

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

КУРСОВАЯ РАБОТА
по дисциплине «Программирование»
Тема: Обработка PNG файла

Студент гр. 3341

Романов А.К.

Преподаватель

Глазунов С.А.

Санкт-Петербург

2024

ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

Студент Романов А. К.

Группа 3341

Вариант 23

Программа обязательно должна иметь CLI (опционально дополнительное использование GUI). Более подробно тут:

http://se.moevm.info/doku.php/courses:programming:rules_extra_kurs

Программа должна реализовывать весь следующий функционал по обработке png-файла

Общие сведения

Формат картинки PNG (рекомендуем использовать библиотеку libpng) без сжатия

файл может не соответствовать формату PNG, т.е. необходимо проверка на PNG формат. Если файл не соответствует формату PNG, то программа должна завершиться с соответствующей ошибкой.

обратите внимание на выравнивание; мусорные данные, если их необходимо дописать в файл для выравнивания, должны быть нулями.

все поля стандартных PNG заголовков в выходном файле должны иметь те же значения что и во входном (разумеется кроме тех, которые должны быть изменены).

Программа должна иметь следующую функции по обработке изображений:

(1) Рисование окружности. Флаг для выполнения данной операции: `--circle`.

Окружность определяется:

координатами ее центра и радиусом. Флаги `--center` и `--radius`. Значение флаг `--center` задаётся в формате `x.y`, где x – координата по оси x, y –

координата по оси y. Флаг `--radius` На вход принимает число больше 0
толщиной линии окружности. Флаг `--thickness`. На вход принимает число
больше 0
цветом линии окружности. Флаг `--color` (цвет задаётся строкой `rrr.ggg.bbb`,
где `rrr/ggg/bbb` – числа, задающие цветовую компоненту. пример `--color
255.0.0` задаёт красный цвет)
окружность может быть залитой или нет. Флаг `--fill`. Работает как бинарное
значение: флага нет – `false` , флаг есть – `true`.
цветом которым залита сама окружность, если пользователем выбрана
залитая окружность. Флаг `--fill_color` (работает аналогично флагу `--color`)

(2) Фильтр rgb-компонент. Флаг для выполнения данной операции: `--
rgbfilter`. Этот инструмент должен позволять для всего изображения либо
установить в диапазоне от 0 до 255 значение заданной компоненты.

Функционал определяется

Какую компоненту требуется изменить. Флаг `--component_name`.

Возможные значения `'red'`, `'green'` и `'blue'`.

В какой значение ее требуется изменить. Флаг `--component_value`.

Принимает значение в виде числа от 0 до 255

(3) Разделяет изображение на N*M частей. Флаг для выполнения данной
операции: `--split`. Реализация: провести линии заданной толщины.

Функционал определяется:

Количество частей по “оси” Y. Флаг `--number_x`. На вход принимает число
больше 1

Количество частей по “оси” X. Флаг `--number_y`. На вход принимает число
больше 1

Толщина линии. Флаг `--thickness`. На вход принимает число больше 0

Цвет линии. Флаг `--color` (цвет задаётся строкой `rrr.ggg.bbb`, где `rrr/ggg/bbb`

– числа, задающие цветовую компоненту. пример `--color 255.0.0` задаёт красный цвет)

Все подзадачи, ввод/вывод должны быть реализованы в виде отдельной функции.

Содержание пояснительной записки:

разделы «Аннотация», «Содержание», «Введение», «Ход работы», «Пример работы программы», «Заключение», «Список использованных источников»

Предполагаемый объем пояснительной записки:

Не менее 15 страниц.

Дата выдачи задания: 18.03.2024

Дата сдачи реферата: 20.05.2024

Дата защиты реферата: 22.05.2024

Студент		Романов А. К.
Преподаватель		Глазунов С.А.

