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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Обработка изображений в Си

Студент гр. 3384 _____ Рудаков А.Л.

Преподаватель _____ Глазунов С.А.

Санкт-Петербург

2024

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студент Рудаков А.Л.

Группа 3384

Тема работы: Обработка изображений в Си

Исходные данные:
Обработать PNG изображение с помощью языка Си и выполнить задания, такие как: нарисовать окружность, для всего изображения изменить значение одной из rgb-компонент, разделить изображение на N*M частей.

Содержание пояснительной записки:
«Содержание», «Введение», «Заключение», «Список использованных источников».

Предполагаемый объем пояснительной записки:
Не менее 20 страниц.

Дата выдачи задания: 18.03.2024

Дата сдачи реферата: __.05.2024

Дата защиты реферата: 23.05.2024

Студент _____ Рудаков А.Л.

Преподаватель _____ Глазунов С.А.

АННОТАЦИЯ

Программа принимает флаги и аргументы из командой строки. При неверных введенных флагах или аргументах выводит соответствующую ошибку. При верно введенных запускает одну из 5 функций курсовой работы. Первая функция (флаг —circle) рисует на переданном изображении окружность в указанной точке центра, с указанным радиусом, цветом, толщиной, а так же закрашивает её, если это необходимо. Вторая функция (флаг —rgbfilter) заменяет одну rgb-компонент на всем изображении, ей на вход подаются название компоненты и значение, которое для неё требуется установить. Третья функция (флаг —split) разделяет изображение на N*M частей, ей на вход передаются количества частей по осям Y и X, толщина линии и её цвет. Четвертая функция (флаг —help) выводит информацию о программе. Пятая функция (флаг —info) выводит информацию о изображении.

SUMMARY

The program accepts flags and arguments from the command line. If incorrect flags or arguments are entered, it outputs the corresponding error. If entered correctly, it starts one of the 5 functions of the course work. The first function (the circle flag) draws a circle on the transmitted image at the specified center point, with the specified radius, color, thickness, and also fills it in, if necessary. The second function (flag —rgbfilter) replaces one rgb component in the entire image, the name of the component and the value that needs to be set for it are fed to it. The third function (the split flag) divides the image into N*M parts, the number of parts along the Y and X axes, the thickness of the line and its color are transmitted to it as input. The fourth function (the -help flag) displays information about the program. The fifth function (flag —info) displays information about the image.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель работы: разработать программу на языке Си, обрабатывающую изображения PNG формата.

Задачи: Для достижения поставленной цели требуется написать функции считывания и записи изображения, а так же разработать алгоритмы рисования окружности, замены rgb-компоненты и разделения изображения.

1. ПОДКЛЮЧАЕМЫЕ БИБЛИОТЕКИ

Для работы программы были подключены библиотеки: *stdio.h, unistd.h, getopt.h, string.h, stdlib.h, png.h.*

2. ФУНКЦИИ

2.1. Функции считывания и записи изображения

Функция *read_png* принимает на вход названия изображения, которое нужно считать, а так же указатель на структуру *png*. Программа делает проверки является ли формат открываемого изображения png, если нет, то выводит соответствующую ошибку, если да, то записывает информацию об изображении, а так же выделяет память и считывает попиксильно само изображение.

Функция *write_png* принимает на вход названия изображения, которое нужно записать, а так же указатель на структуру *png*. Функция создает пустое изображение и записывает в него данные (заголовок), а так же само полученное в ходе работы программы изображение. Очищает память, выделяемую для хранения изображения.

2.2. Функция рисования окружности и её вспомогательные функции

Функция *set_pixel* получает на вход координаты пикселя и значения rgb-компоненты, которую нужно ему задать, после чего, задает ему переданный цвет.

Функции *min* и *max* получают на вход 2 целых числа и возвращают наименьшее и наибольшее из них соответственно.

Функция *circle* получает на вход указатель на структуру *png*, координаты центра, радиус, толщину и цвет линии, флаг, определяющий нужна ли заливка, и цвет заливки. В функции используется *delta* — толщина, на которую нужно отступать в каждую сторону от единичной окружности заданного радиуса. По формуле окружности рисуется окружность с заданной толщиной, если требуется заливка, то заливка происходит сразу же. Цвета пикселям задаются при помощи вызова функции *set_pixel*.

2.3. Функция фильтра rgb-компонент

Функция *rgb_filter* получает на вход указатель на структуру *png*, название компоненты и её значение. Внутри происходит сравнение переданного

названия цвета с названиями цветов, после чего определенный нужный цвет меняет свое значение на переданное на всем изображении.

2.4. Функция разделения изображения на N*M частей

Функция *rgb_split* принимает на вход указатель на структуру *png*, количество частей по осям Y и X, толщину линий и их цвет. Далее по формуле определяются координаты линий и закрашиваются. Заливка пикселей происходит при помощи вызова функции *set_pixel*.

2.5. Функция main

В функции *main* при помощи *getopt_long* считываются переданные флаги и аргументы и проверяются на корректность, а также запускаются функции с переданными аргументами, которые соответствуют переданным флагам.

ЗАКЛЮЧЕНИЕ

Была реализована программа, которая использует систему CLI и в зависимости от переданных команд запускает рисование на изображении или обработку изображения, а так же сохраняет новое изображение после обработки.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

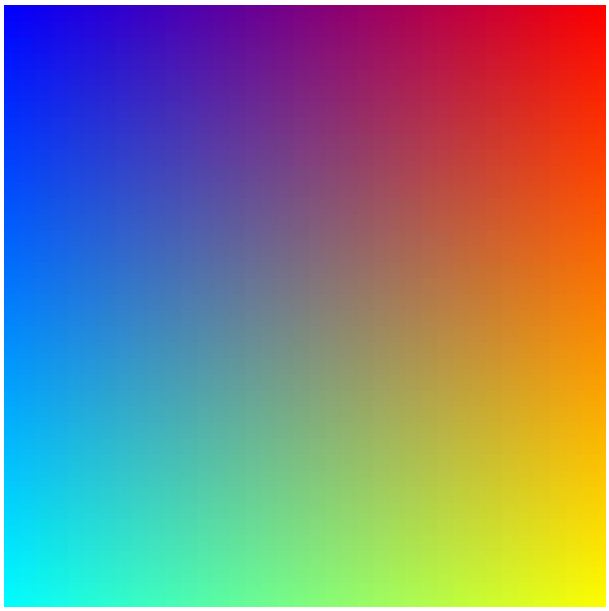
1. БАЗОВЫЕ СВЕДЕНИЯ К ВЫПОЛНЕНИЮ КУРСОВОЙ РАБОТЫ ПО ДИСЦИПЛИНЕ «ПРОГРАММИРОВАНИЕ». ВТОРОЙ СЕМЕСТР / сост.: М. М. Заславский, А. А. Лисс, А. В. Гаврилов, С. А. Глазунов, Я. С. Государкин, С. А. Тиняков, В. П. Голубева, К. В. Чайка, В. Е. Допира. СПб.: Изд-во СПбГЭТУ «ЛЭТИ», 2024. 36 с.

2. Электронный учебник для изучения Си и Си++ // geeksforgeeks. URL: https://www.geeksforgeeks.org/bresenhams-line-generation-algorithm/

ПРИЛОЖЕНИЕ А

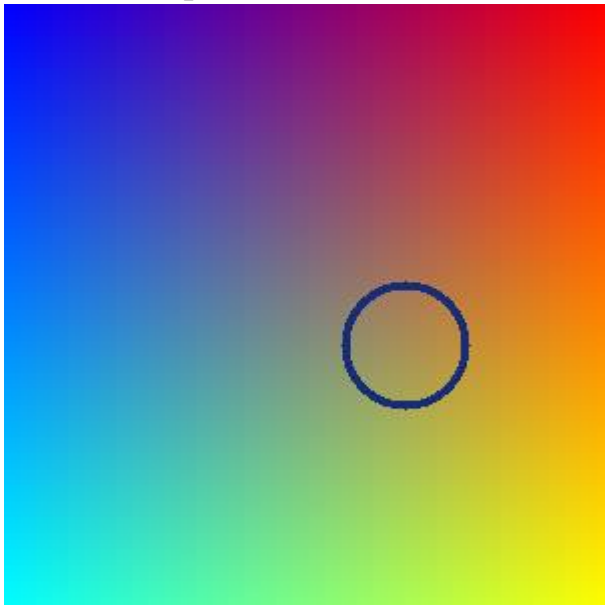
ТЕСТ

Входное изображение:



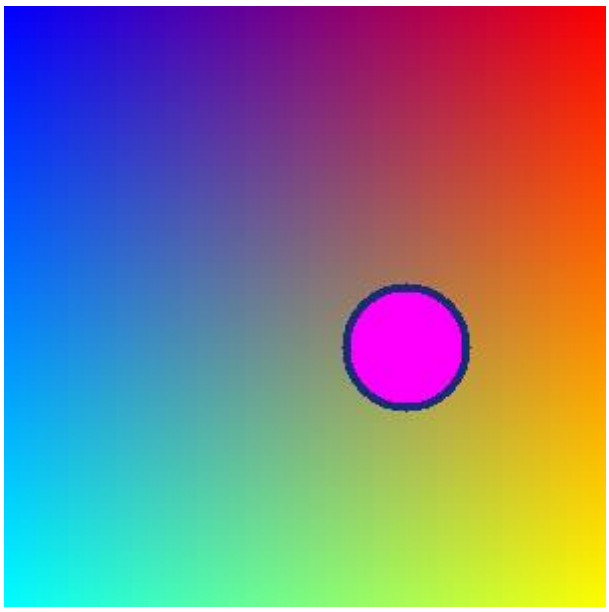
Тест 1: ./cw --circle --center 200.170 --radius 30 --thikness 5 --color 23.45.122 -i img.png -o out.png

Выходной файл:



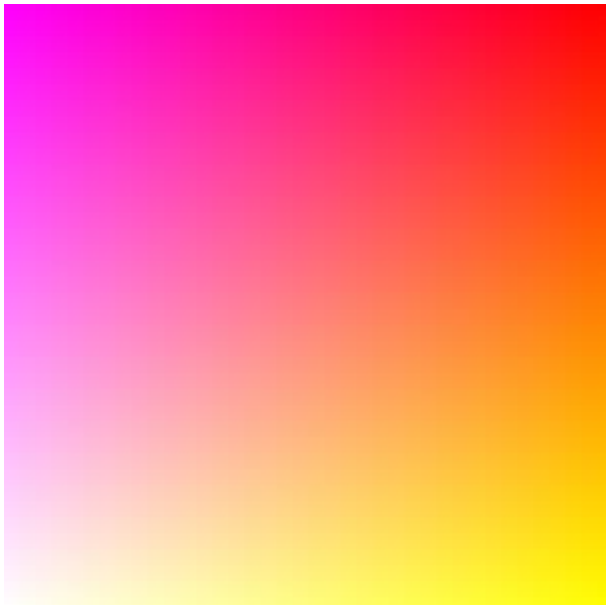
Тест 2: ./cw --circle --center 200.170 --radius 30 --thikness 5 --color 23.45.122 --fill --fill_color 255.0.255 -i img.png -o out.png

Выходной файл:



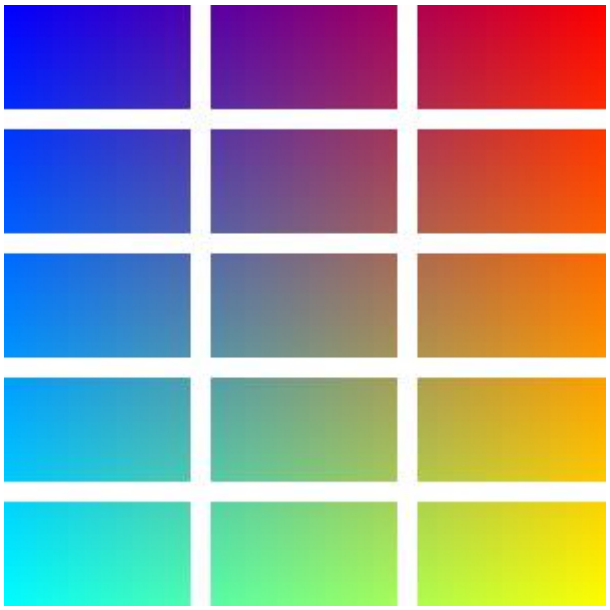
Тест 3: ./cw --rgbfilter --component_name red --component_value 255 -i img.png -o out.png

Выходной файл:



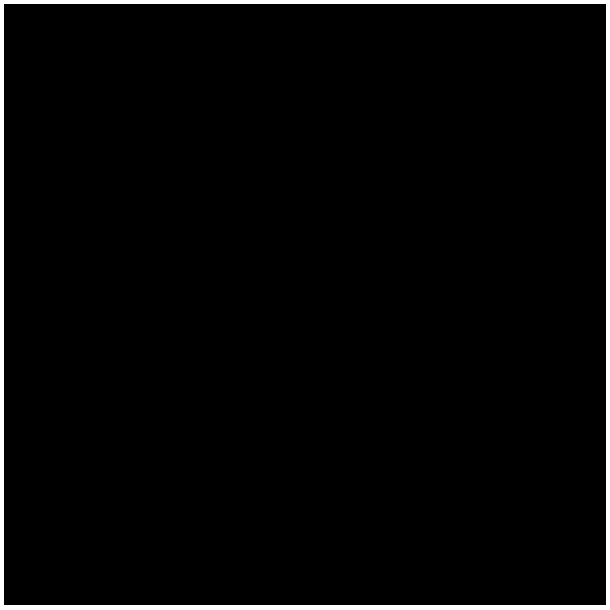
Тест 4: ./cw --split --number_x 3 --number_y 5 --thikness 10 --color 255.255.255 -i img.png -o out.png

Выходной файл:



Тест 5: ./cw --split --number_x 1000 --number_y 100 --thikness 10 --color 0.0.0 -i img.png -o out.png

Выходной файл:



ПРИЛОЖЕНИЕ Б
КОД

Название файла: additional_functions.h

```
#ifndef ADDITIONAL_FUNCTIONS
#define ADDITIONAL_FUNCTIONS

int min(int first, int second);

int max(int first, int second);

#endif
```

Название файла: additional_functions.c

```
int min(int first, int second){
    if (first > second){
        return second;
    } else {
        return first;
    }
}

int max(int first, int second){
    if (first > second){
        return first;
    } else {
        return second;
    }
}
```

Название файла: png_process.h

```
#ifndef PNG_PROCESS
#define PNG_PROCESS

#include <png.h>

typedef struct png{
    int width, height;
    png_structp png_ptr;
    png_infop info_ptr;
    png_byte color_type;
    png_byte bit_depth;
    int number_of_passes;
    png_bytep *row_pointers;
} png;

void inf(png* image);

void read_png(char *file_name, png *image);

void write_png(char *file_name, png *image);

void rgb_filter(png *image, char *component_name, int component_value);
```

```
void rgb_split(png *image, int number_x, int number_y, int thikness, int
red, int green, int blue);

void circle(png *image, int x0, int y0, int radius, int thikness, int red,
\
    int green, int blue, int fill, int fill_red, int fill_green, int
fill_blue);

#endif
```

