0)\
            printf(" ");\
    }\
    printf("\n");\
}\

int main (void) {
    unsigned int a = 0b10110101;
    unsigned int b = 0b00101110;
    PRINT_BINARY("a     = ", a);
    PRINT_BINARY("b     = ", b);
    PRINT_BINARY("a & b = ", a & b);
    PRINT_BINARY("a | b = ", a | b);
    PRINT_BINARY("a ^ b = ", a ^ b);
    PRINT_BINARY("~a    = ", ~a);
    return 0;
}
```

### Побитовое И

### Побитовое ИЛИ

### Побитовое исключающее ИЛИ

### Побитовое НЕ

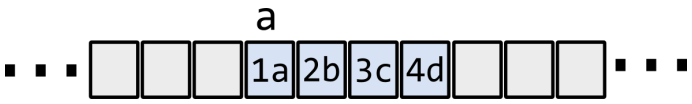
### Побитовый сдвиг влево

### Побитовый сдвиг вправо

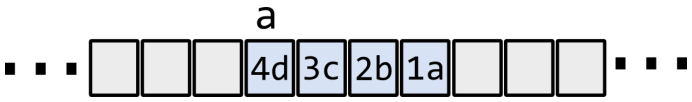
### Порядок байт. Little и Big Endian

То в каком порядке лежат байты многобайтового числа в памяти может различаться на разных системах. Различают два основных порядка байт. Разберём их на примере числа: `int a = 0x1a2b3c4d;`

- Прямой порядок байт или *Big Endian*
  При таком порядке записи, число записывается в памяти от старшего байта к младшему.

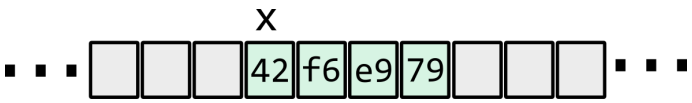



- Обратный порядок байт или *Little Endian*
  При таком порядке записи, число записывается в памяти от младшего байта к старшему.

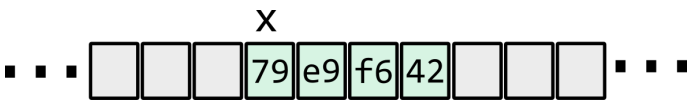



То же самое работает и для других скалярных типов данных, таких как числа с плавающей точкой и указатели. Рассмотрим число с плавающей точкой `float x = 123.456`. В памяти оно будет выглядеть следующим образом:

- При использовании порядка байт *Big Endian*:




- При использовании порядка байт *Little Endian*:



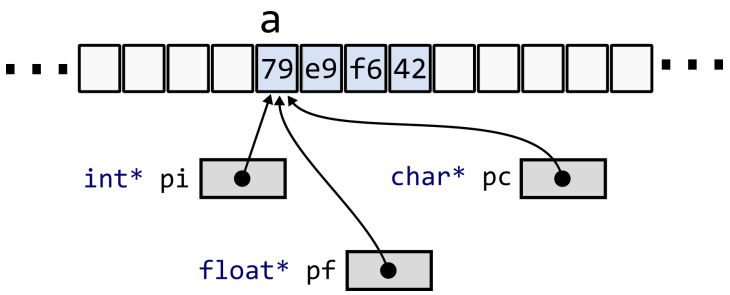

На большинстве систем используется порядок байт Little Endian. В дальнейших примерах по умолчанию будет использоваться этот порядок.

## Указатели разных типов, указывающие на одно и то же место в памяти

Рассмотрим следующий пример. На переменную `a` указывают две переменные разных типов: `int*`, `float*` и `char*`. Оба указателя хранят одно и то же значение, но работают по разному при разыменовании.

```
#include <stdio.h>
int main()
{
    int a = 0x42f6e979;
    int* pi = &a;
    float* pf = (float*)&a;
    char* pc = (char*)&a;

    printf("%x\n", *pi);
    printf("%f\n", *pf);
    printf("%c\n", *pc);
}
```




**Задача: Что напечатает следующая программа и почему она это напечатает?**

```
#include <stdio.h>
int main()
{
    int a = 7627075;
    char* p = (char*)&a;
    printf("%s\n", p);
}
```

### Просмотр байт переменной

Просмотреть, что содержится в байтах какого-либо объекта можно с помощью указателя на `unsigned char`.

