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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Обработка изображения в формате PNG

Студент гр. 3343 _____ Жучков О.Д.

Преподаватель _____ Государкин Я.С.

Санкт-Петербург

2024

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студент: Жучков Олег

Группа: 3343

Тема: Обработка изображения в формате PNG

Условия задания (Вариант 4.21):

Программа должна иметь следующие функции по обработке изображений:

1. Рисование прямоугольника. Флаг для выполнения данной операции: `--rect`. Он определяется:
   - Координатами левого верхнего угла. Флаг `--left_up`, значение задаётся в формате `left.up`, где left – координата по x, up – координата по y.
   - Координатами правого нижнего угла. Флаг `--right_down`, значение задаётся в формате `right.down`, где right – координата по x, down – координата по y.
   - Толщиной линий. Флаг `--thickness`. На вход принимает число больше 0.
   - Цветом линий. Флаг `--color` (цвет задаётся строкой `rrr.ggg.bbb`, где rrr/ggg/bbb – числа, задающие цветовую компоненту. пример `--color 255.0.0` задаёт красный цвет).
   - Прямоугольник может быть залит или нет. Флаг `--fill`. Работает как бинарное значение: флага нет – false , флаг есть – true.
   - Цветом которым он залит, если пользователем выбран залитый. Флаг `--fill_color` (работает аналогично флагу `--color`).
2. Рисование правильного шестиугольника. Флаг для выполнения данной операции: `--hexagon`. Шестиугольник определяется:
   - Координатами его центра и радиусом в который он вписан. 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 линий. Флаг `--thickness`. На вход принимает число больше 0.
- Цветом линий. Флаг `--color` (цвет задаётся строкой `rrr.ggg.bbb`, где rrr/ggg/bbb – числа, задающие цветовую компоненту. пример `--color 255.0.0` задаёт красный цвет).
- Шестиугольник может быть залит или нет. Флаг `--fill`. Работает как бинарное значение: флага нет – false , флаг есть – true.
- Цветом которым залит шестиугольник, если пользователем выбран залитый. Флаг `--fill_color` (работает аналогично флагу `--color`).

3. Копирование заданной области. Флаг для выполнения данной операции: `--copy`. Функционал определяется:
   - Координатами левого верхнего угла области-источника. Флаг `--left_up`, значение задаётся в формате `left.up`, где left – координата по x, up – координата по y.
   - Координатами правого нижнего угла области-источника. Флаг `--right_down`, значение задаётся в формате `right.down`, где right – координата по x, down – координата по y.
   - Координатами левого верхнего угла области-назначения. Флаг `--dest_left_up`, значение задаётся в формате `left.up`, где left – координата по x, up – координата по y.

Дата выдачи задания: 18.03.2024

Дата сдачи реферата: 23.05.2024

Дата защиты реферата: 23.05.2024

АННОТАЦИЯ

В процессе работы создан проект на языке программирования C, обрабатывающий PNG изображения. Для работы с изображениями в программе используется библиотека libpng. В программе реализованы следующие задачи: рисование прямоугольника, рисование правильного шестиугольника, копирование области. Программа имеет интерфейс командной строки (используется библиотека getopt). Функции программы разделены по нескольким файлам, для сборки проекта используется утилита make.

ВВЕДЕНИЕ

Цель работы заключается в изучении структуры PNG файлов и освоении работы с ними. Необходимо разработать программу с функциями обработки изображения, таких как рисование геометрических фигур, и для взаимодействия с пользователем необходимо реализовать интерфейс командной строки. Программа должна обрабатывать возможные ошибки, такие как некорректный пользовательский ввод.

1. ОПИСАНИЕ СТРУКТУР ДАННЫХ

1.1. Структура png_t

Данная структура предназначена для хранения информации об изображении и его содержании. Используется при чтении и записи PNG файла, при обработке изображения.

1.2. Структура rgb_t

Данная структура хранит цвет в формате RGB (значения красного, зеленого, красного цветов). Используеся для рисования на изображении, копирования пикселей.

1.3. Структура params_t

Структура содержит обработанные аргументы, введённые пользователем с помощью интерфейса командной строки.

2. ОПИСАНИЕ СТРУКТУРЫ ПРОГРАММЫ

2.1. params.c

- *void print_help()* – вывод справки.
- *void init_params(params_t* params)* – инициализирует поля params_t.
- *void raise_arg_error()* – выводит сообщение об ошибке при некорректном вводе.
- *void parse_coords(char* arg, int* x, int* y)* – обработка введённых координат формата “x.y”.
- *void parse_rgb(char* arg, rgb_t* rgb)* – обработка введённого цвета формата “r.g.b”.
- *void parse_params(params_t* optparams, int argc, char** argv)* – обработка пользовательского ввода через интерфейс командной строки.

2.2. read_write_png.c

- *void read_png(char* file_name, png_t* image)* – открытие и считывание PNG файла.
- *void write_png(char* file_name, png_t* image)* – запись обработанного изображения в PNG файл.

2.3. png_edit.c

- *rgb_t int_to_rgb(int r, int g, int b)* – создание структуры rgb_t из 3 значений int.
- *rgb_t png_byte_to_rgb(png_byte r, png_byte g, png_byte b* – создание структуры rgb_t из 3 значений png_byte.
- *bool check_coords(png_t* image, int x, int y)* – проверка координат на вхождение в размеры изображения.
- *void set_pixel(png_t* image, int x, int y, rgb_t rgb)* – закрашивание данного пикселя данным цветом.
- *rgb_t get_pixel(png_t* image, int x, int y)* – получить цвет данного пикселя и вернуть в виде rgb_t.

- *void copy_pixel_to(png_t* image, int x0, int y0, int x1, int y1)* – копирование значение цвета одного пикселя в другой.

2.4. png_draw

- *void draw_line(png_t* image, int x0, int y0, int x1, int y1, rgb_t rgb, int thickness)* – рисование линии заданной толщины с использованием алгоритма Брезенхэма.
- *void fill_circle(png_t* image, int x0, int y0, rgb_t rgb, int radius)* – рисование закрашенного круга (используется для толщины).
- *void draw_circle(png_t* image, int x0, int y0, rgb_t rgb, int radius)* – рисование окружности с использованием алгоритма Брезенхэма.
- *void draw_rectangle(png_t* image, int x0, int y0, int x1, int y1, rgb_t rgb, int thickness, bool fill, rgb_t fillrgb)* – рисование прямоугольника.
- *void fill_rectangle(png_t* image, int x0, int y0, int x1, int y1, rgb_t rgb)* – закрашивание прямоугольника.
- *void draw_hexagon(png_t* image, int x0, int y0, rgb_t rgb, int radius, int thickness, bool fill, rgb_t rgbfill)* – рисование шестиугольника.
- *void fill_hexagon(png_t* image, int x0, int y0, rgb_t rgb, int radius)* – закрашивание шестиугольника.
- *void copy_area_to(png_t* image, int x0, int y0, int x1, int y1, int x2, int y2)* – копирование выделенной области.

2.5. main.c

В функции main() вызывется функция обработки введённых аргументов и вызываются функции открытия, обработки, записи изображения.

ТЕСТИРОВАНИЕ

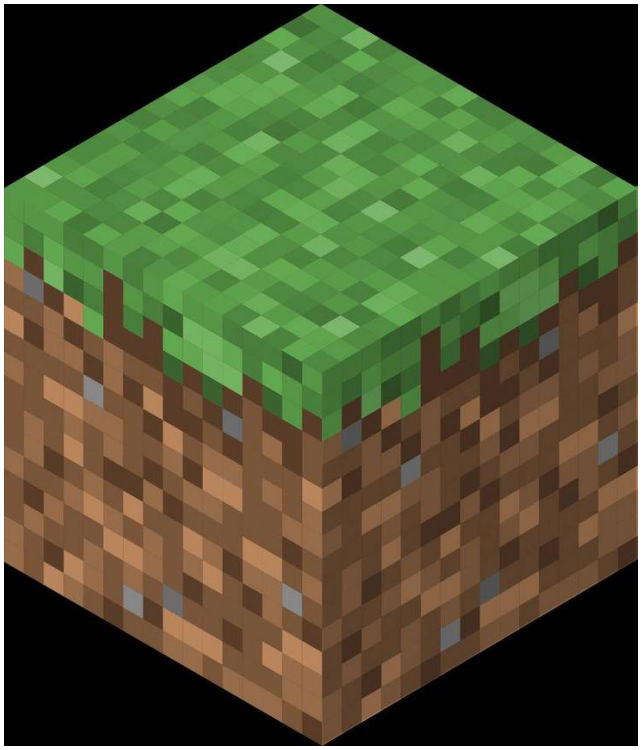
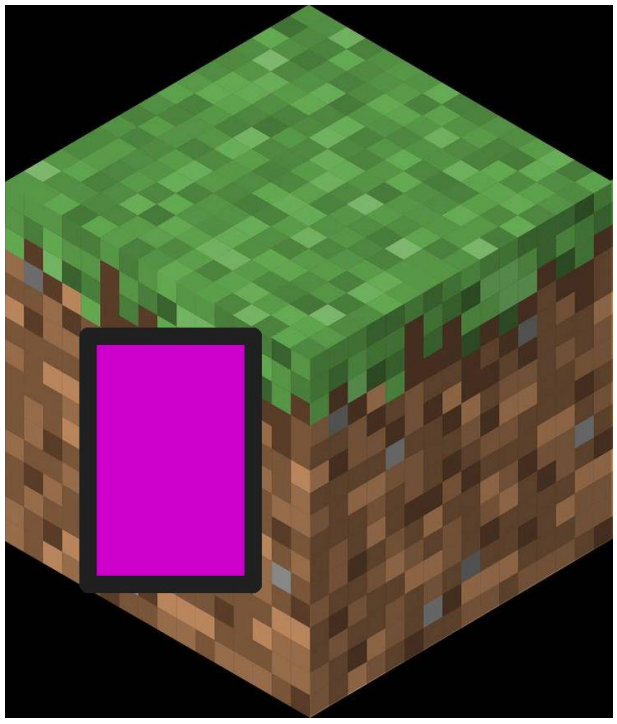


Рисунок 1 – изображение для тестирования

1. Задание *rectangle*:

   Аргументы запуска: ./cw --rect --left_up 100.400 --right_down 300.700 --



color 32.32.32 --thickness 20 --fill --fill_color 204.0.204 in.png

Рисунок 2 – результат работы для задания *rectangle*

2. Задание *hexagon*:

   Аргументы запуска: ./cw --hexagon --center 368.200 --radius 100 --color 255.51.51 --fill --fill_color 255.128.0 --thickness 10 in.png

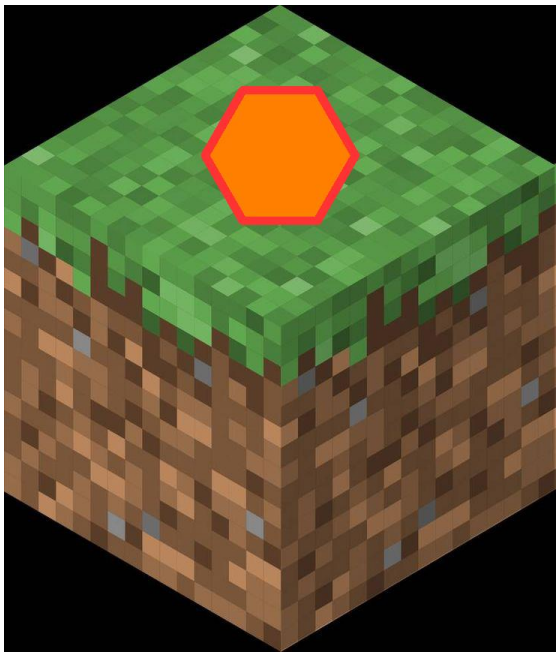


Рисунок 3 – результат работы для задания *hexagon*

3. Задание *copy*:

   Аргументы запуска: ./cw --copy --left_up 100.100 --right_down 640.300 --dest_left_up 100.500 in.png

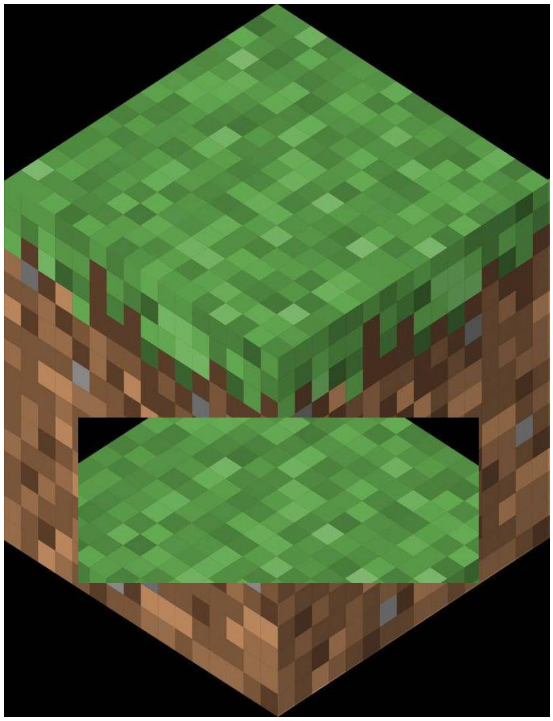


Рисунок 4 – результат работы для задания *ornament*

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы создана программа на языке программирования C, осуществляющая обработку PNG изображения с помощью библиотеки libpng. Программа может выполнять такие задачи, как: рисование прямоугольника, шестиугольника, копирование области. Выбор задачи и ввод аргументов производится через интерфейс командной строки. Проект разделен на несколько файлов и компилируется с помощью make.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: params.h

```
#ifndef param
#define param
#include <getopt.h>
#include <structs.h>
void init_params(params_t* params);
void raise_arg_error();
void parse_coords(char* arg, int* x, int* y);
void parse_rgb(char* arg, rgb_t* rgb);
void parse_params(params_t* optparams, int argc, char** argv);
#endif
```

Название файла: png_draw.h

```
#ifndef draw
#define draw
#include <structs.h>
#include <png.h>
#include <stdbool.h>
void draw_line(png_t* image, int x0, int y0, int x1, int y1, rgb_t rgb,
int thickness);
void draw_line_symmetrical_point(png_t* image, int x0, int y0, int x1,
int y1, int x2, int y2, rgb_t rgb, int thickness);
void fill_circle(png_t* image, int x0, int y0, rgb_t rgb, int radius);
void draw_circle(png_t* image, int x0, int y0, rgb_t rgb, int radius);
void draw_rectangle(png_t* image, int x0, int y0, int x1, int y1, rgb_t
rgb, int thickness, bool fill, rgb_t fillrgb);
void fill_rectangle(png_t* image, int x0, int y0, int x1, int y1, rgb_t
rgb);
void draw_hexagon(png_t* image, int x0, int y0, rgb_t rgb, int radius,
int thickness, bool fill, rgb_t rgbfill);
void fill_hexagon(png_t* image, int x0, int y0, rgb_t rgb, int radius);
void copy_area_to(png_t* image, int x0, int y0, int x1, int y1, int x2,
int y2);
#endif
```

Название файла: png_edit.h

```
#ifndef pngedit
#define pngedit
#include <png.h>
#include <structs.h>
#include <stdbool.h>

rgb_t int_to_rgb(int r, int g, int b);
rgb_t png_byte_to_rgb(png_byte r, png_byte g, png_byte b);
bool check_coords(png_t* image, int x, int y);
void set_pixel(png_t* image, int x, int y, rgb_t rgb);
rgb_t get_pixel(png_t* image,int x, int y);
void copy_pixel_to(png_t* image, int x0, int y0, int x1, int y1);
#endif
```

Название файла: read_write_png.h

```
#ifndef read_write
#define read_write
#include <png.h>
#include <structs.h>

void read_png(char* file_name, png_t* image);
void write_png(char* file_name, png_t* image);
#endif
```

Название файла: structs.h

```
#ifndef structs
#define structs

#include <png.h>
#include <stdbool.h>

typedef struct {
    png_byte r, g, b;
} rgb_t;

typedef struct {
    png_structp png_ptr;
    png_infop info_ptr;
    png_byte bit_depth, color_type;
    int number_passes;
    int width, height;
    png_bytepp rows;
} png_t;

typedef struct{
    char* input;
    char* output;
    bool info;
    bool help;

    bool rect;
    int left_up_x;
    int left_up_y;
    int right_down_x;
    int right_down_y;
    int thickness;
    rgb_t color;
    bool fill;
    rgb_t fill_color;
    bool color_input;
    bool fcolor_input;

    bool hexagon;
    int center_x;
    int center_y;
    int radius;

    bool copy;
    int dest_left_up_x;
```

```
    int dest_left_up_y;
} params_t;
#endif
```

Название файла: main.c

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

#include "structs.h"
#include "read_write_png.h"
#include "png_draw.h"
#include "png_edit.h"
#include "params.h"


void print_info(png_t* image){
    printf("Image info:\nHeight: %d; Width: %d; Color type: %d; Bit
depth: %d\n", image->height, image->width, image->color_type, image-
>bit_depth);
}

int main(int argc, char** argv){
    params_t params;
    init_params(&params);
    parse_params(&params, argc, argv);
    png_t image;
    if (strcmp(params.input,params.output)==0){
        puts("Input and output can't be the same file!");
        exit(49);
    }
    read_png(params.input, &image);
    int cnt = 0;
    if (params.rect) cnt++;
    if (params.hexagon) cnt++;
    if (params.copy) cnt++;
    if (cnt > 1){
        raise_arg_error();
    }
    if (params.rect){
        if (params.left_up_x != -1 && params.left_up_y != -1 &&
        params.right_down_x != -1 && params.right_down_y != -1 &&
        params.color_input &&
        params.thickness != -1)
            if (params.fill && (!params.fcolor_input))
                raise_arg_error();
            else
                draw_rectangle(&image, params.left_up_x, params.left_up_y,
params.right_down_x, params.right_down_y, params.color, params.thickness,
params.fill, params.fill_color);
        else
            raise_arg_error();
    }
    if (params.hexagon){
        if (params.center_x != -1 && params.center_y != -1 &&
params.radius != -1 &&
```

```
        params.color_input &&
        params.thickness != -1)
            if (params.fill && (!params.fcolor_input))
                raise_arg_error();
            else
                draw_hexagon(&image, params.center_x, params.center_y,
params.color, params.radius, params.thickness, params.fill,
params.fill_color);
        else
            raise_arg_error();
    }
    if (params.copy){
        if (params.left_up_x!= -1 && params.left_up_y != -1 &&
params.right_down_x != -1 &&
        params.right_down_y != -1 && params.dest_left_up_x != -1 &&
params.dest_left_up_y != -1)
            copy_area_to(&image, params.left_up_x, params.left_up_y,
params.right_down_x, params.right_down_y, params.dest_left_up_x,
params.dest_left_up_y);
        else
            raise_arg_error();
    }
    if (params.info) print_info(&image);
    write_png(params.output, &image);
}
```

Название файла: params.c

```
#include <params.h>
#include <getopt.h>
#include <stdlib.h>
#include <string.h>
#include <png_edit.h>

void print_help(){
    printf("Course work for option 4.21, created by Zhuchkov Oleg.\n"
    "-h --help: Display parameters information\n"
    "-i --input: Name of the input png. REQUIRED (if not using --input
put the filename at the end of command line)\n"
    "-o --output: Name for the output png. out.png by default\n"
    "--info: Display png information\n"
    "--rect: Draw a rectangle\n"
    "--hexagon: Draw a hexagon\n"
    "--copy: Copy a rectangular area\n"
    "--left_up: First corner of an area (for rectangle and copy)\n"
    "--right_down: Second corner of an area\n"
    "--center: Center of the hexagon\n"
    "--radius: Radius of the hexagon\n"
    "--thickness: Thickness of drawn figures\n"
    "--color: Color of drawn figures\n"
    "--fill: Fill drawn shape\n"
    "--fill_color: Color to fill drawn shape\n"
    "Coords format: x.y Color format: r.g.b\n");
}

void init_params(params_t* params){
    params->input = NULL;
    params->output = "out.png";
```

