Семинар #14: Безопасность и работа с файлами. Классные задачи.

Проверка на ошибки. Переменная `errno` и функция `perror`.

При работе с функциями, которые взаимодействуют с операционной системой, такие как `malloc` или `fopen` нужно всегда предусматривать возможность возникновения ошибки. Так например, при вызове `malloc` заправшиваемого объёма памяти может не найтись или при вызове `fopen` файл с нужным названиям может не существовать. В этих случаях функции возвращают значени `NULL`.

```
#include <stdio.h>
int main()
{
        FILE* file = fopen("input.txt", "r");
        if (file == NULL)
        {
                perror("Error");
                exit(1);
        }
        // Работаем с указателем file
        fclose(file);
}
```

При этом ошибка может быть разной. Например, при открытии файла ошибки могут быть такими: нет такого файла; неправильный режим открытия; файл есть, но нет прав доступа для работы с ним; файл является директорией и другие. Чтобы понять какая именно ошибка случилась, можно воспользоваться переменной `errno`, которая хранит в себе код последней ошибки. Пример в файле `0errno.c`. Если нужно просто распечатать сообщение о последней ошибке, то можно воспользоваться функцие `perror`. Пример в файле `1perror.c`.

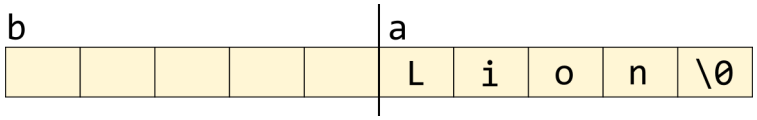
- Напишите программу, которая будет проверять существует ли файл. (Если файл не существует, то переменная `errno` будет равна константе `ENOENT`).

Опасность функций семейства `scanf` при считывании строк.

Функции `scanf`, `fscanf` и `sscanf` имеют одну неприятную особеность при считывании строк. Спецификатор `%s` делает следующее: считывает строку и записывает её всю по передаваемому ей адресу. Это может привести к серьёзным ошибкам если то место, куда мы записываем строку будет меньше, чем записываемая строка. Рассмотрим, например, следующую простую программу:

```
#include <stdio.h>
int main()
{
        char a[5] = "Lion";
        char b[5];

        scanf("%s", b);
        printf("a = %s\n", a);
        printf("b = %s\n", b);
}
```

- Что напечатает эта программа, если на вход передать строку `Cat`?
- Что напечатает эта программа, если на вход передать строку `Zebra`?
- Что напечатает эта программа, если на вход передать строку `Elephant`?
- Как исправить ошибки?



Функция `scanf` выходит за границы массива `b` и переписывает другую строку. Если бы на месте строки `a` была бы переменная другого типа, то `scanf` испортил бы и её. Более того, таким поведением `scanf` могут воспользоваться злоумышленники и взломать вашу программу.

Функция `fgets`.

Таким образом функция `scanf` со спецификатором `%s` является небезопасной. Чтобы недопустить подобных ошибок лучше использовать функцию `fgets`:

`fgets(<адрес куда записываем>, <вместимость строки куда записываем>, <файловый указатель или stdin>)`

Функция `fgets` считывает строку до символа `\n` либо пока не закончится место. В конце файла функция `fgets` возвращает `NULL`. Таким образом пример выше перепишется в виде:

```
#include <stdio.h>
int main()
{
        char a[5] = "Lion";
        char b[5];

        fgets(b, 5, stdin);
        printf("a = %s\n", a);
        printf("b = %s\n", b);
}
```

- Что напечатает эта программа, если подать на вход строки `Cat`, `Zebra` или `Elephant`?
- Напишите программу, которая печатает всё содержимое файла на экран с помощью функции `fgets` в предположение, что строка файла не больше 200 символов. Выведите на экран содержимое файла `sail.txt`.

Функция `fgetc`.

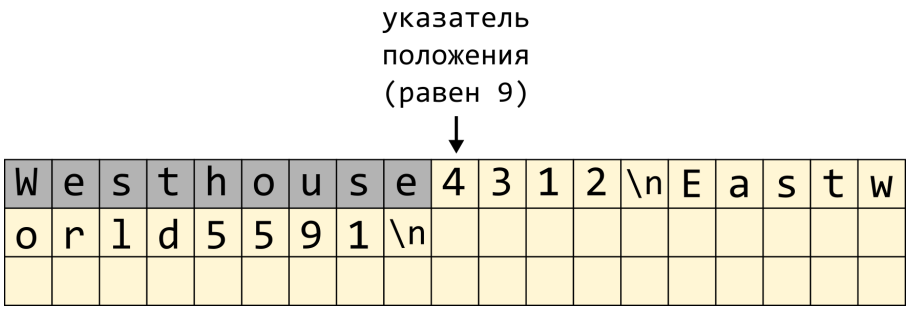
Функция `fgetc` считывает 1 символ и возвращает код `ASCII` символа или `EOF` если дошли до конца файла (`EOF` это просто константа равная -1). Пример считывания:

```
#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.