```
#include <stdio.h>
int main()
{
    int a = 0x42f6e979;

    unsigned char* p = (unsigned char*)&a;
    for (size_t i = 0; i < sizeof(a); ++i)
        printf("%x ", *(p + i));
    printf("\n");
}
```

### Уровни оптимизации

### Правила строгого алиасинга (*strict aliasing rule*)

**Стандартные функции** `memset`, `memcpy` **и** `memmove`.

# Работы с бинарными файлами

`fwrite` записывает некоторый участок памяти в файл без обработки.
`fread` считывает данные из файла в память без обработки.

Пример. Записываем 4 байта памяти переменной `a` в файл `binary.dat`:

```
#include <stdio.h>
int main()
{
    int a = 0x11223344;
    FILE* fb = fopen("binary.dat", "wb");
    fwrite(&a, sizeof(int), 1, fb);
    fclose(fb);
}
```

- **Печать в текстовом и бинарном виде:**
  В файле `text_and_binary.c` содержится пример записи числа в текстовом и бинарном виде. Скомпилируйте эту программу и запустите. Должно появиться 2 файла (`number.txt` и `number.bin`). Изучите оба эти файла, открывая их в текстовом редакторе, а также с помощью утилиты `xxd`. Объясните результат.

- **Печать массива в бинарном виде:**
  Пусть есть массив из чисел типа `int`: `int array[5] = {111, 222, 333, 444, 555};`
  Запишите эти числа в текстовый файл `array.txt`, используя `fprintf`. Изучите содержимое этого файла побайтово с помощью `xxd`.
  Запишите эти числа в бинарный файл `array.bin`, используя `fwrite`. Изучите содержимое этого файла побайтово с помощью `xxd`.

### Функция `fgetc`.

Функция `fgetc` считывает 1 символ и возвращает код `ASCII` символа или `EOF` если дошли до конца файла (`EOF` это просто константа равная -1). Пример считывания:

```
#include <stdio.h>
int main()
{
    FILE* f = fopen("test.txt", "r");
    while (1)
    {
        // Считываем 1 символ
        int c = fgetc(f);

        // Если он равен EOF, то выходим из цикла
        if (c == EOF)
            break;

        printf("%c\n", c);
        }
    fclose(f);
}
```

- Напишите программу, которая печатает количество строк в файле.
- Напишите программу, которая печатает размер самой длинной строки файла.

### Функции `ftell` и `fseek`.

Процесс считывания файла можно представить как перемещение по набору байт. При открытии файла указатель положения равен нулю. При считывании он увеличивается на количество считанных байт.


Однако, положение в файле можно менять и без считывания при помощь функции `fseek`:

`fseek(<файловый указатель>, <смещение>, <начало отсчёта>)`

Начало отсчёта в этой функции может принимать 3 значения:

1. `SEEK_SET` – отсчитывать от начала файла
2. `SEEK_CUR` – отсчитывать от текущего положения
3. `SEEK_END` – отсчитывать от конца файла

Например:

```
#include <stdio.h>
int main()
{
    FILE* f = fopen("test.txt", "r");
```

```
    fseek(f, 10, SEEK_SET); // Перемещаемся на 11 - й символ
    fseek(f, -1, SEEK_END); // Перемещаемся к последнему символу