Название файла: png_process.c

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <png.h>
#include "png_process.h"
#include "additional_functions.h"


void inf(png* image){
    printf("width - %d.\n", image->width);
    printf("height - %d.\n", image->height);
    printf("color type - %u.\n", image->color_type);
    printf("bit_depth - %u.\n", image->bit_depth);
}

void set_pixel(png *image, int x, int y, int red, int green, int blue){
    image->row_pointers[y][x * 3] = red;
    image->row_pointers[y][x * 3 + 1] = green;
    image->row_pointers[y][x * 3 + 2] = blue;
}

void read_png(char *file_name, png *image){
    FILE *fp = fopen(file_name, "rb");
    if (!fp){
        puts("Error: the file could be opened");
        exit(43);
    }

    char header[8];
    fread(header, 1, 8, fp);
    if (!png_check_sig(header, 8)){
        puts("Error: the file isn't PNG");
        exit(43);
    }

    image->png_ptr = png_create_read_struct(PNG_LIBPNG_VER_STRING, NULL,
NULL, NULL);
    if (!image->png_ptr){
        puts("Error: failed to create a structure of image");
        exit(42);
    }

    image->info_ptr = png_create_info_struct(image->png_ptr);
    if (!image->info_ptr){
```

```
        puts("Error: failed to create a structure of image");
        exit(42);
    }

    if (setjmp(png_jmpbuf(image->png_ptr))){
        puts("Error: image reading error");
        exit(42);
    }
    png_init_io(image->png_ptr, fp);
    png_set_sig_bytes(image->png_ptr, 8);
    png_read_info(image->png_ptr, image->info_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->number_of_passes = png_set_interlace_handling(image->png_ptr);
    png_read_update_info(image->png_ptr, image->info_ptr);

    image->row_pointers    =    (png_bytep*)malloc(sizeof(png_bytep)    *
image->height);
    for (size_t i = 0; i < image->height; i++){
        image->row_pointers[i]                                           =
(png_byte*)malloc(png_get_rowbytes(image->png_ptr, image->info_ptr));
    }
    png_read_image(image->png_ptr, image->row_pointers);

    fclose(fp);
}

void write_png(char *file_name, png *image){
    FILE *fp = fopen(file_name, "wb");
    if (!fp){
        puts("Error: the file could be opened");
        exit(43);
    }

    image->png_ptr = png_create_write_struct(PNG_LIBPNG_VER_STRING, NULL,
NULL, NULL);
    if (!image->png_ptr){
        puts("Error: failed to create a structure of image");
        exit(42);
    }

    image->info_ptr = png_create_info_struct(image->png_ptr);
    if (!image->info_ptr){
        puts("Error: failed to create a structure of image");
        exit(42);
    }

    if (setjmp(png_jmpbuf(image->png_ptr))){
        puts("Error: image recording error");
        exit(42);
    }
```

```
    png_init_io(image->png_ptr, fp);

    if (setjmp(png_jmpbuf(image->png_ptr))){
        puts("Error: image recording error");
        exit(42);
    }

    png_set_IHDR(image->png_ptr,      image->info_ptr,      image->width,
image->height, image->bit_depth, \
        image->color_type, PNG_INTERLACE_NONE, PNG_COMPRESSION_TYPE_BASE,
PNG_FILTER_TYPE_BASE);

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

    if (setjmp(png_jmpbuf(image->png_ptr))){
        puts("Error: image recording error");
        exit(42);
    }

    png_write_image(image->png_ptr, image->row_pointers);
    png_write_end(image->png_ptr, NULL);
    fclose(fp);
}

void rgb_filter(png *image, char *component_name, int component_value){
    if (strstr(component_name, "red")){
        for (int y = 0; y < image->height; y++){
            for (int x = 0; x < image->width; x++){
                image->row_pointers[y][x * 3] = component_value;
            }
        }
    } else if (strstr(component_name, "green")){
        for (int y = 0; y < image->height; y++){
            for (int x = 0; x < image->width; x++){
                image->row_pointers[y][x * 3 + 1] = component_value;
            }
        }
    } else if (strstr(component_name, "blue")){
        for (int y = 0; y < image->height; y++){
            for (int x = 0; x < image->width; x++){
                image->row_pointers[y][x * 3 + 2] = component_value;
            }
        }
    }                                                                  else
{
            //incorrect color
        puts("error: incorrect color");
        exit(0);
    }
}

void rgb_split(png *image, int number_x, int number_y, int thikness, int
red, int green, int blue){
    int line_x, line_y;
    line_x = (image->width - ((number_x - 1) * thikness)) / number_x;
    line_y = (image->height - ((number_y - 1) * thikness)) / number_y;
```

