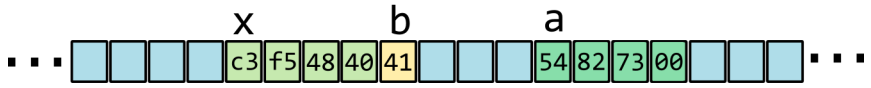
Семинар #9: Файлы и изображения. Классные задания.

Часть 1: Переменные в памяти. Little и Big Endian

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 $65 = 41_{16}$.

Часть 2: Просмотр байт

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 = 0x11223344;

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

Задача:

- Напечатайте байты объекта `a` типа `double`.

  ```
  double a = 123.456;
  ```

- Напечатайте байты объекта `b` типа `int`.

  ```
  int b = -1;
  ```

- Напечатайте байты объекта `c` типа `struct cat`.

  ```
  struct cat
  {
      char first;
      int second;
  };

  int main()
  {
      struct cat c = {0x50, 0x12345678}
  }
  ```

Часть 3: Работа с памятью. Стандартные функции `memset`, `memcpy` и `memmove`.

Часть 4: Работы с бинарными файлами `fread` и `fwrite`

`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`:

```c
#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`.

Часть 5: Работа с файлами. Функции `fgetc`, `fseek`, `ftell`.

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:

```c
#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:

```c
#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.

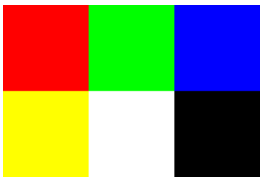
Часть 6: Работа с изображениями формата `.ppm`

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.

Часть 4: Работа с изображениями формата `.jpeg`