АННОТАЦИЯ

В данной курсовой работе была реализована программа, обрабатывающая PNG изображения, не имеющие сжатия. Программа проверяет тип изображения, его версию, при соответствии требованиям в дальнейшем обрабатывает его и подаёт на выход изменённую копию изображения. Взаимодействие с программой осуществляется с помощью CLI (интерфейс командной строки).

SUMMARY

In this course has been created a program that processes uncompressed PNG images. The program checks the type of image, its version, if it meets the requirements, it further processes it and outputs a modified copy of the image. Interaction with the program is performed using CLI (command line interface).

СОДЕРЖАНИЕ

	Введение	7
1.	Работа с файлами	8
2.	Ввод аргументов и проверка их корректности	11
3.	Обработка изображения	13
3.1	Фильтр RGB компонент	13
3.2	Рисование окружности	13
3.3	Разделение изображения на части	15
4.	Заключение	8
5.	Приложение А. Исходный код программы	8
6.	Приложение В. Демонстрация работы программы	8

ВВЕДЕНИЕ

Целью данной работы является создание программы на языке Си для обработки PNG изображений.

Для достижения поставленной цели потребовалось решить ряд задач:

- изучить, как устроены PNG файлы, что они в себе содержат;
- научиться распознавать PNG файлы среди прочих и проверять их характеристики;
- научиться считывать и записывать PNG изображения;
- разработать функцию рисования круга на изображении, его заливки;
- разработать функцию, разделяющую изображения на $m \times n$ частей линиями заданной толщины и цвета;
- разработать функцию, выставляющую требуемое значение одной из RGB компонент всем пикселям;
- изучить библиотеку *getopt.h*;
- научиться работать с аргументами командной строки, длинными и короткими флагами;
- создать *Makefile* для сборки программы;
- протестировать разработанную программу.

1. РАБОТА С ФАЙЛАМИ

Работа с файлом происходит при помощи библиотеки *png.h*. Функции по считыванию файла, проверки его и заполнения соответствующей структуры, а также функция по созданию PNG файла и записыванию в него полученную структуру описаны в мануале *pnglib*.

2. ВВОД АРГУМЕНТОВ

Ввод аргументов в программу происходит по средству флагов. Считать их можно благодаря библиотеке `getopt.h`, в которой описывается функция `getopt_long()`. Получаемые значения проходят сквозь ветку `switch-case`, записывая необходимые аргументы в соответствующие переменные, а затем через еще одну ветку `switch-case`, в которой вызываются необходимые функции обработки изображения (`draw_circle`, `split`, `filter_RGB`).

Перед каждым вызовом такой функции осуществляется проверка аргументов (функции `valid_circle`, `valid_split`, `valid_filter` соответственно). Если какой-либо из аргументов принимает недопустимое значение, работа программы прекращается.

Также осуществляется проверка корректности введенного цвета после парсинга строки с аргументом. Если значение какой-либо из RGB компонент меньше 0 или больше 255, программа завершает работу.

3. ОБРАБОТКА ИЗОБРАЖЕНИЙ

3.1 Фильтр RGB компонент

Осуществляется при помощи функции *filter_RGB()*. На вход функции подаётся имя компоненты, которую требуется изменить, значение, в которое ее нужно поставить. Для изменения изображения осуществляется перебор всех пикселей картинки, и для каждого пикселя нужной компоненте присваивается указанное значение.

3.2 Рисование окружности

Рисование круга осуществляется с помощью функции *draw_circle()*, которая принимает на вход координаты центра окружности (x и y), радиус окружности, толщину линии окружности, ее цвет, флаг fill (нужна ли заливка или нет), а также цвет заливки. В функции также осуществляется перебор всех пикселей. Если координаты пикселя удовлетворяют уравнению окружности с соответствующим центром и радиусом (с учётом толщины линии), то пиксели перекрашивается. Если необходимо сделать заливку, то координаты пикселя должны удовлетворять неравенству окружности.