    fseek(f, -1, SEEK_CUR); // Перемещаемся на 1 символ назад
    fseek(f, 0, SEEK_SET);  // Возвращаемся к началу
    fclose(f);
}
```

Функция `ftell(<файловый указатель>)` возвращает целое число – текущее положение в файле.

- Написать программу, которая будет печатать 3 последних символа в файле.

- Написать программу, которая будет считывать файл `test.txt` и печатать число, которое начинается с 10-го символа.

- Написать программу, которая будет принимать название файла через аргумент командной строки и печатать его размер в байтах.
  *Подсказка:* Используйте `fseek`, чтобы перейти в конец файла и `ftell`, чтобы узнать позицию.

- В файле `numbers.txt` хранятся некоторые целые числа (но не указано их количество). Напишите программу, которая будет считывать все числа из этого файла и печатать их на экран. Есла в файле содержится какие-то другие символы кроме цифр и пробельных символов, то программа должна печатать `Error!` и завершаться.
  *Подсказка:* Для начала нужно узнать количество чисел. Это можно сделать, используя `fgetc`. Затем считываем. Память для чисел выделяем в куче, так как их количество изначально неизвестно и может быть болишим.

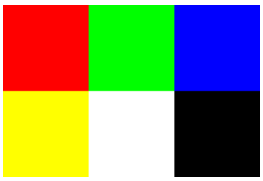
# Работа с изображениями формата `.ppm`

Простейший формат для изображение имеет следующую структуру

```
P3
3 2
255
255 0 0
0 255 0
0 0 255
255 255 0
255 255 255
0 0 0
```

- В первой строке задаётся тип файла `P3` - означает, что в этом файле будет храниться цветное изображение, причём значения пикселей будет задаваться в текстовом формате.
- Во второй строке задаются размеры картинки - 3 на 2 пикселя.
- Во третьей строке задаётся максимальное значение RGB компоненты цвета.
- Дальше идут RGB компоненты цветов каждого пикселя в текстовом формате.

Картинка имеет следующий вид:



## Задачи

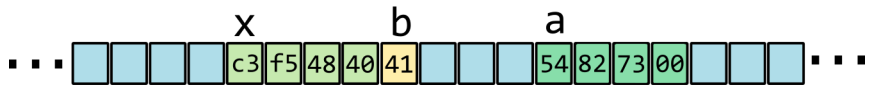
- Написать программу, которая генерирует одноцветную картинку (500 на 500) в формате `.ppm`. Цвет должен передаваться через аргументы командной строки.
- **Белый шум:** Написать программу, которая случайное изображение в формате `.ppm`. Цвет каждого пикселя задаётся случайно.
- **Градиент:** Написать программу, которая генерирует градиентную картинку в формате `.ppm`. Два цвета должны передаваться через аргументы командной строки.
- **Черно-белая картинка:** Написать программу, которая считывает изображение в формате `.ppm` и сохраняет его в черно-белом виде. Файл изображения должен передаваться через аргументы командной строки. Считайте файл `russian_peasants_1909.ppm` и сделайте его черно-белым.

## Работа с изображениями формата `.jpeg`

## Представление чисел в памяти

Положение любой переменной в памяти характеризуется двумя числами: её адресом(номером первого байта этой переменной) и её размером. Рассмотрим ситуацию, когда были созданы 3 переменные типов `int` (размер 4 байта), `char` (размер 1 байт) и `float` (размер 4 байта). На рисунке представлено схематическое расположение этих переменных в памяти (одному квадратику соответствует 1 байт):

```
int a = 0x738254;
char b = 'A';
float x = 3.14;
```




Какие выводы можно сделать из этого изображения:

- Значение одного байта памяти удобно представлять двузначным шестнадцатиричным числом.
- Каждая переменная заняла столько байт, чему равен её размер.
- Переменные в памяти могут хранится не в том порядке, в котором вы их объявляете.
- Переменные в памяти хранятся не обязательно вплотную друг к другу.
- Байты переменных `a` и `b` хранятся в обратном порядке. Такой порядок байт называется `Little Endian`. Обратите внимание, что обращается только порядок байт, а не бит. Большинство компьютеров применяют именно такой порядок байт. Но в некоторых системах может использоваться обычный порядок байт – `Big Endian`. Обратный порядок байт применяется не только к типу `int`, но и ко всем базовым типам.
- Переменная `b` хранит ASCII-код символа `A`. Он который равен $65 = 41_{16}$.