Семинар #10: Аргументы командной строки. Файлы.

Вспомогательный материал: функции `sprintf` и `sscanf`.

На предыдущих занятиях мы прошли функции ввода/вывода из стандартных потоков: `printf` и `scanf`. А также функции для ввода/вывода из файлов: `fprintf` и `fscanf`. Функции `sprintf` и `sscanf` предназначены для ввода/вывода в строку или из строки. Пример использования можно посмотреть в файле `0sscanf.c`.

- **Задача: Конвертация переменных в строки и обратно**

```
char str1[10] = "79";
char str2[10] = "435";
```

  Конвертирутуйте эти 2 строки в числа, сложите и напечатайте результат

Часть 1: Аргументы командной строки

Программы могут принимать аргументы. Простейший пример – утилита `ls`. Если запустить `ls` без аргументов, то она просто напечатает содержимое текущей директории. Если же использовать эту программу с опцией `-l`: `ls -l`, то на экран выведется подробное описание файлов и папок в текущей директории. Другой пример - опция для компилятора `gcc -std=c99`.

Задачи

- `xxd` - это простая программа, которая выводит на экран всё содержимое файла побайтово. Если, например, запустить программму следующим образом: `xxd a.out`, то она выведет на экран всё содержимое этого исполняемого файла. Часто используемые опции командной строки: `-h` (сокращение от help) и `-v` (сокращение от version).
  - Запустите `xxd` с аргументом – именем файла `hello.txt`. Этот файл содержит лишь строку `Hello`. `xxd` покажет вам содержимое этого файла в шестнадцатеричном виде и в виде `ASCII`.
  - Запустите `xxd` с опцией `-h`.
  - Запустите `xxd` с нужной опцией, чтобы вывод файла `hello.txt` был представлен в двоичном виде.
  - * Если файл большой, то весь вывод `xxd` не поместится на экран. Перенаправить вывод в нужный файл можно следующим образом: `xxd a.out > temp.txt`. После этого в файле `temp.txt` будет хранится всё, что было бы напечатано на экран.
  - * Создайте программу Hello World и скомпилируйте её в файл `a.out`. Сохраните вывод `xxd ./a.out` в отдельном файле `hw.txt`. Измените файл `hw.txt`, так чтобы программа печатала Hello MIPT. Создайте исполняемый файл из файла `hw.txt`, используя `xxd` с опцией `-r`.
- **argc:** Простейшая программа `1argc.c` печатает количество аргументов командной строки. Скомпилируйте эту программу и протестируйте её, запуская с разным количеством аргументов.
- **argv:** Простейшая программа `2argv.c` печатает аргументы командной строки. Скомпилируйте эту программу и протестируйте её, запуская с разным количеством аргументов.

```
#include <stdio.h>
int main(int argc, char** argv) {
    printf("Number of arguments = %d\n", argc);
    for (int i = 0; i < argc; ++i)
        printf("Argument #%d = %s\n", i, argv[i]);
}
```

- **Сумма аргументов:** Создайте программу `sum`, которая будет печатать сумму всех аргументов. Например, при вызове
  `./sum 4 8 15 16 23 42`
  программа должна напечатать `108`

Часть 2: Работы с текстовыми файлами

- `fopen`: Открывает файл для чтения/записи
  Режимы открытия файла:

`r`	открыть существующий файл для чтения (read)
`w`	создать новый файл и открыть его для записи (write) если файл уже существует, то он удалится перед записью
`a`	открыть для записи в конец файла (append)
`r+`	открыть для чтения/записи, с начала файла
`w+`	создать новый файл и открыть его для чтения/записи
`a+`	открыть для чтения/записи в конец файла

  Для бинарных файлов в Windows нужно добавить символ `b`.

- `fclose`: Закрывает файл

- `fprintf/fscanf`: Функции работают аналогично `printf/scanf`, но только работают с файлом. Файл(указатель на специальную структуру `FILE`) нужно передать первым аргументом.

Пример программы, которая создаёт файл и записывает в него `Hello world!`.

```c
#include <stdio.h>
#include <stdlib.h>
int main() {
    FILE* fp = fopen("myfile.txt", "w");
    if (fp == NULL) {
        printf("Error!\n");
        exit(1);
    }
    fprintf(fp, "Hello world!");
    fclose(fp);
}
```

В этой программе мы делаем следующее:

– Создаём и открываем файл `"myfile.txt"` на запись (так как режим открытия `w`).

– Проверяем получилось ли открыть файл. Если не получилось, то пишем сообщение об ошибке и выходим. В дальнейших примерах эта проверка будет опускаться для экономии места.

– Если получилось открыть, то записываем в файл строку с помощью `fprintf`.

– Закрываем файл.

Задачи

- Скомпилируйте программу `3fprintf.c` и запустите. В результате выполнения программы должен появиться файл `myfile.txt` с содержимым `Hello world!`.

- Напишите программу, которая будет создавать файл `numbers.txt` и записывать туда все числа от 0 до 1000, делящиеся на 7.

- В файле `input.txt` лежат числа (сначала идёт количество чисел, а потом сами числа). Вам нужно считать эти числа и вывести их сумму на экран.

- Измените программу из предыдущей задачи так, чтобы она записывала результат не на экран, а в файл `output.txt`.

Часть 3: 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 = 287454020;  // Это число = 0x11223344 в шестнадцатеричной системе
    FILE* fb = fopen("binary.dat", "w");
    fwrite(&a, sizeof(int), 1, fb);
    fclose(fb);
}
```

- **Печать в текстовом и бинарном виде:**
  В файле `4text_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 4: Посимвольное чтение из файла

`fgetc` - посимвольное чтение из файла - возвращает ASCII код следующего символа из файла. Если символов не осталось, то она возвращает константу `EOF` равную `-1`.
Пример программы, которая находит количество цифр в файле:

```
#include <stdio.h>
int main() {
    FILE* f = fopen("input.txt", "r");
    int c;
    int num_of_digits = 0;

    while ((c = fgetc(f)) != EOF) {
        if (c >= '0' && c <= '9')
            num_of_digits += 1;
    }
    printf("Number of digits = %d\n", num_of_digits);
    fclose(f);
}
```

Эта программа содержится в файле `5number_of_digits.c`

Задачи

- Написать программу `symbolcount`, которая считает количество символов в файле. название файла должно передаваться через аргумент командной строки:

```
gcc -o symbolcount main.c
./symbolcount war_and_peace.txt
3332371
```

- Написать программу `linecount`, которая находит количество строк в файле.
- Написать программу `wordcount`, которая находит количество слов в файле. Слово это любая последовательность символов, разделённая *одним или несколькими* пробельными, символами. Пробельные символы это пробел, перенос на новую строку(`\n`) либо табуляция(`\t`).

Функции `ftell` и `fseek`

`ftell`: Функция, которая возвращает текущее положение в файле. Например, если мы начали считать с начала файла и считали 10 символов, то эта функция вернёт 10.
`fseek`: Функция, которая устанавливает положение в файле. Например, следующая строка устанавливает положение на 100-й символ:

```
fseek (fout, 100, SEEK_SET);
```

Следующие функции будут считывать всё начиная с 101-го символа.

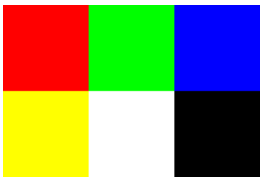
Часть 5: Работа с изображениями формата `.ppm`

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.