```
    if (line_x <= 0 || line_y <= 0){
        for (int y = 0; y < image->height; y++){
            for (int x = 0; x < image->width; x++){
                set_pixel(image, x, y, red, green, blue);
            }
        }
    } else {
        for (int i = line_x; i < image->width - thikness; i += line_x +
thikness){
            for (int x = i; x < i + thikness; x++){
                for (int y = 0; y < image->height; y++){
                    set_pixel(image, x, y, red, green, blue);
                }
            }
        }

        for (int i = line_y; i < image->height - thikness; i += line_y +
thikness){
            for (int y = i; y < i + thikness; y++){
                for (int x = 0; x < image->width; x++){
                    set_pixel(image, x, y, red, green, blue);
                }
            }
        }
    }
}

void circle(png *image, int x0, int y0, int radius, int thikness, int red,
\
    int green, int blue, int fill, int fill_red, int fill_green, int
fill_blue){

        int delta = thikness / 2;
        if (delta == 0){
            delta++;
        }
        for (int x = max(0, x0 - (radius + delta)); x <= min(image->width
- 1, x0 + (radius + delta)); x++){
            for (int y = max(0, y0 - (radius + delta)); y <=
min(image->height - 1, y0 + (radius + delta)); y++){
                int left = (x-x0) * (x-x0) + (y-y0) * (y-y0);
                int right_max = (radius + delta) * (radius + delta);
                int right_min = 0;
                if (delta < radius){
                    right_min = (radius - delta) * (radius - delta);
                }
                if (left <= right_max && left >= right_min){
                    set_pixel(image, x, y, red, green, blue);
                }
                if (fill != 0){
                    if (left < right_min){
                        set_pixel(image, x, y, fill_red, fill_green,
fill_blue);
                    }
                }
            }
        }
```

