

Семинар #6: Файлы и изображения. Классные задания.

Часть 1: 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, скорее 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 шестнадцатеричная системы счисления. В шестнадцатеричной системе счисления есть 16 цифр: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f.

$$1a.8_{16} = 1 \cdot 16^1 + 10 \cdot 16^0 + 8 \cdot 16^{-1} = 26.5_{10}$$

Задача. Переводите следующие числа в десятичную систему:

- | | | |
|-------------|--------------|--------------|
| – 11011_2 | – $2b_{16}$ | – 40_8 |
| – 1.1_2 | – $a.c_{16}$ | – 10_{123} |

Шестнадцатеричная и восьмеричная системы в языке C:

Язык C поддерживает шестнадцатеричные и восьмеричные числа. Чтобы получить восьмеричное число нужно написать 0 перед числом. Чтобы получить шестнадцатеричное число нужно написать 0x перед числом.

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

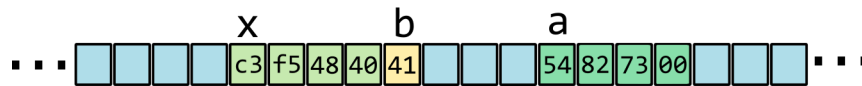
Также, можно печатать и считывать числа в этих системах счисления с помощью спецификаторов %o (для восьмеричной системы – octal) и %x (для шестнадцатеричной – hexadecimal). Спецификатор %d можно использовать для десятичной системы – decimal. Пример программы, которая считывает число в шестнадцатеричной системе и печатает в десятичной:

```
#include <stdio.h>
int main()
{
    int a;
    scanf("%x", &a);
    printf("%d\n", a);
}
```

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 храниться 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 $65 = 41_{16}$.

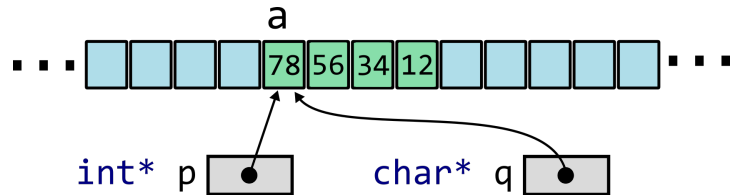
Указатели разных типов

Как вы могли заметить тип указателя зависит от типа элемента на который он указывает. Но все указатели, независимо от типа, по сути хранят одно и то же (адрес первого байта переменной). Чем же они различаются друг от друга? Разница проявляется как раз при их разыменовывании. Например, при разыменовывании указатель `int*` берёт 4 байта и воспринимает их как переменную типа `int`, а указатель `char*` берёт 1 байт и воспринимает его как переменную типа `char`.

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

```
#include <stdio.h>
int main()
{
    int a = 0x12345678;
    int* p = &a;
    char* q = &a;

    printf("%p %p\n", p, q);
    printf("%x\n", *p);
    printf("%x\n", *q);
}
```



Преобразование типов указателя

В предыдущем примере есть такая строка `char* p = &a`; Необычность этой строки в том, что слева и справа от знака `=` находятся объекты разных типов. Слева – `char*`, а справа – `int*`. В этот момент происходит неявное преобразование типов один тип указателя преобразуется в другой. Это всё похоже на преобразование типов обычных переменных.

```
int a = 4.1;           // Неявное преобразование из double в int
int b = (int)4.1;      // Явное преобразование из double в int

char* p = &a;          // Неявное преобразование из int* в char* ( не работает в C++ )
char* p = (char*)&a;    // Явное преобразование из int* в char*
```

Надо отметить, что язык C++ строже относится к соблюдению типов, чем язык C, и не позволит вам неявно преобразовать указатель одного типа в указатель другого типа.

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);
}
```

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`:

```
#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:

```
#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 помощи 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. Если 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 большим.

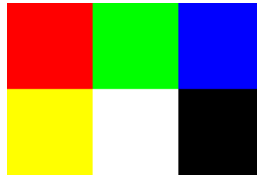
Часть 6: Работа с изображениями формата .ppm

Простейший формат для изображения 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