

Домашняя работа №4
13.10.2018

1. Реализовать аналог команды **cat** — печатает в стандартный поток вывода содержимое указанного файла. Имя файла передается в качестве аргумента командной строки (гарантируется корректный запуск). Нельзя использовать заголовочные файлы `<stdio.h>` и `<stdlib.h>` и вызовы серии **exec**.

Содержимое input.txt	Bimbo's flight!
Запуск в терминале	./01 input.txt
Вывод в терминале	Bimbo's flight!

2. Реализовать аналог команды **wc** — печатает количество строк (то есть переносов на новую строку), количество слов (слово — непрерывная последовательность символов отличных от пробела, табуляции и переноса строки) и общее количество символов (количество байт) указанного файла. Имя файла передается в качестве аргумента командной строки. Нельзя использовать заголовочные файлы `<stdio.h>` и `<stdlib.h>` и вызовы серии **exec**. (вывод чисел реализовать самостоятельно).

Содержимое input.txt	we all live in a yellow submarine
Запуск	./02 input.txt
Вывод	2 7 33

3. Реализовать аналог команды **cp** с фильтрацией — копирует файл, оставляя в нем только буквы. Имена файлов передаются в качестве аргумента командной строки (первое имя — откуда, второе имя — куда копировать).

Содержимое input.txt	Bimbo's flight!
Запуск	./03 input.txt output.txt
Содержимое output.txt	Bimbos flight

4. В качестве аргумента командной строки передается одно целое число N — количество процессов, которые должен породить родительский процесс (тип порождения `dandelion` — одуванчик). Каждый дочерний процесс должен напечатать свой `pid` и родительский `pid`.

Для перевода строкового представления числа в реальное число используйте функцию:

```
int atoi(const char *nptr);
```

Запуск	./prog 4
Вывод	16702 16701 16703 16701 16704 16701 16705 16701

5. В качестве аргумента командной строки передается одно целое число N — количество процессов, которые должен породить друг друга по цепочке (тип порождения `bamboo` — бамбук). Каждый дочерний процесс должен напечатать свой `pid` и родительский `pid`.

Для перевода строкового представления числа в реальное число используйте функцию:

```
int atoi(const char *nptr);
```

Запуск	./prog 4
Вывод	16702 16701 16703 16702 16704 16703 16705 16704

6. (Требуется какая-либо программа из первых четырех). Рядом с вашей программой лежит код от одной из предыдущих программ (они называются `01.c` `02.c` `03.c` `04.c`). В качестве аргумента командной строки подается имя программы (вместо с суффиксом `.c`), которую надо скомпилировать. Вам необходимо написать программу, которая компилирует указанный код компилятором `gcc` со стандартным для практикума набором флагов. Например программу `04.c` надо компилировать так

```
gcc 04.c -o 04 -Wall -Werror
```

Запуск	./05 04.c
Итог	скомпилированный код 04 в текущем каталоге

7. Программа в бесконечном цикле читает строки пока ей не попадут на ввод `exit` или `quit`. Гарантируется, что все строки не длиннее 10 символов и не содержать никаких символов кроме латинских букв (в том числе отсутствуют и пробелы). После каждой незавершающей строки необходимо породить дочерний процесс (`fork`) и произвести запуск соответствующей команды в дочернем процессе (`exec`, без дополнительных аргументов).

Для ввода и сравнения строк используйте функцию:

```
char *fgets(char *s, int size, FILE *stream);  
int strcmp(const char *s1, const char *s2);
```

Ввод	ls date pwd exit
Вывод	Documents Downloads Music St oct 14 12:00:00 +06 2018 /home/valera

Следующие 3 задания требуют правильно подготовленной картинки в формате bmp. Это несложно сделать в редакторе gimp, который установить можно командой `sudo apt install gimp`:

- открываем любое изображение в gimp;
- выбираем в меню: изображение — размер холста;
- выравниваем и ширину, и высоту на **числа кратные 4**;
- выбираем в меню: файл — экспортировать как — изображение windows bmp;
- параметры совместимости: **не сохранять данные о цветовом формате**;
- дополнительные параметры: **24 разряда (R8 G8 B8)**;
- проверьте в терминале размер файла командой `ls -l`, он должен быть равен $(3 \cdot w \cdot h + 54)$ байт.

8 Определить размер bmp-изображения, имя которого передано в качестве аргумента командной строки. Алгоритм:

- 1) пропускаем первые 18 байт из файла (начало заголовка bmp-файла).
- 2) считываем 4 байта, соответствующие ширине, и 4 байта, соответствующие высоте.

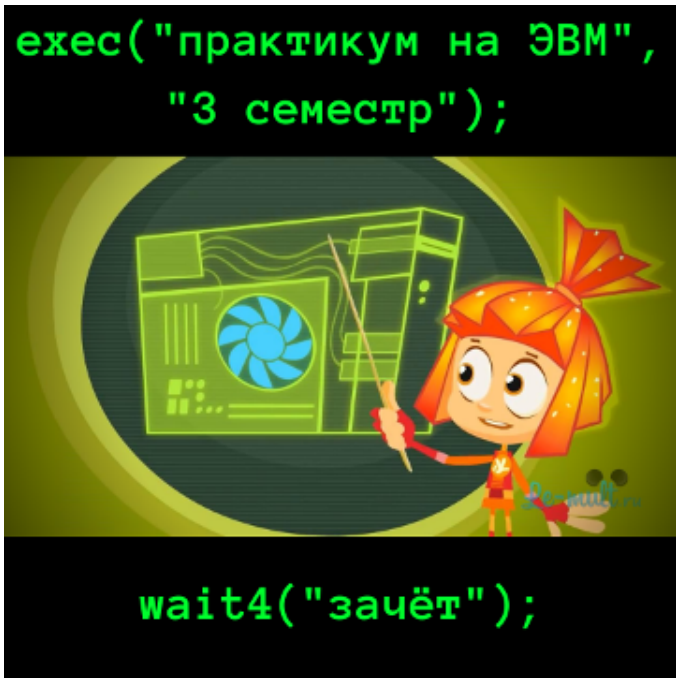
Для хранения заголовка файла изображения создать структуру:

```
struct Image {
    char preheader[18];
    int w, h;
};
```

Считывать каждое поле структуры отдельным вызовом `read()`. Процесс считывания данных изображения оформить в виде функции:

```
struct Image openImage(char *filename) {
    struct Image img;
    int fd = open(filename, O_RDONLY);
    read(fd, &img.preheader, sizeof(img.preheader));
    read(fd, &img.w, sizeof(img.w));
    read(fd, &img.h, sizeof(img.h));
    close(fd);
    return img;
}
```

Вывод ответа произвести в функции `main()`.

Изображение avatar.bmp	
Запуск	./08 avatar.bmp
Вывод	360 360

9 Создать файл с зеркальным отражением исходного bmp-изображения по вертикали. В качестве аргументов командной строки передается: имя исходного файла и имя выходного файла. Алгоритм:

- 1) пропускаем первые 18 байт из файла (начало заголовка bmp-файла).
- 2) считываем 4 байта, соответствующие ширине, и 4 байта, соответствующие высоте.
- 3) пропускаем еще 28 байт из файла (конец заголовка bmp-файла).
- 4) считываем $w \cdot h$ пачек по 12 байт в виде структур:

```
struct Pixel {  
    unsigned char r, g, b;  
};
```

Данные пачки описывают пиксели изображения, уложенные по строкам слева направо, а строки идут **снизу вверх**. Для хранения пикселей добавить в структуру `Img` поле типа «динамическая матрица»:

```
struct Image {  
    char preheader[18];  
    int w, h;  
    char postheader[28];  
    struct Pixel **pixmap;  
};
```

Процесс ввода и вывод в виде функций:

```
struct Image openImage(char *filename);  
void saveImage(struct Image img, char *filename);
```

Изображение avatar.bmp	
Запуск	./09 avatar.bmp avatar-sym.bmp
Изображение avatar-sym.bmp	

- 10 Создать файл отфильтрованным исходным bmp-изображением. В качестве аргументов командной строки передается: имя исходного файла, имя выходного файла и уровень фильтрации. Фильтр заключается в выделении границ по заданному уровню `level` у изображения, сохраненного в формате `*.bmp` без палитры. Каждый пиксель P_{ij} описывается тремя каналам цвета: красный r_{ij} , зеленый g_{ij} и синий b_{ij} (от 0 до 255 каждый). Их необходимо заменить на черный (0, 0, 0) или белый (255, 255, 255) цвет согласно нижеприведенным правилам. Алгоритм (первые 4 шага как предыдущей задаче):

5) строим матрицу интенсивностей `int **D` как сумму каналов для всех i и j :

$$D_{i,j} = r_{i,j} + g_{i,j} + b_{i,j}.$$

6) если дискретный градиент G в пикселе i и j

$$G_{i,j} = \max\{|D_{i,j} - D_{i,j+1}|, |D_{i,j} - D_{i,j-1}|, |D_{i,j} - D_{i-1,j}|, |D_{i,j} - D_{i+1,j}|\}$$

больше заданного уровня `level`, то красим данный пиксель в черный цвет (все три канала равны 0), иначе красим в белый цвет (все три канала равны 255). Будьте аккуратны на граничных пикселях, чтобы не получить `segfault`!)

7) первые 54 байта исходного файла копируем в итоговый файл без изменений.

8) пишем $w \cdot h$ пачек по 12 байт в виде структур. Данные пачки описывают пиксели изображения, уложенные по строкам слева направо, а строки идут **снизу вверх**.

Изображение avatar.bmp

```
ехес("практикум на ЭВМ",  
"3 семестр");
```



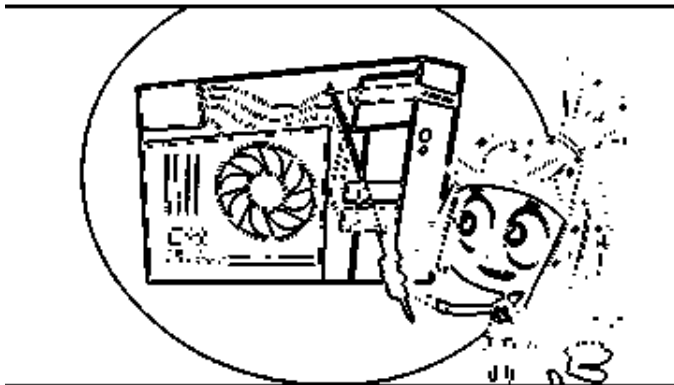
```
wait4("зачёт");
```

Запуск

```
./10 avatar.bmp avatar-border.bmp 100
```

Изображение avatar-sym.bmp

```
ехес("практикум на ЭВМ",  
"3 семестр");
```



```
wait4("зачёт");
```