```
    }
```

Название файла: main.c

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <getopt.h>
#include <string.h>
#include "png_process.h"


typedef struct Data{
    int circle;
    int filter;
    int split;
    int x_center, y_center;
    int radius;
    int thikness;
    int red, green, blue;
    int fill;
    int fill_red, fill_green, fill_blue;
    char *component_name;
    int component_value;
    int number_x, number_y;
} Data;

/*void scan_circle_data(int argc, char **argv, circle *data){
    int opt;
    static struct option long_options[] = {
        {"center", 1, 0, 'C'},
        {"radius", 1, 0, 'R'},
        {"thikness", 1, 0, 'T'},
        {"color", 1, 0, 'L'},
        {"fill", no_argument, 0, 'F'},
        {"fill_color", 1, 0, 'O'}
    };

    data->fill = 0;
    char digit[] = "0123456789";
    int count = 0;

    while ((opt = getopt_long(argc, argv, "C:R:T:L:FO:", long_options,
NULL)) != -1){
        switch (opt){


            case 'c':
                if     (sscanf(optarg,     "%d.%d",     &(data->x_center),
&(data->y_center)) == 2){
                    char* dot = strstr(optarg, ".");
                    int add = 1;
                    if (*(dot + 1) == '-'){
                        add = 2;
                    }
                    if (strlen(optarg) == ((dot + strspn(dot + add, digit)
+ add) - optarg)){
```

```
                        printf("center   [%d,   %d]\n",   data->x_center,
data->y_center);                    //correct center
                    } else {
                        puts("error:       incorrect       arguments\n");
//incorrect center
                        exit(0);
                    }
                } else {
                    puts("error:         incorrect         arguments\n");
//incorrect center
                    exit(0);
                }

                count++;
                break;


            case 'r':
                if (sscanf(optarg, "%d", &(data->radius)) == 1){
                    if (strlen(optarg) == strspn(optarg, digit)){
                        if (data->radius > 0){
                            printf("radius    [%d]\n",    data->radius);
//correcr radius
                        } else{
                            puts("error:  radius  must  be  greater  than
0\n");
                            exit(0);
//radius <= 0
                        }
                    } else {
                        puts("error:       incorrect       arguments\n");
//incorrect radius
                        exit(0);
                    }
                } else {
                    puts("error:         incorrect         arguments\n");
//incorrect radius
                    exit(0);
                }

                count ++;
                break;


            case 't':
                if (sscanf(optarg, "%d", &(data->thikness)) == 1){
                    if (strlen(optarg) == strspn(optarg, digit)){
                        if (data->thikness >0){
                            printf("thikness   [%d]\n",   data->thikness);
//correct thikness
                        }
                        else{
                            puts("error: thikness must be greater than
0\n");                          //thikness <= 0
                            exit(0);
                        }
                    } else{
```

```
                        puts("error:       incorrect       arguments\n");
//incorrect thikness
                        exit(0);
                    }
                } else{
                    puts("error:         incorrect         arguments\n");
//incorrect thikness
                    exit(0);
                }

                count++;
                break;


            case 'l':
                if     (sscanf(optarg,     "%d.%d.%d",     &(data->red),
&(data->green), &(data->blue)) == 3){
                    char *first_dot = strstr(optarg, ".") + 1;
                    char *second_dot = strstr(first_dot, ".") + 1;
                    if  (strlen(optarg)  ==  (strspn(optarg,  digit)  +
strspn(first_dot, digit) + \
                        strspn(second_dot, digit) + 2)){
                            if  (data->red  >=  0  &&  data->green  >=  0  &&
data->blue >= 0 \
                                && data->red < 256 && data->green < 256 &&
data->blue < 256){           //correct color
                                    printf("color  rgb  [%d,  %d,  %d]\n",
data->red, data->green, data->blue);
                            } else {
                                puts("error:  rgb  components  must  be
greatest       than       0       and       less       than       256\n");
//components of color < 0
                                exit(0);
                            }
                    } else {
                        puts("error:       incorrect       arguments\n");
//incorrect color
                        exit(0);
                    }
                } else {
                    puts("error:         incorrect         arguments\n");
//incorrect color
                    exit(0);
                }

                count++;
                break;


            case 'f':
                data->fill = 1;

                count++;
                break;


            case 'o':
```

```
                if (data->fill == 0){
                    puts("error: none flag --fill");
                    exit(0);
                } else {
                    if  (sscanf(optarg,  "%d.%d.%d",  &(data->fill_red),
&(data->fill_green), &(data->fill_blue)) == 3){
                        char *first_dot = strstr(optarg, ".") + 1;
                        char *second_dot = strstr(first_dot, ".") + 1;
                        if  (strlen(optarg)  ==  (strspn(optarg,  digit)  +
strspn(first_dot, digit) + \
                            strspn(second_dot, digit) + 2)){
                                if     (data->fill_red     >=     0     &&
data->fill_green >= 0 && data->fill_blue >= 0 \
                                    &&    data->fill_red    <    256    &&
data->fill_green < 256 && data->fill_blue < 256){
                                        printf("color_fill            rgb
[%d, %d, %d]\n", data->fill_red,\
                                            data->fill_green,
data->fill_blue);                             //correct color_fill
                                } else {
                                    puts("error: rgb components must be
greatest       than       0       and       less       than       256\n");
//components of color < 0
                                    exit(0);
                                }
                        } else {
                            puts("error:     incorrect     arguments\n");
//incorrect color_fill
                            exit(0);
                        }
                    } else {
                        puts("error:       incorrect       arguments\n");
//incorrect color_fill
                        exit(0);
                    }
                }

                count++;
                break;


            default:
                puts("error:             incorrect             flags\n");
//incorrect flag
                exit(0);
        }
    }

    if (!((data->fill == 0 && count == 4) || (data->fill == 1 && count ==
6))){
        puts("error:    few    flags    have    been    intoduced\n");
//few flags
        exit(0);
    }
}*/

int main(int argc, char **argv){
```

```
puts("Course work for option 4.23, created by Rudakov Aleksandr");

if (argc == 1){
    puts("Error: no flags");
    exit(40);
}

int opt;
static struct option long_options[] = {
    {"help", no_argument, 0, 'h'},
    {"input", required_argument, 0, 'i'},
    {"output", required_argument, 0 ,'o'},
    {"info", no_argument, 0, 'f'},
    {"circle", no_argument, 0, 'c'},
    {"rgbfilter", no_argument, 0, 'r'},
    {"split", no_argument, 0, 's'},
    {"center", required_argument, 0, 'C'},
    {"radius", required_argument, 0, 'R'},
    {"thikness", required_argument, 0, 'T'},
    {"color", required_argument, 0, 'L'},
    {"fill", no_argument, 0, 'F'},
    {"fill_color", required_argument, 0, 'O'},
    {"component_name", required_argument , 0, 'N'},
    {"component_value", required_argument, 0, 'V'},
    {"number_x", required_argument, 0, 'X'},
    {"number_y", required_argument, 0, 'Y'}
};

Data data;
data.circle = 0;
data.filter = 0;
