**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](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 B. 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\*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 компнент**

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 значени.

**3.2 Рисование окружности**

Рисование круга осуществляется с помощью функции *draw\_circle()*, которая принимает на вход координаты центра окружности (х и у), 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 «Про-

граммирование». 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\nAvailiable 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\*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:

