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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Работа с изображениями в Си

Студентка гр. 3384 _____ Конасова Я. О.

Преподаватель _____ Глазунов С. А.

Санкт-Петербург

2024

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студентка Конасова Я. О.

Группа 3384

Тема работы: Работа с изображениями в Си

Исходные данные:

Вариант 4.20
Необходимо написать программу, которая считывает исходное изображение, определенным образом изменяет его и сохраняет новое изображение. Функции, которые необходимо реализовать: рисование линии, отображение заданной области, рисование пентаграммы.

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

Предполагаемый объем пояснительной записки:
Не менее 30 страниц.

Дата выдачи задания: 18.03.2024

Дата сдачи реферата: 17.05.2024

Дата защиты реферата: 22.05.2024

Студентка _____ Конасова Я.О

Преподаватель _____ Глазунов С.А

АННОТАЦИЯ

Разработана программа, которая при помощи функции *getopt* считывает набор флагов из консоли. На основании введенных данных программа считывает входное изображение, изменяет его при помощи одной из 3 задаваемых функций, а затем сохраняет новое полученное изображение. Пользователь может ввести оду из 3 функций на выбор, а также необходимые для ее реализации флаги: рисование линии, отображение заданной области, рисование пентаграммы.

Пример работы программы приведен в приложении А.

Исходный код программы приведен в приложении Б.

SUMMARY

A program has been developed that uses the getopt function to read a set of flags from the console. Based on the entered data, the program reads the input image, changes it using one of the 3 preset functions, and then saves the new received image. The user can enter an ode of 3 functions to choose from, as well as the flags necessary for its implementation: drawing a line, displaying a given area, drawing a pentagram.

An example of how the program works is given in Appendix A.

The source code of the program is given in Appendix B.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель работы: Написать программу на языке Си, которая считывает изображение, изменяет его и сохраняет новую версию.

Задачи, которые необходимо для этого решить:

1) Реализовать функции для считывания вводных данных.
2) Реализовать функции для корректного считывания изображения, а также его сохранения.
3) Реализовать функции для обработки изображения.

1. СЧИТЫВАНИЕ ДАННЫХ

1.1. Работа с флагами

В программе используется функция *getopt_long* для обработки флагов и считывания аргументов командной строки. Определение флагов: вначале создается массив *long_option*, который содержит структуру *option* для каждого флага. Каждый флаг имеет свое имя, информацию о необходимости аргумента (*no_argument* или *required_argument*), указатель на переменную (в данном случае 0) и короткий вариант флага (например, *'l'* для *--line*).

Считывание аргументов: Функция *getopt_long* используется для итерации по аргументам командной строки и обработки каждого флага. Внутри цикла *switch(opt)* проверяет значение *opt* и выполняет соответствующие действия.

1.2. Обработка ошибок

В цикле *while*, где происходит чтение флагов и их аргументов с помощью *getopt_long*, проверяются различные ошибки ввода. Если пользователь вводит некорректные данные, программа выводит сообщение об ошибке и завершает выполнение с соответствующим кодом ошибки. Например, некорректный аргумент флага *--start* или *–end,* некорректный аргумент флага *--color,* некорректный аргумент флага *–axis*. Также происходит проверка уникальности имен входного и выходного файлов. Если имена входного и выходного файлов совпадают, программа выводит сообщение об ошибке и завершает выполнение. Также есть обработка ошибок при работе с изображением. Перед выполнением операций с изображением программа проверяет наличие входного файла. Если входной файл не указан, программа должна корректно обработать этот случай. Также, после выполнения операций с изображением, программа освобождает память, занятую изображением.

2. РАБОТА С ИЗОБРАЖЕНИЕМ

2.1. Считывание изображения

Сначала функция пытается открыть файл *PNG* для чтения в бинарном режиме *("rb").* Если файл не удается открыть, выводится сообщение об ошибке, и программа завершает свою работу с кодом ошибки 41*:* Затем выделяется память для структуры *Png*. Если выделить память не удалось, то освобождается ресурс файла (если он был открыт) и выводится сообщение об ошибке о невозможности выделения памяти. Проверка сигнатуры *PNG*: считываются первые 8 байт файла, и проверяется, соответствует ли сигнатура файлу *PNG*. Если это не так, то файл закрывается, память освобождается и выводится сообщение об ошибке. Инициализация структур *png_struct* и *png_info*: Создаются структуры *png_struct* и *png_info*, необходимые для работы с изображением *PNG*. Если не удалось создать эти структуры, выводится сообщение об ошибке и освобождается выделенная память. Установка начала чтения файла: Устанавливается начало чтения файла с помощью функции. Чтение информации об изображении: Считывается информация об изображении, такая как ширина, высота, тип цвета и глубина цвета. Выделение памяти для строк изображения: Выделяется память для хранения строк изображения. Чтение строк изображения: Происходит чтение строк изображения и их сохранение в массив *row_pointers*. Закрытие файла и возврат структуры *Png*: Файл закрывается, и возвращается указатель на структуру *Png*.

2.2. Сохранение изображения

Открытие файла для записи: Сначала функция пытается открыть файл для записи в бинарном режиме *("wb").* Если не удается открыть файл, выводится сообщение об ошибке и программа завершает свою работу с кодом ошибки 48. Затем создаются структуры *png_str*uct и *png_info* для записи *PNG* изображения. Если не удается создать эти структуры, то закрывается файл, выводится сообщение об ошибке и программа завершает свою работу с кодом ошибки 47

или 46 в зависимости от причины ошибки. Обработка ошибок *libpng*: При помощи *setjmp* устанавливается точка возврата для обработки ошибок *libpng*. Если произошла ошибка, программа закрывает файл, освобождает память, выводит сообщение об ошибке и завершает свою работу с кодом ошибки 45. Инициализация записи PNG изображения: Устанавливаются параметры изображения с помощью функции *png_set_IHDR*. Выполняется запись информации об изображении. Происходит запись строк изображения в файл, завершается процесс записи PNG изображения, файл закрывается.

3. РЕАЛИЗАЦИЯ ФУНКЦИЙ

3.1. Функция draw_line

Вычисляются разности между координатами *start_x* и *end_x*, а также между *start_y* и *end_y*. Вычисляется длина отрезка с помощью формулы длины вектора в двумерном пространстве. Вектор отрезка нормализуется, чтобы получить единичный вектор, указывающий направление линии. Вычисляется половина толщины линии. Проходим по всей длине отрезка с шагом 0.5. Вычисляется текущая точка на линии с помощью нормализованного вектора отрезка. Вычисляется вектор, перпендикулярный вектору отрезка, который используется для определения границы линии. Проходим по всей толщине линии с шагом 0.5. Вычисляется точка на границе линии с учетом нормали и текущей точки. Координаты точки на границе линии округляются до целых чисел. Проверяется, что координаты находятся в пределах размеров изображения. Если точка находится в пределах изображения, то цвет пикселя устанавливается в соответствии с переданным цветом. В зависимости от типа цвета изображения (RGB или RGBA) используется соответствующее количество компонент цвета.

3.2. Функция mirror

Эта функция позволяет отражать определенную область изображения относительно горизонтальной или вертикальной оси. Функция сначала проверяет тип цвета изображения, чтобы определить, какую информацию использовать для обработки каждого пикселя. Проверяется, что координаты левого верхнего угла области *(left, up)* не больше координат правого нижнего угла *(right, down).* Если указана горизонтальная ось, функция проходит через каждый пиксель в половине высоты указанной области и меняет его местами с пикселем, расположенным симметрично относительно горизонтальной оси. Если указана вертикальная ось, функция проходит через каждый пиксель в половине ширины указанной области и меняет его местами с пикселем, расположенным симметрично относительно вертикальной оси. Для каждого

канала цвета (*RGB* или *RGBA*) временный буфер *temp* используется для обмена значениями цвета пикселей. Функция использует *memcpy* для копирования данных цвета пикселей, что позволяет эффективно менять пиксели местами. Если указана некорректная ось для отражения, функция выдает сообщение об ошибке.

3.3. Функция draw_pentagram

Эта функция обеспечивает рисование пентаграммы на заданном изображении в заданных координатах с указанным радиусом, толщиной линий и цветом. Расчет радиуса: Первым шагом функция увеличивает радиус на половину толщины (*thickness/2*). Это сделано для того, чтобы пентаграмма рисовалась на расстоянии *thickness/2* от указанного радиуса. Рисование окружности: Вложенные циклы проходят через прямоугольную область с центром в заданных координатах *center* и шириной и высотой, равными *radius* * 2. Внутри циклов проверяется, находится ли текущий пиксель внутри кольца, определенного вокруг центра с радиусом *radius* и толщиной *thickness*. Если пиксель находится внутри этого кольца, ему назначается цвет, заданный параметром *color*, с помощью функции *draw_point.*: Затем вычисляются координаты вершин пентаграммы, используя геометрические вычисления на основе радиуса и угловых координат. После того как координаты вершин пентаграммы вычислены, функция рисует линии, соединяющие эти вершины, используя функцию draw_line. Каждая линия соединяет вершину с индексом i с вершиной с индексом (i + 2) % 5, чтобы создать форму пентаграммы.

ЗАКЛЮЧЕНИЕ

В данной работе была разработана и реализована программа для обработки изображений формата *PNG* с использованием библиотеки *libpng*. Программа включает в себя несколько ключевых функций для работы с изображениями: чтение и запись *PNG* файлов, а также рисование геометрических объектов, а именно: рисование линии и пентаграммы в круге. Также реализована функция отображения заданной области. В коде также реализована командная строка для взаимодействия с пользователем. С помощью различных флагов и аргументов командной строки можно задать необходимые параметры для изображения и его преобразования. Программа обеспечивает корректную обработку изображений с различными типами цветовых данных (*RGB* и *RGBA*), а также включает в себя механизмы обработки ошибок и проверки корректности входных данных.

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/(дата обращения: 16.042024).

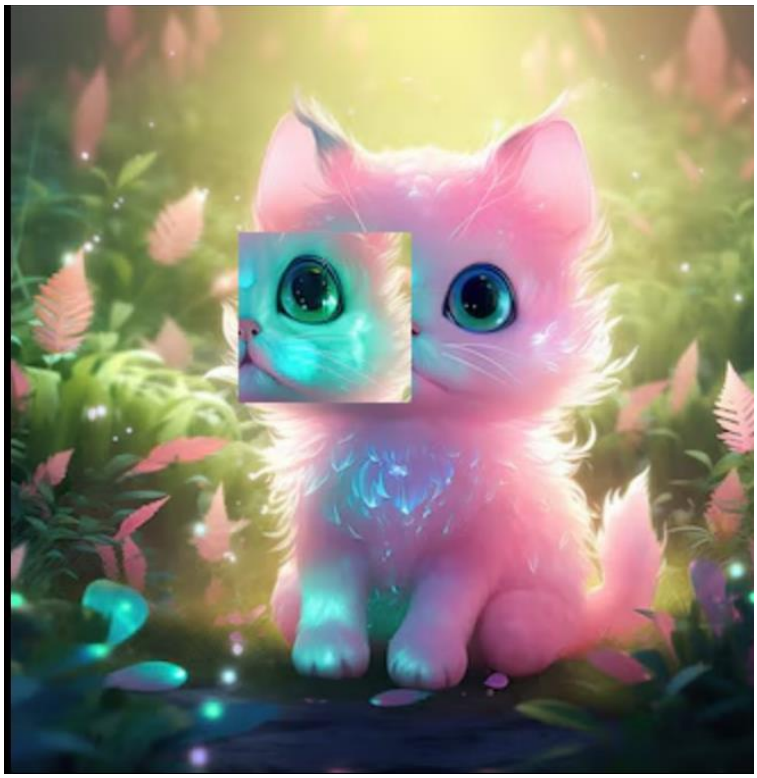
ПРИЛОЖЕНИЕ А
ТЕСТИРОВАНИЕ ПРОГРАММЫ

Исходное изображение:

Примеры запуска программы:

```
$ ./cw --mirror --axis x --left_up 100.100 --right_down 175.175 --output lol.png --input test8.png
```

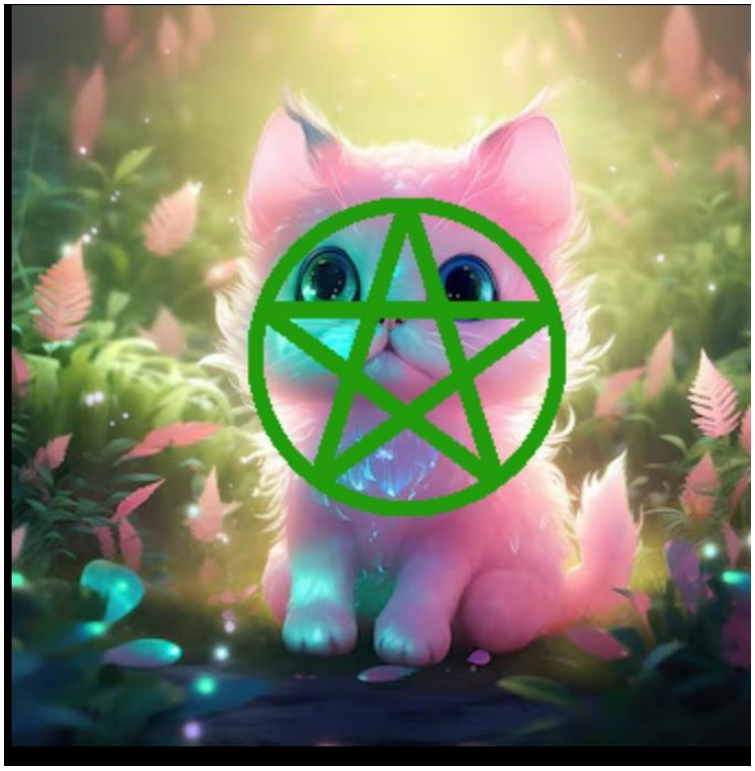
Результат работы программы:



Примеры запуска программы:

```
./cw --pentagram --center 180.160 --radius 70 --color 34.155.12 --thickness 7  --output lol.png --input test8.png
```

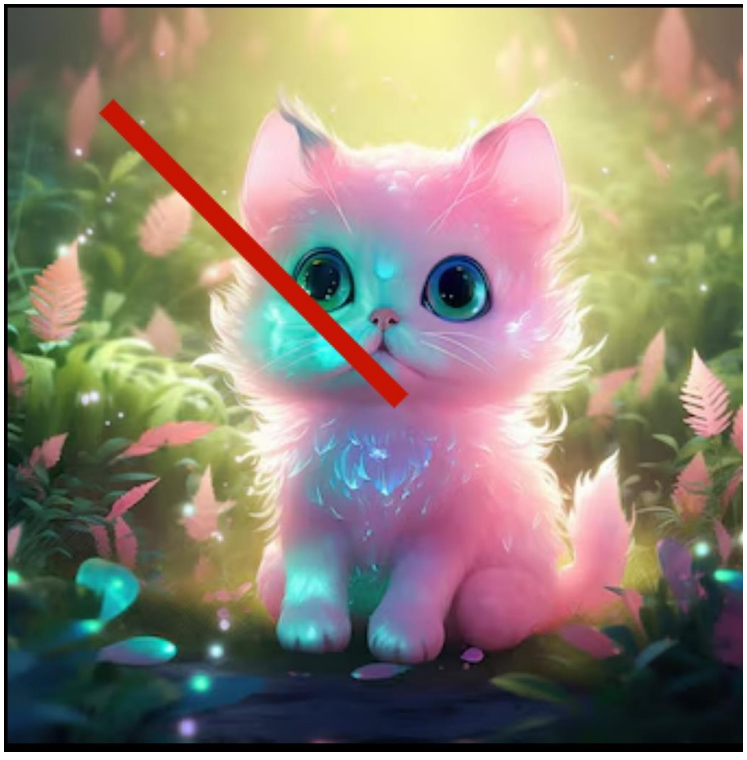
Результат работы программы:



Примеры запуска программы:

```
./cw --line --start 45.45 --end 180.180 --color 200.20.2 --thickness 8 --output lol.png --input test8.png
```

Результат работы программы:



ПРИЛОЖЕНИЕ В

КОД ПРОГРАММЫ

```
#include <stdio.h>
#include <stdlib.h>
#include <getopt.h>
#include <png.h>
#include <unistd.h>
#include <string.h>
#include <math.h>
#define M_PI 3.14159265358979323846

typedef struct{
    int red;
    int green;
    int blue;
}Color;

typedef struct {
    int x;
    int y;
} Point;

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

void process_file(struct Png *png) {
    int x,y;
    if (png_get_color_type(png->png_ptr, png->info_ptr)
!=PNG_COLOR_TYPE_RGBA){
        return;
    }
    for (y = 0; y < png->height; y++) {
        png_byte *row = png->row_pointers[y];
        for (x = 0; x < png->width; x++) {
            png_byte *ptr = &(row[x * 4]);
            if (ptr[3] == 0){
                ptr[3] = 255;
            }
        }
    }
 // Изменение альфа канала

}

struct Png *read_png(const char *filename) {
    FILE *fp = fopen(filename, "rb");
    if (!fp) {
        printf("Не удалось открыть файл %s для чтения\n", filename);
        exit(41);
```

```
    }

    struct Png *png = malloc(sizeof(struct Png));
    if (!png) {
        fclose(fp);
        printf("Ошибка выделения памяти\n");
        exit(41);
    }

    png_byte header[8];

// Считываем первые 8 байт файла
    fread(header, 1, 8, fp);

// Проверяем, является ли файл PNG
    if (png_sig_cmp(header, 0, 8)) {
        fclose(fp);
        free(png);
        printf("Файл %s не является PNG изображением\n", filename);
        exit(41);
    }

// Создаем структуру png_struct для работы с PNG изображением
    png->png_ptr = png_create_read_struct(PNG_LIBPNG_VER_STRING, NULL,
NULL, NULL);
    if (!png->png_ptr) {
        fclose(fp);
        free(png);
        printf("Ошибка создания структуры png_struct\n");
        exit(41);
    }

    png->info_ptr = png_create_info_struct(png->png_ptr);
    if (!png->info_ptr) {
        fclose(fp);
        free(png);
        printf("Ошибка создания структуры png_info\n");
        exit(49);
    }

// Устанавливаем начало чтения файла
    png_init_io(png->png_ptr, fp);
    png_set_sig_bytes(png->png_ptr, 8);

    png_read_info(png->png_ptr, png->info_ptr);
    png->width = png_get_image_width(png->png_ptr, png->info_ptr);
    png->height = png_get_image_height(png->png_ptr, png->info_ptr);
    png->color_type = png_get_color_type(png->png_ptr, png->info_ptr);
    png->bit_depth = png_get_bit_depth(png->png_ptr, png->info_ptr);


    png->number_of_passes = png_set_interlace_handling(png->png_ptr);
    png_read_update_info(png->png_ptr, png->info_ptr);


// Считываем строки изображения и сохраняем их
```

```
    png->row_pointers = (png_bytep *)malloc(sizeof(png_bytep) * png-
>height);
    for (int y = 0; y < png->height; y++) {
        png->row_pointers[y] = (png_byte *)malloc(png_get_rowbytes(png-
>png_ptr, png->info_ptr));
    }

    png_read_image(png->png_ptr, png->row_pointers);

    fclose(fp);
    process_file(png);
    return png;
}
void save_png(const char *filename, struct Png *png) {
    FILE *fp = fopen(filename, "wb");
    if (!fp) {
        printf("Ошибка открытия файла %s для записи\n", filename);
        exit(48);
    }


    png_structp png_ptr = png_create_write_struct(PNG_LIBPNG_VER_STRING,
NULL, NULL, NULL);
    if (!png_ptr) {
        fclose(fp);
        printf("Ошибка создания структуры png_struct для записи\n");
        exit(47);
    }

    png_infop info_ptr = png_create_info_struct(png_ptr);
    if (!info_ptr) {
        fclose(fp);
        png_destroy_write_struct(&png_ptr, NULL);
        printf("Ошибка создания структуры png_info для записи\n");
        exit(46);
    }

    if (setjmp(png_jmpbuf(png_ptr))) {
        fclose(fp);
        png_destroy_write_struct(&png_ptr, &info_ptr);
        printf("Ошибка записи PNG изображения\n");
        exit(45);
    }

    png_init_io(png_ptr, fp);
    int color_type = PNG_COLOR_TYPE_RGB;
    if (png->color_type == PNG_COLOR_TYPE_RGBA) {
        color_type = PNG_COLOR_TYPE_RGBA;
    }

    png_set_IHDR(png_ptr, info_ptr, png->width, png->height, png-
>bit_depth, color_type, PNG_INTERLACE_NONE, PNG_COMPRESSION_TYPE_DEFAULT,
PNG_FILTER_TYPE_DEFAULT);

    png_write_info(png_ptr, info_ptr);
```

```
    png_write_image(png_ptr, png->row_pointers);

    png_write_end(png_ptr, NULL);

    fclose(fp);
}

void draw_line(struct Png *png, float start_x, float start_y, float
end_x, float end_y, Color color, int thickness) {
    // Вычисляем разницу между координатами начала и конца отрезка
    float dx = end_x - start_x;
    float dy = end_y - start_y;

    // Вычисляем длину отрезка
    float length = sqrt(dx * dx + dy * dy);

    // Нормализуем вектор отрезка
    float norm_dx = dx / length;
    float norm_dy = dy / length;

    // Половина толщины
    float half_thickness = thickness / 2.0f;

    // Цикл по длине отрезка
    for (float i = 0; i < length; i += 0.5f) {
        // Текущая точка на линии
        float current_x = start_x + norm_dx * i;
        float current_y = start_y + norm_dy * i;

        // Вычисляем нормаль к линии
        float normal_x = -norm_dy;
        float normal_y = norm_dx;

        png_byte red = color.red;
        png_byte green = color.green;
        png_byte blue = color.blue;


        // Цикл по толщине линии
        for (float j = -half_thickness; j <= half_thickness; j += 0.5f) {
            // Точка на границе линии
            float border_x = current_x + normal_x * j;
            float border_y = current_y + normal_y * j;

            // Округляем координаты до целых
            int px = (int)round(border_x);
            int py = (int)round(border_y);

            // Проверяем, находится ли точка в пределах изображения
            if (px >= 0 && px < png->width && py >= 0 && py < png-
>height) {
                // Устанавливаем цвет пикселя
                if (png_get_color_type(png->png_ptr, png->info_ptr)
!=PNG_COLOR_TYPE_RGBA){
                    png_byte *ptr = &(png->row_pointers[py][px * 3]);

                    ptr[0] = red;
```

```
                    ptr[1] = green;
                    ptr[2] = blue;

                }
                if (png_get_color_type(png->png_ptr, png->info_ptr)
==PNG_COLOR_TYPE_RGBA){
                    png_byte *ptr = &(png->row_pointers[py][px * 4]);

                    ptr[0] = red;
                    ptr[1] = green;
                    ptr[2] = blue;
                    ptr[3] = 255;
                }

            }
        }
    }
}

void mirror_region(struct Png *png, char axis, int left, int up, int
right, int down) {
    int x, y;


    if (png_get_color_type(png->png_ptr, png->info_ptr)
==PNG_COLOR_TYPE_RGBA){
        png_byte temp[4];


        if (left >= right || up >= down) {
            printf("Некорректно задана область для отражения\n");
            return;
        }

    // Отражение относительно горизонтальной оси
        if (axis == 'y') {
            for (y = up; y < (up + down) / 2; y++) {
                for (x = left; x < right; x++) {

                    memcpy(temp, &(png->row_pointers[y][x * 4]), 4);
                    memcpy(&(png->row_pointers[y][x * 4]), &(png-
>row_pointers[down - (y - up)][x * 4]), 4);
                    memcpy(&(png->row_pointers[down - (y - up)][x * 4]),
temp, 4);
                }
            }
        }
    // Отражение относительно вертикальной оси
        else if (axis == 'x') {
            for (x = left; x < (left + right) / 2; x++) {
                for (y = up; y < down; y++) {
                    memcpy(temp, &(png->row_pointers[y][x * 4]), 4);
                    memcpy(&(png->row_pointers[y][x * 4]), &(png-
>row_pointers[y][(right - (x - left)) * 4]), 4);
                    memcpy(&(png->row_pointers[y][(right - (x - left)) *
4]), temp, 4);
                }
```

```
            }
        }
        else {
            printf("Некорректно указана ось для отражения\n");
            return;
        }
    }

    //for RGB

    if (png_get_color_type(png->png_ptr, png->info_ptr)
!=PNG_COLOR_TYPE_RGBA){
        png_byte temp[3];

        if (left >= right || up >= down) {
            printf("Некорректно задана область для отражения\n");
            return;
        }

    // Отражение относительно горизонтальной оси
        if (axis == 'y') {
            for (y = up; y < (up + down) / 2; y++) {
                for (x = left; x < right; x++) {

                    memcpy(temp, &(png->row_pointers[y][x * 3]), 3);
                    memcpy(&(png->row_pointers[y][x * 3]), &(png-
>row_pointers[down - (y - up)][x * 3]), 3);
                    memcpy(&(png->row_pointers[down - (y - up)][x * 3]),
temp, 3);
                }
            }
        }
    // Отражение относительно вертикальной оси
        else if (axis == 'x') {
            for (x = left; x < (left + right) / 2; x++) {
                for (y = up; y < down; y++) {
                    memcpy(temp, &(png->row_pointers[y][x * 3]), 3);