data.split = 0;
data.x_center = 0;
data.y_center = 0;
data.radius = 0;
data.thikness = 0;
data.red = -1;
data.green = -1;
data.blue = -1;
data.fill = 0;
data.fill_red = -1;
data.fill_green = -1;
data.fill_blue = -1;
data.component_value = -1;
data.number_x = 0;
data.number_y = 0;

char digit[] = "0123456789";
int count_circle = 0;
int count_filter = 0;
int count_split = 0;
int help_flag = 0;
int info_flag = 0;

char *input = NULL;
char *output = NULL;
```

```
    while ((opt = getopt_long(argc, argv, "i:o:crfC:R:T:L:FO:N:V:X:Y:",
long_options, NULL)) != -1){
        switch(opt){
            case 'h':
                puts(
                    "Error 40 - incorrect flags\n"
                    "Error 41 - incorrect arguments\n"
                    "Error 42 - the problem of working with image\n"
                    "Error 43 - incorrect file\n"
                    "\n"
                    "Flags:\n"
                    "--input - set the name of the input image\n"
                    "--output - set the name of the output image\n"
                    "--info - print information about the image\n"
                    "--help - display the help\n"
                    "--circle - draw a circle, defined by flags:\n\t--
center x.y\n\t--radius r\n"
                    "\t--thikness  thik\n\t--color  r.g.b\n\t--fill\n\t--
fill_color\n\tr.g.b\n"
                    "--rgbfilter - set the rgb-component value for the
entire image, defined by flags:\n"
                    "\t--component_name name\n\t--component_value val\n"
                    "--split - divide the image into N*M parts, defined by
flags:\n"
                    "\t--number_x   n_x\n\t--number_y   n_y\n\t--thikness
thik\n\t--color r.g.b"
                );
                help_flag = 1;
                exit(0);


            case 'i':
                input = (char*)malloc(sizeof(char) * (strlen(optarg) +
1));
                strncpy(input, optarg, strlen(optarg));
                input[strlen(optarg)] = '\0';
                break;


            case 'o':
                output = (char*)malloc(sizeof(char) * (strlen(optarg) +
1));
                strncpy(output, optarg, strlen(optarg));
                output[strlen(optarg)] = '\0';
                break;


            case 'f':
                info_flag = 1;
                break;

            case 'c':
                data.circle = 1;
                break;
```

```
            case 'r':
                data.filter = 1;
                break;


            case 's':
                data.split = 1;
                break;


            case 'C':
                if      (sscanf(optarg,      "%d.%d",      &data.x_center,
&data.y_center) == 2){
                    char* dot = strstr(optarg, ".");
                    int add = 1;
                    if (*(dot + 1) == '-'){
                        add = 2;
                    }
                    if (strlen(optarg) == ((dot + strspn(dot + add, digit)
+ add) - optarg)){          //correct center
                        count_circle++;
                        break;
                    }                                                   else
{                                                                   //incorrect
center
                        puts("Error: incorrect arguments");
                        exit(41);
                    }
                }                                                       else
{                                                                   //incorrect
center
                    puts("Error: incorrect arguments");
                    exit(41);
                }


            case 'R':
                if (sscanf(optarg, "%d", &data.radius) == 1){
                    if (strlen(optarg) == strspn(optarg, digit)){
                        if                  (data.radius                >
0){                                                               //correcr
radius
                            count_circle++;
                            break;
                        }
else{                                                                //radius
<= 0
                            puts("Error: radius must be greater than 0");
                            exit(41);
                        }
                    }                                                   else
{                                                                   //incorrect
radius
                        puts("Error: incorrect arguments");
                        exit(41);
                    }
```

```
                }                                                   else
{                                                                   //incorrect
radius
                    puts("Error: incorrect arguments");
                    exit(41);
                }


            case 'T':
                if (sscanf(optarg, "%d", &data.thikness) == 1){
                    if (strlen(optarg) == strspn(optarg, digit)){
                        if
(data.thikness >0){                                                 //correct
thikness
                            count_circle++;
                            count_split++;
                            break;
                        }

else{                                                               //thikness
<= 0
                            puts("Error: thikness must be greater than -
1");
                            exit(41);
                        }
                    }
else{
//incorrect thikness
                        puts("Error: incorrect arguments");
                        exit(41);
                    }
                }
else{
    //incorrect thikness
                    puts("Error: incorrect arguments");
                    exit(41);
                }


            case 'L':
                if (sscanf(optarg, "%d.%d.%d", &data.red, &data.green,
&data.blue) == 3){
                    char *first_dot = strstr(optarg, ".") + 1;
                    char *second_dot = strstr(first_dot, ".") + 1;
                    if (strlen(optarg) == (strspn(optarg, digit) +
strspn(first_dot, digit) + \
                        strspn(second_dot, digit) + 2)){
                            if (data.red >= 0 && data.green >= 0 &&
data.blue >= 0 \
                                && data.red < 256 && data.green < 256 &&
data.blue < 256){               //correct color
                                    count_circle++;
                                    count_split++;
                            }                                       else
{                                                                   //incorrect
color
```

```
                                puts("Error: rgb components must be
greatest than 0 and less than 256");
//components of color < 0
                                exit(41);
                            }
                        }                                                   else
{                                                                       //incorrect
color
                            puts("Error: incorrect arguments");
                            exit(41);
                        }
                    }                                                       else
{                                                                       //incorrect
color
                        puts("Error: incorrect arguments");
                        exit(41);
                    }

                    break;


                case 'F':
                    data.fill = 1;

                    count_circle++;
                    break;


                case 'O':
                    if    (sscanf(optarg,    "%d.%d.%d",    &data.fill_red,
&data.fill_green, &data.fill_blue) == 3){
                        char *first_dot = strstr(optarg, ".") + 1;
                        char *second_dot = strstr(first_dot, ".") + 1;
                        if  (strlen(optarg)  ==  (strspn(optarg,  digit)  +
strspn(first_dot, digit) + \
                            strspn(second_dot, digit) + 2)){
                                if (data.fill_red >= 0 && data.fill_green >=
0 && data.fill_blue >= 0 \
                                && data.fill_red < 256 && data.fill_green
< 256 && data.fill_blue < 256){   //correct color_fill
                                    count_circle++;
                                    break;