3.3 Разделение изображения на части

Осуществляется с помощью функции *split()*. Если произведение количества линий на толщину линии больше размерности изображения, то все пиксели изображения перекрашиваются в нужный цвет. Иначе, рассчитывается позиция края верхней линии и количество пикселей между краями двух соседних линий (шаг). После этого перебираются горизонтальные ряды пикселей картинки с вычисленным шагом и рисуются линии нужной толщины. Аналогично для вертикальных линий.

ЗАКЛЮЧЕНИЕ

Разработана программа на языке программирования Си, обрабатывающая PNG изображения и имеющая CLI. В ходе выполнения работы было изучено устройство PNG файлов; изучены методы считывание и записи файлов; получены навыки обработки изображений; разработаны функции для рисования окружности и ее заливки; разделения изображения на нужное количество частей; изменения RGB компонент; изучены библиотеки *libpng* и *getopt.h*; изучена работа с аргументами командной строки.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. М. М. Заславский, А. А. Лисс, А. В. Гаврилов, С. А. Глазунов, Я. С. Государкин, С. А. Тиняков, В. П. Голубева, К. В. Чайка, В. Е. Допира. Б17 Базовые сведения к выполнению курсовой работы по дисциплине «Программирование». Второй семестр, 2024.
2. <https://firststeps.ru/linux/r.php?11> — работа с getopt

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.c

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <png.h>
#include <math.h>
#include <getopt.h>

const char* short_opts = "ho:i:";

const struct option long_options[] = {
    {"help", no_argument, NULL, 'h'},
    {"output", required_argument, NULL, 'o'},
    {"input", required_argument, NULL, 'i'},
    {"rgbfilter", no_argument, NULL, 2001},
    {"component_name", required_argument, NULL, 2002},
    {"component_value", required_argument, NULL, 2003},
    {"circle", no_argument, NULL, 3001},
    {"center", required_argument, NULL, 3002},
    {"radius", required_argument, NULL, 3003},
    {"thickness", required_argument, NULL, 3004},
    {"fill", no_argument, NULL, 3005},
    {"color", required_argument, NULL, 5001},
    {"fill_color", required_argument, NULL, 5002},
    {"split", no_argument, NULL, 4001},
    {"number_x", required_argument, NULL, 4002},
    {"number_y", required_argument, NULL, 4003},
    {NULL, 0, NULL, 0}
};

struct Png
{
    int                pixel_size;
    png_structp        png_ptr;
    png_infop          info_ptr;
    png_infop          end_info_ptr;
    int                width;
    int                height;
    png_byte           bit_depth;
    png_byte           color_type;
    png_byte           interlace_type;
    png_byte           compression_type;
    png_byte           filter_type;
    int                number_of_passes;
    png_bytepp         row_pointers;
};

struct RGB_color
{
    png_byte red;
    png_byte green;
    png_byte blue;
    png_byte alpha;
};
```

```

void read_png(const char* path, struct Png *image)
{
    FILE *file_pic = fopen(path, "rb");

    if (!file_pic)
    {
        exit(42);
    }

    png_byte signature[8];
    fread(signature, sizeof(png_byte), 8, file_pic);
    if(png_sig_cmp(signature, 0, 8) != 0){
        exit(42);
    }

    image->png_ptr = png_create_read_struct(PNG_LIBPNG_VER_STRING, NULL, NULL,
    NULL);
    if (!image->png_ptr)
    {
        fclose(file_pic);
        exit(42);
    }

    image->info_ptr = png_create_info_struct(image->png_ptr);
    if (!image->info_ptr)
    {
        png_destroy_read_struct(&image->png_ptr, NULL, NULL);
        fclose(file_pic);
        exit(42);
    }

    image->end_info_ptr = png_create_info_struct(image->png_ptr);
    if (!image->end_info_ptr)
    {
        png_destroy_read_struct(&image->png_ptr, NULL, NULL);
        fclose(file_pic);
        exit(42);
    }
    if (setjmp(png_jmpbuf(image->png_ptr)))
    {
        png_destroy_read_struct(&image->png_ptr, &image->info_ptr, &image-
>end_info_ptr);
        fclose(file_pic);
        exit(42);
    }

    png_init_io(image->png_ptr, file_pic);
    png_set_sig_bytes(image->png_ptr, 8);
    png_read_info(image->png_ptr, image->info_ptr);

    if(image->bit_depth == 16)
        png_set_strip_16(image ->png_ptr);

    if (image->bit_depth < 8) {
        png_set_packing(image->png_ptr);
    }

    if(image -> color_type == PNG_COLOR_TYPE_PALETTE)
        png_set_palette_to_rgb(image -> png_ptr);

    // PNG_COLOR_TYPE_GRAY_ALPHA is always 8 or 16bit depth.

```

```

if(image -> color_type == PNG_COLOR_TYPE_GRAY && image -> bit_depth < 8)
    png_set_expand_gray_1_2_4_to_8(image -> png_ptr);

if(png_get_valid(image -> png_ptr, image -> info_ptr, PNG_INFO_tRNS))
    png_set_tRNS_to_alpha(image -> png_ptr);

// These color_type don't have an alpha channel then fill it with 0xff.
if(image -> color_type == PNG_COLOR_TYPE_RGB ||
    image -> color_type == PNG_COLOR_TYPE_GRAY ||
    image -> color_type == PNG_COLOR_TYPE_PALETTE)
    png_set_filler(image -> png_ptr, 0xFF, PNG_FILLER_AFTER);

if(image -> color_type == PNG_COLOR_TYPE_GRAY ||
    image -> color_type == PNG_COLOR_TYPE_GRAY_ALPHA)
    png_set_gray_to_rgb(image -> png_ptr);

image->width = png_get_image_width(image->png_ptr, image->info_ptr);
image->height = png_get_image_height(image->png_ptr, image->info_ptr);
image->color_type = png_get_color_type(image->png_ptr, image->info_ptr);
image->bit_depth = png_get_bit_depth(image->png_ptr, image->info_ptr);
image->interlace_type = png_get_interlace_type(image->png_ptr, image-
>info_ptr);
image->filter_type = png_get_filter_type(image->png_ptr, image->info_ptr);
image->compression_type = png_get_compression_type(image->png_ptr, image-
>info_ptr);
image->number_of_passes = png_set_interlace_handling(image->png_ptr);
image->pixel_size = 3;
if(image->color_type == 4){
    image->pixel_size = 4;
}

png_read_update_info(image->png_ptr, image->info_ptr);
image->row_pointers = (png_bytep*)malloc(sizeof(png_bytep) * image->height);

for (int y = 0; y < image->height; y++)
{
    image->row_pointers[y] = (png_byte*)malloc(png_get_rowbytes(image-
>png_ptr, image->info_ptr));
}
png_read_image(image->png_ptr, image->row_pointers);
png_read_end(image->png_ptr, image->info_ptr);
png_destroy_read_struct(&image->png_ptr, &image->info_ptr, &image-
>end_info_ptr);
fclose(file_pic);

// puts("Read successful\n");
}

void write_png_file(const char* path, struct Png* image)
{
    FILE *file_pic = fopen(path, "w");
    if (!file_pic)
    {
        fclose(file_pic);
        exit(42);
    }

    image->png_ptr = png_create_write_struct(PNG_LIBPNG_VER_STRING, NULL, NULL,
NULL);
    if (!image->png_ptr)
    {
        fclose(file_pic);

```