                    memcpy(&(png->row_pointers[y][x * 3]), &(png-
>row_pointers[y][(right - (x - left)) * 3]), 3);
                    memcpy(&(png->row_pointers[y][(right - (x - left)) *
3]), temp, 3);
                }
            }
        }
        else {
            printf("Некорректно указана ось для отражения\n");
            return;
        }
    }
}

float distance(Point p1, Point p2) {
    float dx = p2.x - p1.x;
    float dy = p2.y - p1.y;
    return sqrt(dx * dx + dy * dy);
}
```

```
void draw_point(struct Png *png, int x, int y, Color color){
    if (x >= 0 && x < png->width && y >= 0 && y < png->height) {
        if (png_get_color_type(png->png_ptr, png->info_ptr) ==
PNG_COLOR_TYPE_RGBA) {
            png_byte *ptr = &(png->row_pointers[y][x * 4]);
            ptr[0] = color.red;
            ptr[1] = color.green;
            ptr[2] = color.blue;
            ptr[3] = 255;
        } else if (png_get_color_type(png->png_ptr, png->info_ptr) !=
PNG_COLOR_TYPE_RGBA) {
            png_byte *ptr = &(png->row_pointers[y][x * 3]);
            ptr[0] = color.red;
            ptr[1] = color.green;
            ptr[2] = color.blue;
        }
    }
}


void draw_pentagram(struct Png *png, Point center, int radius, int
thickness, Color color) {
    radius=radius+(thickness/2);
    for(int i = center.x-radius; i<center.x+radius; i++){
        for(int j=center.y - radius; j<center.y+radius; j++){
            if(((center.x - i)*(center.x-i)+(center.y-j)*(center.y-
j)>(radius*radius)-2*thickness*radius)&&((center.x-i)*(center.x-
i)+(center.y-j)*(center.y-j)<(radius*radius))){
                draw_point(png, i, j, color);
            }
        }
    }

    Point points[5];
    for (int i = 0; i < 5; i++) {
        float phi = (M_PI / 5.0) * (2.0 * i + (3.0 / 2.0));
        points[i].x = center.x + round((radius-thickness/2) * cos(phi));
        points[i].y = center.y + round((radius-thickness/2) * sin(phi));
    }

    // Рисуем линии пентаграммы
    for (int i = 0; i < 5; i++) {
        draw_line(png, points[i].x, points[i].y, points[(i + 2) % 5].x,
points[(i + 2) % 5].y, color, thickness);
    }
}

int main(int argc, char *argv[]){
    int opt;
    int option_index=0;
    int will_draw_line=0;
    int will_mirror=0;
    int will_draw_pentagram=0;
    int start_x, start_y;
    Color color = {0, 0, 0};
    int end_x, end_y;
    int thickness_of_line=0;
```

```
    char wich_mirror;
    int up_x, up_y;
    int right_x, right_y;
    Point center = {0.0, 0.0};
    int radius;
    int print_info=0;
    char* png_name_of_file=NULL;
    char* png_out_name=NULL;
    static struct option long_option[]={
        {"line", no_argument, 0, 'l'},
        {"start", required_argument, 0, 's'},
        {"end", required_argument, 0, 'e'},
        {"color", required_argument, 0, 'k'},
        {"thickness", required_argument, 0, 't'},
        {"mirror",no_argument, 0, 'm'},
        {"axis", required_argument, 0, 'a'},
        {"left_up", required_argument, 0, 'u'},
        {"right_down", required_argument, 0, 'd'},
        {"pentagram", no_argument, 0, 'p'},
        {"center", required_argument, 0, 'c'},
        {"radius", required_argument, 0, 'r'},
        {"help", no_argument, 0, 'h'},
        {"input", required_argument, 0, 'i'},
        {"output", required_argument, 0, 'o'},
        {"info", no_argument, 0, 'z'}
    };
    while ((opt=getopt_long(argc, argv, "ls:e:k:t:ma:u:d:pc:r:hi:o:z",
long_option, &option_index))!=-1){
        switch(opt){
            case 'h':
                printf("Course work for option 4.20, created by Yana
Konasova. Вызов флага --line рисует линию. Используйте флаг --start для
задачи координат для начала линии и флаг --end для задачи координат конца
линии, а также флаг --color для задачи цвета и флаг --thickness для
задачи толщины линии. Вызов флага --mirror отображает заданную область.
Используйте флаг --axis для выбора оси относительно которой отображать, а
также флаг --left_up для задачи левого верхнего угла области и флаг --
right_down для задачи правого нижнего угла области. Вызов флага --
pentagram рисует пентаграмму в круге. Используйте флаг --center для
задачи координат центра круга и флаг -radius для задачи радиуса круга, а
также флаг --thickness для задачи толщины линий и флаг --color для задачи
цвета.\n ");
                return 0;
                break;
            case 'l':
                will_draw_line=1;
                break;