                            }                                               else
{                                                                       //incorrect
color_fill
                                puts("Error: rgb components must be
greatest than 0 and less than 256");
//components of color < 0
                                exit(41);
                            }
                        }                                                   else
{                                                                       //incorrect
color_fill
                            puts("Error: incorrect arguments");
                            exit(41);
                        }
```

```
                }                                                   else
{                                                                   //incorrect
color_fill
                    puts("Error: incorrect arguments");
                    exit(41);
                }


            case 'N':
                count_filter++;

                if                                          (!strcmp(optarg,
"red")){                                                            //correct
component_name
                    data.component_name = (char*)malloc(sizeof(char) * 4);
                    strncpy(data.component_name, optarg, 3);
                    data.component_name[3] = '\0';
                    break;
                }           else            if              (!strcmp(optarg,
"green")){                                                          //correct
component_name
                    data.component_name = (char*)malloc(sizeof(char) * 6);
                    strncpy(data.component_name, optarg, 5);
                    data.component_name[5] = '\0';
                    break;
                }           else            if              (!strcmp(optarg,
"blue")){                                                           //correct
component_name
                    data.component_name = (char*)malloc(sizeof(char) * 5);
                    strncpy(data.component_name, optarg, 4);
                    data.component_name[4] = '\0';
                    break;
                }                                                   else
{
    //incorrect component_name
                    puts("Error: incorrect arguments");
                    exit(41);
                }


            case 'V':
                if (sscanf(optarg, "%d", &data.component_value) == 1){
                    if (data.component_value >= 0 && data.component_value
< 256){
                        if      (strlen(optarg)     ==      strspn(optarg,
digit)){                                    //correct component_value
                            count_filter++;
                            break;
                        }                                           else
{                                                                   //incorrect
component_value
                            puts("Error: incorrect arguments");
                            exit(41);
                        }
                    }                                               else
{                                                                   //incorrect
component_value
```

```
                        puts("Error: component_value must be greater than
-1 and less than 256");
                        exit(41);
                    }
                }                                                   else
{
   //incorrect component_value
                    puts("Error: incorrect arguments");
                    exit(41);
                }


            case 'X':
                if (sscanf(optarg, "%d", &data.number_x) == 1){
                    if (strlen(optarg) == strspn(optarg, digit)){
                        if                  (data.number_x                  >
0){                                                          //correct
number_x
                            count_split++;
                            break;
                        }                                               else
{                                                          //incorrect
number_x
                            puts("Error:  number_x  must  be  greater  then
0");
                            exit(41);
                        }
                    }                                                   else
{                                                          //incorrect
number_x
                        puts("Error: incorrect argument");
                        exit(41);
                    }
                }
else{
        //incorrect number_x
                    puts("Error: incorrect arguments");
                    exit(41);
                }


            case 'Y':
                if (sscanf(optarg, "%d", &data.number_y) == 1){
                    if (strlen(optarg) == strspn(optarg, digit)){
                        if                  (data.number_y                  >
0){                                                          //correct
number_y
                            count_split++;
                            break;
                        }                                               else
{                                                          //incorrect
number_y
                            puts("Error:  number_y  must  be  greater  then
0");
                            exit(41);
                        }
```

```
                    }                                                   else
{                                                                   //incorrect
number_y
                        puts("Error: incorrect argument");
                        exit(41);
                    }
                }
else{
        //incorrect number_y
                    puts("Error: incorrect arguments");
                    exit(41);
                }


            default:
                puts("Error:              incorrect              flags");
//incorrect flag
                exit(40);
        }
    }

    png image;
    read_png(input, &image);

    if (info_flag == 1){
        inf(&image);
        exit(0);
    }

    if (data.fill == 0 && data.fill_red != -1){
        puts("Error: there are too many flags");
        exit(40);
    }

    if (data.circle == 1 && !((data.fill == 0 && count_circle == 4) ||
(data.fill == 1 && count_circle == 6))){ //few flags
        puts("Error: few flags have been intoduced");
        exit(40);
    }
    if      (data.filter      ==      1      &&      count_filter      !=
2){                                                                 //few flags
        puts("Error: few flags have been intoduced");
        exit(40);
    }
    if      (data.split      ==      1      &&      count_split      !=
4){                                                                 //few
flags
        puts("Error: few flags have been intoduced");
        exit(40);
    }
    if     (data.circle     +     data.filter     +     data.split     >
1){                                                                 //too many flags
        puts("Error: there are too many flags");
        exit(40);
    }
    if (data.circle + data.filter + data.split == 0){
        puts("Error: few flags have been intoduced");
```

```
        exit(40);
    }

    if (data.circle == 1){
        circle(&image,   data.x_center,   data.y_center,   data.radius,
data.thikness, data.red, \
           data.green,     data.blue,     data.fill,     data.fill_red,
data.fill_green, data.fill_blue);
    }

    if (data.filter == 1){
        rgb_filter(&image, data.component_name, data.component_value);
    }

    if (data.split == 1){
        rgb_split(&image,  data.number_x,  data.number_y,  data.thikness,
data.red, data.green, data.blue);
    }

    if (help_flag == 0){
        write_png(output, &image);
    }

    for (int y = 0; y < image.height; y++)
         free(image.row_pointers[y]);
    free(image.row_pointers);
}
```

Название файла: Makefile

```
all: main.o png_process.o additional_functions.o
     gcc main.o png_process.o additional_functions.o -lpng -o cw

main.o: main.c png_process.h
     gcc -c main.c

png_process.o: png_process.c png_process.h additional_functions.h
     gcc -c png_process.c

additional_functions.o: additional_functions.c additional_functions.h
     gcc -c additional_functions.c

clear:
     rm *.o
```