```

        exit(43);
    }

    image->info_ptr = png_create_info_struct(image->png_ptr);
    if (!image->info_ptr)
    {
        png_destroy_write_struct(&image->png_ptr, &image->info_ptr);
        fclose(file_pic);
        exit(43);
    }

    if (setjmp(png_jmpbuf(image->png_ptr)))
    {
        png_destroy_write_struct(&image->png_ptr, &image->info_ptr);
        fclose(file_pic);
        exit(43);
    }

    png_init_io(image->png_ptr, file_pic);

    png_set_IHDR(image->png_ptr,
        image->info_ptr,
        image->width,
        image->height,
        image->bit_depth,
        image->color_type,
        image->interlace_type,
        image->compression_type,
        image->filter_type);

    png_write_info(image->png_ptr, image->info_ptr);

    if (setjmp(png_jmpbuf(image->png_ptr)))
    {
        png_destroy_write_struct(&image->png_ptr, &image->info_ptr);
        fclose(file_pic);
        exit(43);
    }

    png_write_image(image->png_ptr, image->row_pointers);
    png_write_end(image->png_ptr, NULL);
    png_destroy_write_struct(&image->png_ptr, &image->info_ptr);

    fclose(file_pic);

    // puts("Write successful\n");
}

void filter_RGB(char* component_name, int component_value, struct Png* image)
{
    int change_color = -1;

    if(strcmp(component_name, "green")==0){
        puts("one");
        change_color = 1;
    }
    if(strcmp(component_name, "blue")==0){
        puts("zero");
        change_color = 2;
    }
    if(strcmp(component_name, "red")==0){

```



```

        puts("two");
        change_color = 0;
    }

    for(int y = 0; y < image->height; y++){
        png_byte *row = image->row_pointers[y];
        for(int x = 0; x < image->width; x++){
            png_byte *pix = &(row[x * image->pixel_size]);
            pix[change_color] = component_value;
        }
    }
}

void split(int number_x, int number_y, int thickness, struct RGB_color *color,
struct Png* image)
{
    // in case of user's stupidity
    if(number_x * thickness > image->height || number_y * thickness > image->
width){
        // puts("wow, you are really dumb");
        for(int y = 0; y < image->height; y++){
            png_byte *row = image->row_pointers[y];
            for(int x = 0; x < image->width; x++){
                row[x * image->pixel_size] = color->red;
                row[x * image->pixel_size + 1] = color->green;
                row[x * image->pixel_size + 2] = color->blue;
            }
        }
        return;
    }

    int up = thickness / 2;
    int step_y = image->height / number_x;
    int start_hor = step_y - up;

    //in case if thickness too big
    if(start_hor < 0){
        start_hor = 0;
    }

    for(int y = start_hor; y < image->height; y+=step_y){
        for(int add_thicc = 0; add_thicc < thickness; add_thicc++){
            if(y+add_thicc >= image->height){ //in case if thickness too big
                continue;
            }
            png_byte *row = image->row_pointers[y+add_thicc];
            for(int x = 0; x < image->width; x++){
                png_byte *pix = &(row[x * image->pixel_size]);
                pix[0] = color->red;
                pix[1] = color->green;
                pix[2] = color->blue;
            }
        }
    }

    int left = thickness / 2;
    int step_x = image->width / number_y;
    int start_ver = step_x - left;

```

```

    if(start_ver < 0){          // if too thicc (ಁಁ)
        start_ver = 0;
    }

    for(int y = 0; y < image->height; y++){
        png_byte *row = image->row_pointers[y];
        for(int x = start_ver; x < image->width ; x+=step_x){
            for(int add_thicc = 0; add_thicc < thickness; add_thicc++){
                if(x+add_thicc >= image->width){
                    continue;          //if too thicc
                }
                png_byte *pix = &(row[(x+add_thicc) * 3]);
                pix[0] = color->red;
                pix[1] = color->green;
                pix[2] = color->blue;
            }
        }
    }
}

void draw_circle(int x_center, int y_center, int radius, int thickness, int
fill,
struct RGB_color *line_color, struct RGB_color *fill_color, struct Png* image)
{
    // if(x_center < 0 || x_center > image->width || y_center < 0 || y_center >
image->height){
        //      exit(0);
        // }

