Аргументы командной строки. Файлы.

Функции sprintf и sscanf.

```
#include <stdio.h>
int main(){
    char str[20] = "2010-5-2";
    // Считываем из строки:
    int year, month, day;
    sscanf(str, "%d-%d-%d", &year, &month, &day);
}
```

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

  ```
  char str1[10] = "435";
  char str2[10] = "73";
  ```

  Конвертирутуйте эти 2 строки в числа, сложите и напечатайте результат

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`.

- `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 опцией `-h`.
  - Запустите `xxd` с нужной опцией, чтобы вывод был представлен в двоичном виде.
  - Перенаправить вывод в нужный файл можно следующим образом: `xxd a.out > temp.txt`
  - * 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:** Следующая простейшая программа печатает количество аргументов командной строки. Скомпилируйте эту программу и протестируйте её, запуская с разным количеством аргументов.

  ```
  int main(int argc, char** argv)
  {
          // argc - количество аргументов командной строки
          printf("%d\n", argc);
  }
  ```

- **argv:** Следующая программа печатает аргументы командной строки. Скомпилируйте эту программу и протестируйте её, запуская с разным количеством аргументов.

  ```
  int main(int argc, char** argv)
  {
          // argv - массив массивов символов ( или массив строк ) - сами аргументы
          for (int i = 0; i < argc; i++)
                  printf("%d: %s\n", i, argv[i]);
  }
  ```

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

Функции для работы с файлами

- `fprintf/fscanf:` Создаём и открываем файл "myfile.txt"на запись ("w"означает "write" , "r"означает "read"). Проверяем получилось ли открыть файл, если нет, то пишем сообщение об ошибке и выходим. В дальнейших примерах эта проверка будет опускаться. Если получилось открыть, то записываем в файл строку с помощью fprintf().

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

Режимы открытия файла:

r	открыть существующий файл для чтения
w	создать новый файл и открыть его для записи
a	открыть для записи в конец файла
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 нужно добавить символ `b`.

- `fread/fwrite` **- бинарное чтение/запись:** Записываем содержимое массива arr в файл `numbers.dat`.

```c
#include <stdio.h>
int main(){
    FILE* fp = fopen("numbers.dat", "wb");
    int arr[5] = {55, 66, 77, 88, 99};
    fwrite(arr, sizeof(int), 5, fp1);
    fclose(fp1);
}
```

  - Пусть есть массив из чисел типа `unsigned char`:
    `unsigned char array[] = {97, 255, 4, 145};`
    Запишите эти числа в текстовый файл `text.txt`, используя `fprintf`. Изучите содержимое этого файла побайтово с помощью `xxd`.
  - Запишите эти числа в бинарный файл `data.dat`, используя `fwrite`. Изучите содержимое этого файла побайтово с помощью `xxd`.
  - **Little or big endian:** Число `int a = 6242983;` в шестнадцатеричной системе счисления имеет вид `005f42a7`. Запишите это число в файл в бинарном виде, используя функцию fwrite(). Проверьте, что записалось в файл. Чтобы посмотреть файл в шестнадцатеричном виде можно использовать утилиту xxd.Определить какой порядок байт используется в вашей системе: прямой(big endian) или обратный(little endian).

- `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:

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

  - 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
```

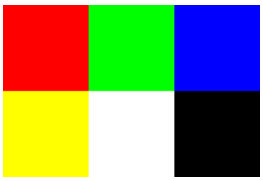
  - Написать программу `linecount`, которая находит количество строк в файле.
  - Написать программу `wordcount`, которая находит количество слов в файле. Слово это любая последовательность символов, разделённая одним или несколькими пробельными, символами. Пробельные символы это пробел, перенос на новую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

- **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:



– 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.