```
    params->info = false;
    params->help = false;
    params->rect = false;
    params->left_up_x = -1;
    params->left_up_y = -1;
    params->right_down_x = -1;
    params->right_down_y = -1;
    params->thickness = -1;
    params->color.r = -1;
    params->color.g = -1;
    params->color.b = -1;
    params->fill = false;
    params->fill_color.r = -1;
    params->fill_color.g = -1;
    params->fill_color.b = -1;
    params->hexagon = false;
    params->center_x = -1;
    params->center_y = -1;
    params->radius = -1;
    params->copy=false;
    params->dest_left_up_x=-1;
    params->dest_left_up_y=-1;
    params->color_input=false;
    params->fcolor_input=false;
}

void raise_arg_error(){
    puts("Incorrect or missing argument(s)!");
    exit(42);
}

void parse_coords(char* arg, int* x, int* y){
    char xx[30], yy[30];
    int i=0, j=0, l=0;
    while (arg[i] != '\0'){
        if (arg[i]== '.'){
            xx[l] = '\0';
            if (j==1) raise_arg_error();
            j++;
            i++;
            l = 0;
            continue;
        }
        if ((arg[i]>'9'||arg[i]<'0')&&arg[i]!='-') raise_arg_error();
        if (j==0){
            xx[l] = arg[i];
        }
        else{
            yy[l] = arg[i];
        }
        i++;
        l++;
    }
    yy[l] = '\0';
    if(strlen(xx) == 0 || strlen(yy) == 0) raise_arg_error();
    *x = strtol(xx,NULL,10);
    *y = strtol(yy,NULL,10);
}
```

```
void parse_rgb(char* arg, rgb_t* rgb){
    char r[30], g[30], b[30];
    int i=0, j=0, l=0;
    while (arg[i] != '\0'){
        if (arg[i]== '.'){
            if (j==2) raise_arg_error();
            if (j==1){
                g[l] = '\0';
            }
            if (j==0){
                r[l] = '\0';
            }
            j++;
            i++;
            l=0;
            continue;
        }
        if (arg[i]>'9'||arg[i]<'0') raise_arg_error();
        if (j==0){
            r[l] = arg[i];
        }
        if (j==1){
            g[l] = arg[i];
        }
        else
            b[l] = arg[i];
        i++;
        l++;
    }
    b[l] = '\0';
    if (strlen(r) == 0 || strlen(g) == 0 || strlen(b) == 0)
raise_arg_error();
    int rr = strtol(r,NULL,10);
    int gg = strtol(g,NULL,10);
    int bb = strtol(b,NULL,10);
    if (rr > 255 || gg > 255 || bb > 255) raise_arg_error();
    *rgb = int_to_rgb(rr,gg,bb);
}

void parse_params(params_t* optparams, int argc, char** argv){
    opterr = 0;
    static struct option options[] = {
        {"input", 1, NULL, 'i'},
        {"output", 1, NULL, 'o'},
        {"info", 0, NULL, 400},
        {"help", 0, NULL, 'h'},
        {"rect", 0, NULL, 401},
        {"left_up", 1, NULL, 402},
        {"right_down", 1, NULL, 403},
        {"thickness", 1, NULL, 404},
        {"color", 1, NULL, 405},
        {"fill", 0, NULL, 406},
        {"fill_color", 1, NULL, 407},
        {"hexagon", 0, NULL, 408},
        {"center", 1, NULL, 409},
        {"radius", 1, NULL, 410},
        {"copy", 0, NULL, 411},
```

```
        {"dest_left_up", 1, NULL, 412},
        {0,0,0,0}
    };
    int c = 1;
    int arg;
    while(c != -1){
        c = getopt_long(argc, argv, "hi:o:",options,NULL);
        if (c == -1) break;
        switch (c){
            case 'h':{
                optparams->help = true;
                break;}
            case 'i':{
                optparams->input = optarg;
                break;
                }
            case 'o':{
                optparams->output = optarg;
                break;
                }
            case 400:{
                optparams->info = true;
                break;
                }
            case 401:{
                optparams->rect = true;
                break;
                }
            case 402:{
                parse_coords(optarg, &(optparams->left_up_x),
&(optparams->left_up_y));
                break;
            }
            case 403:{
                parse_coords(optarg, &(optparams->right_down_x),
&(optparams->right_down_y));
                break;
            }
            case 404:{
                arg = strtol(optarg, NULL, 10);
                if (arg <= 0) raise_arg_error();
                optparams->thickness = arg;
                break;
            }
            case 405:{
                parse_rgb(optarg, &(optparams->color));
                optparams->color_input=true;
                break;
            }
            case 406:{
                optparams->fill = true;
                break;
            }
            case 407:{
                parse_rgb(optarg, &(optparams->fill_color));
                optparams->fcolor_input=true;
                break;
            }
```

```
            case 408:{
                optparams->hexagon = true;
                break;
            }
            case 409:{
                parse_coords(optarg, &(optparams->center_x), &(optparams-
>center_y));
                break;
                }
            case 410:{
                arg = strtol(optarg, NULL, 10);
                if (arg <= 0) raise_arg_error();
                optparams->radius = arg;
                break;
            }
            case 411:{
                optparams->copy = true;
                break;
            }
            case 412:{
                parse_coords(optarg, &(optparams->dest_left_up_x),
&(optparams->dest_left_up_y));
                break;
            }
            case '?':{
                raise_arg_error();
                break;
            }
            default:{
                raise_arg_error();
                break;
            }
        }
    }
    if (optparams->help){
        print_help();
        exit(0);
    }
    if (optparams->input == NULL && optind == argc - 1)
    {
        optparams->input = calloc(strlen(argv[argc - 1]) + 1,
sizeof(char));
        strncpy(optparams->input, argv[argc - 1], strlen(argv[argc - 1])
+ 1);
    }
    if (optparams->input == NULL){
        puts("No input file!");
        exit(49);
    };

}
```

Название файла: png_draw.c

```
#include <png.h>
#include <stdbool.h>
#include <math.h>
#include <stdlib.h>
```

```
#include <png_draw.h>
#include <png_edit.h>
#include <structs.h>


void draw_line(png_t* image, int x0, int y0, int x1, int y1, rgb_t rgb,
int thickness){
    int deltax = abs(x1 - x0);
    int deltay = abs(y1 - y0);
    int temp;
    bool swap = false;
    if(deltay > deltax){
        temp = x0;
        x0 = y0;
        y0 = temp;
        temp = x1;
        x1 = y1;
        y1 = temp;
        temp = deltax;
        deltax = deltay;
        deltay = temp;
        swap = true;
    }
    if (y0 > y1){
        temp = x0;
        x0 = x1;
        x1 = temp;
        temp = y0;
        y0 = y1;
        y1 = temp;
    }
    int error = 0;
    int deltaerr = (deltay + 1);
    int y = y0;
    int diry = 1;
    if (x0 > x1)
        for (int x = x0; x>=x1; x--){
            if (thickness==1)
                if (swap)
                    set_pixel(image,y,x,rgb);
                else
                    set_pixel(image,x,y,rgb);
            else
                if (swap)
                    fill_circle(image,y,x,rgb,(thickness+1)/2);
                else
                    fill_circle(image,x,y,rgb,(thickness+1)/2);
            error = error + deltaerr;
            if (error >= (deltax + 1))
            {
                y = y + diry;
                error = error - (deltax + 1);
            }
        }
    else
        for (int x = x0; x<=x1; x++){
            if (thickness==1)
                if (swap)
```

```
                    set_pixel(image,y,x,rgb);
                else
                    set_pixel(image,x,y,rgb);
            else
                if (swap)
                    fill_circle(image,y,x,rgb,(thickness+1)/2);
                else
                    fill_circle(image,x,y,rgb,(thickness+1)/2);
            error = error + deltaerr;
            if (error >= (deltax + 1))
            {
                y = y + diry;
                error = error - (deltax + 1);
            }
        }
}

void draw_line_symmetrical_point(png_t* image, int x0, int y0, int x1,
int y1, int x2, int y2, rgb_t rgb, int thickness){
    int deltax = abs(x1 - x0);
    int deltay = abs(y1 - y0);
    int temp;
    bool swap = false;
    if(deltay > deltax){
        temp = x0;
        x0 = y0;
        y0 = temp;
        temp = x1;
        x1 = y1;
        y1 = temp;
        temp = deltax;
        deltax = deltay;
        deltay = temp;
        swap = true;
    }
    if (y0 > y1){
        temp = x0;
        x0 = x1;
        x1 = temp;
        temp = y0;
        y0 = y1;
        y1 = temp;
    }
    int error = 0;
    int deltaerr = (deltay + 1);
    int y = y0;
    int diry = 1;
    int ax, ay;
    if (x0 > x1)
        for (int x = x0; x>=x1; x--){
            if (swap){
                    ax = y;
                    ay = x;
                }
                else{
                    ax = x;
                    ay = y;
                }
```

```
            if (thickness==1){
                set_pixel(image,ax,ay,rgb);
                set_pixel(image,x0*2 - ax,ay,rgb);
                set_pixel(image,ax,y0*2 - ay,rgb);
                set_pixel(image,x0*2 - ax,y0*2 - ay,rgb);
            }
            else{
                fill_circle(image,ax,ay,rgb,thickness/2);
                fill_circle(image,x0*2 - ax,ay,rgb,thickness/2);
                fill_circle(image,ax,y0*2 - ay,rgb,thickness/2);
                fill_circle(image,x0*2 - ax,y0*2 - ay,rgb, thickness/2);
            }
            error = error + deltaerr;
            if (error >= (deltax + 1))
            {
                y = y + diry;
                error = error - (deltax + 1);
            }

        }
    else
        for (int x = x0; x<=x1; x++){
            if (swap){
                    ax = y;
                    ay = x;
                }
                else{
                    ax = x;
                    ay = y;
                }
            if (thickness==1){
                set_pixel(image,ax,ay,rgb);
                set_pixel(image,x0*2 - ax,ay,rgb);
                set_pixel(image,ax,y0*2 - ay,rgb);
                set_pixel(image,x0*2 - ax,y0*2 - ay,rgb);
            }
            else{
                fill_circle(image,ax,ay,rgb,thickness/2);
                fill_circle(image,x0*2 - ax,ay,rgb,thickness/2);
                fill_circle(image,ax,y0*2 - ay,rgb,thickness/2);
                fill_circle(image,x0*2 - ax,y0*2 - ay,rgb, thickness/2);
            }
            error = error + deltaerr;
            if (error >= (deltax + 1))
            {
                y = y + diry;
                error = error - (deltax + 1);
            }
        }
}

void fill_circle(png_t* image, int x0, int y0, rgb_t rgb, int radius){
    if (x0-radius > image->width) return;
    if (y0-radius > image->height) return;
    if (x0+radius < 0) return;
    if (y0+radius < 0) return;
    draw_circle(image, x0, y0, rgb, radius);
    for(int y=-radius; y<=radius; y++){
```

```
     if (y0+y < 0) continue;
     if (y0+y > image->height) break;
        for(int x=-radius; x<=radius; x++){
            if(x0+x < 0) continue;
            if (x0+x > image->width) break;
            if(x*x+y*y <= radius*radius)
                set_pixel(image, x0+x, y0+y, rgb);
                }
    }
}

void draw_circle(png_t* image, int x0, int y0, rgb_t rgb, int radius){
    int x = 0;
    int y = radius;
    int delta = 3 - 2 * radius;
    while (y >= x){
        set_pixel(image,x0 + x, y0 + y,rgb);
        set_pixel(image,x0 + x, y0 - y,rgb);
        set_pixel(image,x0 - x, y0 + y,rgb);
        set_pixel(image,x0 - x, y0 - y,rgb);
        set_pixel(image,x0 + y, y0 + x,rgb);
        set_pixel(image,x0 + y, y0 - x,rgb);
        set_pixel(image,x0 - y, y0 + x,rgb);
        set_pixel(image,x0 - y, y0 - x,rgb);
        if (delta < 0)
            delta = delta + 4*x++ + 6;
        else
            delta = delta + 4*(x++ - y--) + 10;
    }
}

void draw_rectangle(png_t* image, int x0, int y0, int x1, int y1, rgb_t
rgb, int thickness, bool fill, rgb_t fillrgb){
    if (fill) fill_rectangle(image, x0, y0, x1, y1, fillrgb);
    draw_line(image, x0, y0, x1, y0, rgb, thickness);
    draw_line(image, x0, y1, x1, y1, rgb, thickness);
    draw_line(image, x0, y0, x0, y1, rgb, thickness);
    draw_line(image, x1, y0, x1, y1, rgb, thickness);
}

void fill_rectangle(png_t* image, int x0, int y0, int x1, int y1, rgb_t
rgb){
    int temp;
    if (x0 > x1){
        temp = x0;
        x0 = x1;
        x1 = temp;
    }
    if (y0 > y1){
        temp = y0;
        y0 = y1;
        y1 = temp;
    }
    if (x0 >= image->width) return;
    if (y0 >= image->height) return;
    if (x1 > image->width) x1 = image->width;
    if (y1 > image->height) y1 = image->height;
    if (x0 < 0) x0 = 0;
```

```
    if (y0 < 0) y0 = 0;
    for (int x = x0; x<= x1; x++)
        for (int y = y0; y<=y1; y++)
            set_pixel(image, x, y, rgb);
}

void draw_hexagon(png_t* image, int x0, int y0, rgb_t rgb, int radius,
int thickness, bool fill, rgb_t rgbfill){
    int x1, y1;
    x1 = x0 - radius / 2;
    y1 = y0 - sqrt(3) / 2 * radius;
    if (fill) fill_hexagon(image, x0, y0, rgbfill, radius);
    draw_line(image, x1,y1, x0-radius,y0, rgb, thickness);
    draw_line(image, x1,y1, x0*2-x1,y1, rgb, thickness);
    draw_line(image, x0*2-x1,y1, x0+radius,y0, rgb, thickness);
    draw_line(image, x0+radius,y0, x0*2-x1,y0*2-y1, rgb, thickness);
    draw_line(image, x0*2 - x1,y0*2-y1, x1,y0*2-y1, rgb, thickness);
    draw_line(image, x1,y0*2-y1, x0-radius, y0, rgb, thickness);
}

void fill_hexagon(png_t* image, int x0, int y0, rgb_t rgb, int radius){
    int x1, y1, x2, y2;
    x1 = x0 - radius;
    y1 = y0;
    x2 = x0 - radius / 2;
    y2 = y0 - sqrt(3) / 2 * radius;
    fill_rectangle(image, x2, y2, x0 * 2 - x2, y0 * 2 - y2, rgb);
    int deltax = abs(x2- x1);
    int deltay = abs(y2 - y1);
    int error = 0;
    int deltaerr = (deltax + 1);
    int x = x2;
    int dirx = -1;
    for (int y = y2; y<=y1; y++){
        for (int yy = y; yy<=y1; yy++){
            set_pixel(image,x,yy,rgb);
            set_pixel(image,x0*2 - x,yy,rgb);
            set_pixel(image,x,y0*2 - yy,rgb);
            set_pixel(image,x0*2 - x,y0*2 - yy,rgb);
        }
        error = error + deltaerr;
        if (error >= (deltay + 1))
        {
            x = x + dirx;
            error = error - (deltay + 1);
        }
    }
}

void copy_area_to(png_t* image, int x0, int y0, int x1, int y1, int x2,
int y2){
    int temp;
    if (x0 > x1){
        temp = x0;
        x0 = x1;
        x1 = temp;
    }
    if (y0 > y1){
```

```
        temp = y0;
        y0 = y1;
        y1 = temp;
    }
    for (int x = 0; x < x1 - x0; x++)
        for (int y = 0; y < y1 - y0; y++)
            copy_pixel_to(image, x0+x, y0+y, x2 + x, y2 + y);
}
```

Название файла: png_edit.c

```
#include <png.h>
#include <png_edit.h>
#include <structs.h>

rgb_t int_to_rgb(int r, int g, int b){
    rgb_t color;
    color.r = (png_byte)r;
    color.g = (png_byte)g;
    color.b = (png_byte)b;
    return color;
}

rgb_t png_byte_to_rgb(png_byte r, png_byte g, png_byte b){
    rgb_t color;
    color.r = r;
    color.g = g;
    color.b = b;
    return color;
}

bool check_coords(png_t* image, int x, int y){
    return ((x < 0 || x >= image->width) || (y < 0 || y >= image-
>height));
}

void set_pixel(png_t* image, int x, int y, rgb_t rgb){
    if (check_coords(image, x, y)) return;
    png_bytep pixel = (png_bytep)(image->rows)[y]+x*3;
    pixel[0] = rgb.r;
    pixel[1] = rgb.g;
    pixel[2] = rgb.b;
}

rgb_t get_pixel(png_t* image,int x, int y){
    png_byte r,g,b;
    r = ((png_bytep)(image->rows[y]))[x*3];
    g = ((png_bytep)(image->rows[y]))[x*3+1];
    b = ((png_bytep)(image->rows[y]))[x*3+2];
    return png_byte_to_rgb(r,g,b);
}

void copy_pixel_to(png_t* image, int x0, int y0, int x1, int y1){
    if (check_coords(image, x0, y0) || check_coords(image, x1, y1))
return;
    rgb_t color;
    color = get_pixel(image, x0, y0);
    set_pixel(image,x1,y1,color);
```

```
}
```

Название файла: read_write_png.c

```
#include <read_write_png.h>
#include <stdlib.h>
#include <stdio.h>
#include <png.h>

void read_png(char* file_name, png_t* image) {
    FILE* fp = fopen(file_name, "rb");
    if (fp == NULL){
        puts("Cannot open file!");
        exit(40);
    }
    png_byte header[8];
    fread(header, sizeof(png_byte), 8, fp);
    if (png_sig_cmp(header, 0, 8)){
        puts("Wrong file signature! Is it a png?");
        exit(40);
    }
    image->png_ptr = png_create_read_struct(PNG_LIBPNG_VER_STRING, NULL,
NULL, NULL);
    if (image->png_ptr == NULL){
        puts("Cannot create read struct!");
        exit(40);
    }
    image->info_ptr = png_create_info_struct(image->png_ptr);
    if (image->png_ptr == NULL){
        puts("Cannot create info struct!");
        exit(40);
    }
    if (setjmp(png_jmpbuf(image->png_ptr))){
        puts("Error when creating read struct!");
        png_destroy_read_struct(&(image->png_ptr), &(image->info_ptr),
NULL);
        exit(40);
    }
    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_passes = png_set_interlace_handling(image->png_ptr);

    if(png_get_color_type(image->png_ptr, image->info_ptr) !=
PNG_COLOR_TYPE_RGB){
        puts("Color type is not RGB!");
        exit(40);
    }

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

    image->rows = (png_bytepp)malloc(sizeof(png_bytepp) * image->height);
```

```
    for (int i = 0; i < image->height; i++)
        image->rows[i] = (png_bytep)calloc(image->width * 3,
sizeof(png_byte));
    png_read_image(image->png_ptr, image->rows);
    png_destroy_read_struct(&(image->png_ptr), &(image->info_ptr), NULL);
    fclose(fp);
}

void write_png(char* file_name, png_t* image){
    FILE *fp = fopen(file_name, "wb");
    if (fp == NULL){
      puts("Cannot open write file!");
      exit(41);
    }
    image->png_ptr = png_create_write_struct(PNG_LIBPNG_VER_STRING, NULL,
NULL, NULL);
    if (image->png_ptr == NULL){
        puts("Cannot create write struct!");
        exit(41);
    }
    image->info_ptr = png_create_info_struct(image->png_ptr);
    if (image->png_ptr == NULL){
        puts("Cannot create info struct!");
        exit(41);
    }
    if (setjmp(png_jmpbuf(image->png_ptr))){
        puts("Error when creating write struct!");
        png_destroy_write_struct(&(image->png_ptr), &(image->info_ptr));
        exit(41);
    }
    png_init_io(image->png_ptr, fp);
    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);
    png_write_image(image->png_ptr, image->rows);
    png_write_end(image->png_ptr, NULL);
    if (setjmp(png_jmpbuf(image->png_ptr))){
        puts("Error when writing PNG!");
        png_destroy_write_struct(&(image->png_ptr), &(image->info_ptr));
        exit(41);
    }
    png_destroy_write_struct(&(image->png_ptr), &(image->info_ptr));
    fclose(fp);
}
```