    for(int y = 0; y < image->height; y++){
        png_byte *row = image->row_pointers[y];
        for(int x = 0; x < image->width;  x++){
            int fit_in = floor(sqrt(pow(x - x_center, 2)+pow(y - y_center, 2)));

            if(fit_in >= (radius - thickness/2) && fit_in <= (radius +
thickness/2)){
                row[x * image->pixel_size] = line_color->red;
                row[x * image->pixel_size + 1] = line_color->green;
                row[x * image->pixel_size + 2] = line_color->blue;
            }
        }
    }

    if (fill){
        for(int y = 0; y < image->height; y++){
            png_byte *row = image->row_pointers[y];
            for(int x = 0; x < image->width;  x++){
                int fit_in = floor(sqrt(pow(x - x_center, 2)+pow(y - y_center,
2)));

                if(fit_in < (radius - thickness/2)){
                    row[x * image->pixel_size] = fill_color->red;
                    row[x * image->pixel_size + 1] = fill_color->green;
                    row[x * image->pixel_size + 2] = fill_color->blue;
                }
            }
        }
    }
}

void help()

```

```

{
    puts("\nWELCOME TO MY CW!\n\nAvailable options:\n");
    puts("1. RGB_FILTER: sets one RGB component to stated value");
    puts("REQUIRED_ARGUMENTS: --component_name (red, green or blue), --
component_value (from 0 to 255)\n");
    puts("2. SPLIT: splits image to m*n parts");
    puts("REQUIRED_ARGUMENTS: --number_x (> 1), --number_y(> 1), --color, --
thickness (> 0)\n");
    puts("3. DRAW_CIRCLE: draws circle over a picture");
    puts("REQUIRED_ARGUMENTS: --center, --radius (> 0), --color, --thickness (>
0), --fill (true/false), --fill_color\n");
    puts("NOTICE:\n\t coordinates should be enter in this format: xx.yy");
    puts("\t colors should be enter in this format: rrr.ggg.bbb (all components
should be > 0 and < 255)");
}

struct RGB_color* parse_color(char* color_str)
{
    struct RGB_color *parsed_color = (struct RGB_color*)malloc(sizeof(struct
RGB_color));
    char *tmp = strtok(color_str, ".");
    if(tmp==NULL){
        exit(41);
    }
    parsed_color->red = atoi(tmp);
    tmp = strtok(NULL, ".");
    if(tmp==NULL){
        exit(41);
    }
    parsed_color->green = atoi(tmp);
    tmp = strtok(NULL, ".");
    if(tmp==NULL){
        exit(41);
    }
    parsed_color->blue = atoi(tmp);
    parsed_color->alpha = 255;
    if(parsed_color->red > 255 || parsed_color->green > 255 || parsed_color-
>blue > 255){
        exit(41);
    }
    return parsed_color;
}

int check_valid_io(char* path_i, char* path_o){
    return strcmp(path_i, path_o);
}

int valid_filter(char* c_name, int c_value){
    if(c_name == NULL || c_value < 0 || c_value > 255 ){
        return 0;
    }
    if (strcmp(c_name, "red")!=0 && strcmp(c_name, "green")!=0 && strcmp(c_name,
"blue")!=0){
        return 0;
    }
    return 1;
}

int valid_split(int num_x, int num_y, int thickness, struct RGB_color *color){
    if(num_x < 1 || num_y < 1 || thickness < 1){
        return 0;
    }
}

```