            case 'm':
                will_mirror=1;
                break;
            case 'p':
                will_draw_pentagram=1;
                break;
            case 's':
                if(sscanf(optarg, "%d.%d", &start_x, &start_y)!=2){
                    printf("Задан некорректный аргумент\n");
                    exit(43);
```

```
                }
                break;
            case 'e':
                if(sscanf(optarg, "%d.%d", &end_x, &end_y)!=2){
                    printf("Задан некорректный аргумент\n");
                    exit(43);
                }
                break;
            case 'k':
                if(sscanf(optarg, "%d.%d.%d", &color.red, &color.green,
&color.blue)!=3){
                    printf("Задан некорректный аргумент\n");
                    exit(43);
                }
                if
(!(color.red>=0&&color.red<=255&&color.blue>=0&&color.blue<=255&&color.gr
een>=0&&color.green<=255)){
                    printf("Задан некорректный цвет.\n");
                    exit(40);
                }

                break;
            case 't':
                if(sscanf(optarg, "%d", &thickness_of_line)!=1){
                    printf("Задан некорректный аргумент\n");
                    exit(43);
                }
                if (thickness_of_line<=0){
                    printf("Задана некорректная толщина линии.\n");
                    exit(41);
                }
                break;
            case 'a':
                if(sscanf(optarg, "%c", &wich_mirror)!=1){
                    printf("Задан некорректный аргумент\n");
                    exit(43);
                }
                if (!((wich_mirror=='x')||(wich_mirror=='y'))){
                    printf("Задана некорректная ось отображения.\n");
                    exit(42);
                }
                break;
            case 'u':
                if(sscanf(optarg, "%d.%d", &up_x, &up_y)!=2){
                    printf("Задан некорректный аргумент\n");
                    exit(43);
                }
                break;
            case 'd':
                if(sscanf(optarg, "%d.%d", &right_x, &right_y)!=2){
                    printf("Задан некорректный аргумент\n");
                    exit(43);
                }
                break;
            case 'c':
                if(sscanf(optarg, "%d.%d", &center.x, &center.y)!=2){
                    printf("Задан некорректный аргумент\n");
```

```
                    exit(43);
                }
                break;
            case 'r':
                if(sscanf(optarg, "%d", &radius)!=1){
                    printf("Задан некорректный аргумент\n");
                    exit(43);
                }
                if (radius<=0){
                    printf("Задан некорректный радиус.\n");
                    exit(43);
                }
                break;
            case 'i':
                png_name_of_file=optarg;
                break;
            case 'o':
                png_out_name=optarg;
                break;
            case 'z':
                print_info=1;
                break;



        }
    }
    if (optind < argc) {
        png_name_of_file = argv[optind];
    }
    if (png_out_name!=NULL&& png_name_of_file!=NULL&&
strcmp(png_name_of_file, png_out_name) == 0) {

        printf("Ошибка: Имена входного и выходного файлов не могут
совпадать.\n");
        exit(44);
    }
    struct Png *png;
    png = read_png(png_name_of_file);

    if (print_info==1){
        printf("Высота изображения: %d\nШирина изображения: %d\nГлубина
цвета изображения: %d\n", png->height, png->width, png->bit_depth);
        switch (png->color_type){
            case PNG_COLOR_TYPE_RGB:
                printf("Тип цвета: RGB\n");
                break;
            case PNG_COLOR_TYPE_RGBA:
                printf("Тип цвета: RGBA\n");
                break;
        }
        return 0;
    }

    if (will_draw_line==1){
        draw_line(png, start_x, start_y, end_x, end_y, color,
thickness_of_line);
```

```
    }

    if (will_mirror==1){
        mirror_region(png, wich_mirror, up_x, up_y, right_x, right_y);
    }

    if (will_draw_pentagram==1){
        draw_pentagram(png, center, radius, thickness_of_line, color);
    }




    save_png(png_out_name, png);
    for (int y = 0; y < png->height; y++) {
        free(png->row_pointers[y]);
    }
    free(png->row_pointers);


    return 0;
}
```