```

    }
    return 1;
}

int valid_circle(int radius, int thickness, int fill, struct RGB_color *color,
struct RGB_color *color_f){
    if(radius < 1 || thickness < 1 || color == NULL){
        return 0;
    }
    if(fill && color_f == NULL){
        return 0;
    }
    // else if(color_f != NULL && fill == 0){
    //     return 0;
    // }
    return 1;
}

void handler(int argc, char* argv[]){
    char path_input[100];
    char path_output[100];

    char *component_name;
    int component_value = -1;

    int center_x;
    int center_y;
    int radius = -1;
    int thickness= 0;
    int fill = 0;

    struct RGB_color *color = NULL;
    struct RGB_color *fill_color = NULL;

    int split_x = 0;
    int split_y = 0;

    struct Png image;

    int func_option = -1;

    int rez;
    int option_index;

    while ((rez = getopt_long(argc, argv, short_opts, long_options,
&option_index)) != -1){
        switch (rez)
        {
            case 'h':
                help();
                exit(0);
            case 'o':
                strncpy(path_output, optarg, strlen(optarg));
                path_output[strlen(optarg)] = '\0';
                break;
            case 'i':
                strncpy(path_input, optarg, strlen(optarg));
                path_input[strlen(optarg)] = '\0';
                break;
            case 2001:
                {

```

```

        func_option = 1;
        break;
    }
    case 2002:
    {
        component_name = (char*)malloc(strlen(optarg) * sizeof(char));
        strncpy(component_name, optarg, strlen(optarg));
        component_name[strlen(optarg)] = '\0';
        break;
    }
    case 2003:
    {
        component_value = atoi(optarg);
        break;
    }
    case 3001:
    {
        func_option = 2;
        break;
    }
    case 3002:
    {
        char center_coords[10];
        strncpy(center_coords, optarg, strlen(optarg));
        char *tmp = strtok(center_coords, ".");
        center_x = atoi(tmp);
        tmp = strtok(NULL, ".");
        center_y = atoi(tmp);
        break;
    }
    case 3003:
    {
        radius = atoi(optarg);
        break;
    }
    case 3004:
    {
        thickness = atoi(optarg);
        break;
    }
    case 3005:
    {
        fill = 1;
        break;
    }
    case 5001:
    {
        color = parse_color(optarg);
        break;
    }
    case 5002:
    {
        fill_color = parse_color(optarg);
        break;
    }
    case 4001:
    {
        func_option = 3;
        break;
    }
    case 4002:
    {

```

```

        split_x = atoi(optarg);
        break;
    }
    case 4003:
    {
        split_y = atoi(optarg);
        break;
    }

    case '?':
    {
        exit(45);
    }
}

}

if(check_valid_io(path_input, path_output) == 0){
    exit(45);
}

read_png(path_input, &image);

switch (func_option)
{
case 1:
    if(!valid_filter(component_name, component_value)){
        exit(42);
    }
    filter_RGB(component_name, component_value, &image);
    break;

case 2:
    if(!valid_circle(radius, thickness, fill, color, fill_color)){
        exit(42);
    }
    draw_circle(center_x, center_y, radius, thickness, fill, color,
fill_color, &image);
    break;

case 3:
    if(!valid_split(split_x, split_y, thickness, color)){
        exit(42);
    }
    split(split_x, split_y, thickness, color, &image);
    break;

default:
    break;
}

write_png_file(path_output, &image);
}

int main(int argc, char* argv[]) //
{
    puts("Course work for option 4.23, created by ALEXander Romanov");
    handler(argc, argv);
}

```

```
    return 0;  
}
```

Название файла: Makefile

```
all:  
    gcc main.c -lpng -lm -o cw
```

ПРИЛОЖЕНИЕ Б ТЕСТИРОВАНИЕ

Фото для обработки – фильтр RGB компонент

```
./cw --input moevem_chan.png --output res.png --rgbfilter --  
component_name blue --component_value 125
```



Результат работы программы:



Фото для обработки – создание коллажа из исходного изображения

```
./cw --input 7.png --output res.pnd --circle --center 500.450 --radius 45  
--thickness 13 --color 255.1.1
```



Результат работы программы:



Фото для обработки – разделение изображения на $m \times n$ частей

```
./cw --input forgor.pfp --output res.png --split --number_x 3 --number_y  
7 --thickness 9 --color 255.255.1
```



Результат работы программы:

