

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

КУРСОВАЯ РАБОТА
по дисциплине «Программирование»
Тема: Разработка программы для обработки изображений

Студент гр. 3344

Пачев Д.К.

Преподаватель

Глазунов С.А.

Санкт-Петербург

2024

ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

Студент Пачев Д.К.

Группа 3344

Тема работы «Разработка программы для обработки изображений»

Исходные данные:

- Программа обязательно должна иметь CLI
- Формат картинки PNG
- Каждую подзадачу следует вынести в отдельную функцию, функции сгруппировать в несколько файлов
- Все поля стандартных PNG заголовков в выходном файле должны иметь те же значения что и во входном
- Реализация интерфейса должна быть с использованием getopt

Содержание пояснительной записки:

- Содержание
- Введение
- Описание работы
- Описание работы программы
- Тестирование
- Заключение
- Список источников

Предполагаемый объем пояснительной записки:

Не менее 40 страниц.

Дата выдачи задания: 18.03.2024

Дата сдачи реферата: 15.05.2024

Дата защиты реферата: 15.05.2024

Студент

Пачев Д.К.

Преподаватель

Глазунов С.А.

АННОТАЦИЯ

Курсовой проект предполагает разработку программы обработки png-файлов с использованием CLI и, возможно, GUI. Программа должна проверять файл на соответствие формату PNG, в зависимости от переданных флагов выполнять различные функции. В процессе разработки используется библиотека libpng для работы с изображениями PNG без сжатия. Методы исследования включают анализ стандартных PNG заголовков, обработку входных параметров. Результатом работы является функциональная программа, способная обрабатывать изображения в соответствии с заданными параметрами, сохраняя структуру и характеристики исходного файла.

СОДЕРЖАНИЕ

Содержание	5
Введение	6
1. Описание варианта работы	7
2. Описание структуры проекта	9
2.1. Файловая структура	9
2.2. Структура Makefile	9
3. Описание структур и функций	10
3.1. Описание структур	10
3.2. Описание функций	10
4. Результаты	14
5. Заключение	15
6. Список использованных источников	16
7. Примеры тестирования	17
Приложение А	19

ВВЕДЕНИЕ

Цель работы: Написать программу для обработки png-изображения в зависимости от выбора пользователя.

Задачи:

- Реализация функций для считывания и сохранения изображений
- Разработка функций для обработки введенных пользователем флагов
- Разработка функций для обработки изображений
- Обработка ошибок

1. ОПИСАНИЕ ВАРИАНТА РАБОТЫ

Задание

Программа обязательно должна иметь CLI (опционально дополнительное использование GUI).

Программа должна реализовывать весь следующий функционал по обработке png-файла

Общие сведения

- Формат картинки PNG (рекомендуем использовать библиотеку libpng)
- без сжатия
- файл может не соответствовать формату PNG, т.е. необходимо проверка на PNG формат. Если файл не соответствует формату PNG, то программа должна завершиться с соответствующей ошибкой.
- обратите внимание на выравнивание; мусорные данные, если их необходимо дописать в файл для выравнивания, должны быть нулями.
- все поля стандартных PNG заголовков в выходном файле должны иметь те же значения что и во входном (разумеется кроме тех, которые должны быть изменены).

Программа должна иметь следующие функции по обработке изображений:

1. Отражение заданной области. Флаг для выполнения данной операции: `--mirror`. Этот функционал определяется:
 - выбором оси относительно которой отражать (горизонтальная или вертикальная). Флаг `--axis`, возможные значения `'x'` и `'y'`
 - Координатами левого верхнего угла области. Флаг `--left_up`, значение задаётся в формате `'left.up'`, где `left` – координата по `x`, `up` – координата по `y`
 - Координатами правого нижнего угла области. Флаг `--right_down`, значение задаётся в формате `'right.down'`, где `right` – координата по `x`, `down` – координата по `y`
2. Рисование пентаграммы в круге. Флаг для выполнения данной операции: `--pentagram`. Пентаграмма определяется:
 - координатами ее центра и радиусом. Флаги `--center` и `--radius`. Значение флаг `--center` задаётся в формате `'x.y'`, где `x` – координата по оси `x`, `y` – координата по оси `y`. Флаг `--radius` На вход принимает число больше 0
 - толщиной линий и окружности. Флаг `--thickness`. На вход принимает число больше 0
 - цветом линий и окружности. Флаг `--color` (цвет задаётся строкой `'rrr.ggg.bbb'`, где `rrr/ggg/bbb` – числа, задающие цветовую компоненту. пример `--color 255.0.0` задаёт красный цвет)

3. 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`)
4. 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`)

Каждую подзадачу следует вынести в отдельную функцию, функции сгруппировать в несколько файлов (например, функции обработки текста в один, функции ввода/вывода в другой). Сборка должна осуществляться при помощи `make` и `Makefile` или другой системы сборки

2. ОПИСАНИЕ СТРУКТУРЫ ПРОЕКТА

2.1. Файловая структура

Существует основной файл `main.c`, а также дополнительные файлы с расширением `.c`, в которых располагаются функции для работы с изображением. Для каждого такого файла есть заголовочный файл с расширением `.h`. В файлах `drawing_figures_functions.c`, `drawing_line_functions`, `mirror_function.c` прописаны функции для обработки изображения в зависимости от ввода пользователя. В файлах `input_functions.c`, `check_functions.c`, `parse_functions.c` реализованы функции для обработки флагов и их аргументов. В файле `png_functions.c` описаны функции для считывания и сохранения изображений.

2.2. Структура Makefile

Главная цель `Makefile all` – создание исполняемого файла `sw`. При создании `sw` компилируются все объектные файлы. Объектные файлы создаются из соответствующих им файлов с расширением `.c`, также есть зависимости в виде заголовочных файлов.

3. ОПИСАНИЕ СТРУКТУР И ФУНКЦИЙ

3.1. Описание структур

В программе описаны три структуры

- Point – структура для представления точки, содержит поля x, y.
- PNGImage – структура для представления png-изображения, содержит поля-характеристики изображения.
- Element – структура для типа данных наподобие словаря, содержит поля ключ и значение (key, value).

3.2. Описание функций

1. main.c

1.1 int main(int argc, char* argv[]) – внутри реализуется считывание флагов и аргументов от пользователя и вызов функции run.

2. input_functions.c

2.1 char* find_value(Element* dict, int len, char* key) – возвращает значение по ключу.

2.2 void check_output_and_input_match(char* input, char*output) – проверяет на совпадение названий входного и выходного файл

2.3 int check_extra_option(Element *dict, int len_dict) – считает количество основных флагов.

2.4 void check_flags(Element *dict, int len_dict, int *input_flag, int *output_flag, int *info_flag, int *help_flag, int *mirror_flag, int *axis_flag, int *left_up_flag, int *right_down_flag, int *pentagram_flag, int *center_flag, int *radius_flag, int *thickness_flag, int *color_flag, int *fill_flag, int *fill_color_flag, int *rect_flag, int *hexagon_flag) – проверяет, какие флаги были введены пользователем

2.5 char *find_main_option(Element *dict, int len_dict, int *input_flag, int *output_flag, int *info_flag, int *help_flag, int *

`mirror_flag, int *axis_flag, int *left_up_flag, int *right_down_flag, int *pentagram_flag, int *center_flag, int *radius_flag, int *thickness_flag, int *color_flag, int *fill_flag, int *fill_color_flag, int *rect_flag, int *hexagon_flag)` – определяет какую основную функцию нужно выполнить, а также корректность аргументов флагов

2.6 `void run(Element *dict, int len_dict)` – внутри с помощью функции `find_main_option()` определяется основная функция, считывается изображение, выполняется функция обработки изображения, сохраняется новое изображение

3. `check_functions.c`

3.1 `int is_inside_polygon(Point polygon[], int n, Point p)` – проверяет находится ли точка внутри выпуклого многоугольника.

3.2 `void check_corner_coords(char *left_up, char *right_down)` – проверяет верны ли координаты углов.

3.3 `void check_center_coords(char *coords)` – проверяет верны ли координаты центра

3.4 `void check_radius(char *radius)` – проверяет верно ли задан радиус

3.5 `void check_thickness(char *thickness)` – проверяет верно ли задана толщина

3.6 `void check_color(char *color)` – проверяет верно ли задан цвет

3.7 `void check_axis(char *axis)` – проверяет верно ли задана ось

4. `drawing_line_functions.c`

4.1 `void draw_pixel(PNGImage *image, int x, int y, int *colors)` – закрашивает пиксель по координатам `x`, `y` цветом `colors`

4.2 `void fill_circle(PNGImage *image, int x0, int y0, int r, int *colors)` – рисует с помощью алгоритма Брезенхема окружность и закрашивает ее

4.3 void draw_thick_line(PNGImage *image, int x1, int y1, int x2, int y2, int thickness, int *colors) – рисует линию с заданной толщиной по алгоритму Брезенхема

5. drawing_figures_functions.c

5.1 void draw_circle(PNGImage *image, int x0, int y0, int r, int thickness, int *colors) – рисует окружность с заданной толщиной

5.2 void draw_rectangle(PNGImage *image, int x1, int y1, int x2, int y2, int thickness, char *color, int fill, char *fill_color) – рисует прямоугольник по заданным параметрам

5.3 void draw_hexagon(PNGImage *image, int center_x, int center_y, int radius, int thickness, char *color, int fill, char *fill_color) – рисует шестиугольник по заданным параметрам

5.4 void draw_pentagram(PNGImage *image, int center_x, int center_y, int radius, int thickness, int *colors) – рисует пентаграмму в круге по заданным параметрам

6. mirror_function.c

6.1 void mirror_image(PNGImage *image, char axis, int x1, int y1, int x2, int y2) – отражает область изображения по заданным параметрам

7. parse_functions.c

7.1 int *parse_coords(char *coords) – преобразует координаты из строкового вида в массив целых чисел размера 2

7.2 int *parse_color(char *color) – преобразует цвет из строкового вида в массив целых чисел размера 3

8. png_functions.c

8.1 void read_png_file(char *file_name, PNGImage *image) – функция для считывания изображения, заполняет поля структуры PNGImage

8.2 void write_png_file(char *file_name, PNGImage *image) –
функция для сохранения создания и сохранения нового
изображения

4. РЕЗУЛЬТАТЫ

Разработанная программа выполняет поставленную задачу.

Функционал работы программы включает в себя считывание изображения, обработку флагов, обработку изображения, сохранения изображения. Данная программа включает в себя:

- Обработка флагов – пользователь может вводить флаги и аргументы для выбора опции для обработки изображения
- Считывание и сохранения изображения – программа может считывать и сохранять изображения формата png и выполнять различные действия с картинками
- Сборку программы – с помощью Makefile можно компилировать только те файлы, которые были изменены, что очень эффективно
- Обработка ошибок – если пользователь ввел какой-то неверный флаг, или во время работы программы возникла проблема, то выведется ошибка с указанием, где она произошла

5. ЗАКЛЮЧЕНИЕ

Программа верно выполняет все поставленные задачи. В ходе тестирования не было выявлено ошибок. В ходе выполнения работы были развиты навыки работы с изображением, библиотекой `libpng` и `getopt`

6. СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

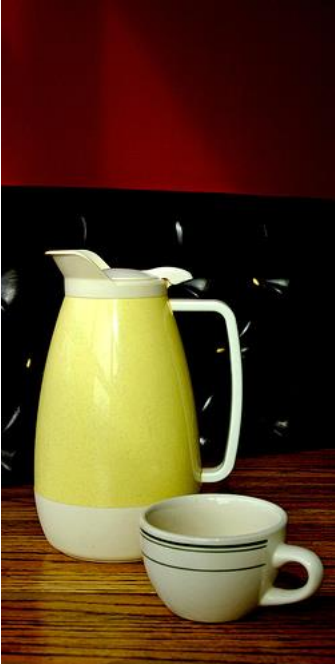

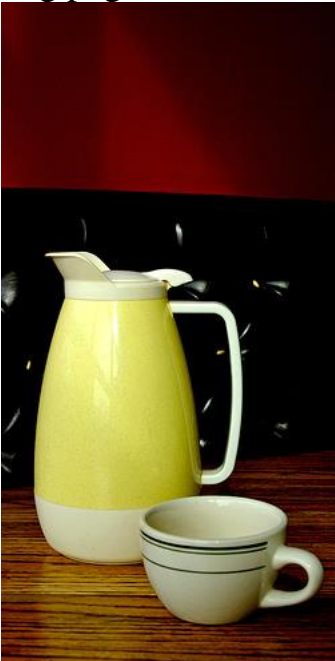
1. Документация libpng // libpng docs URL:



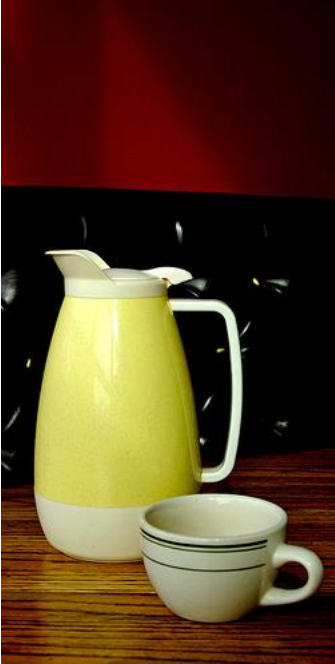
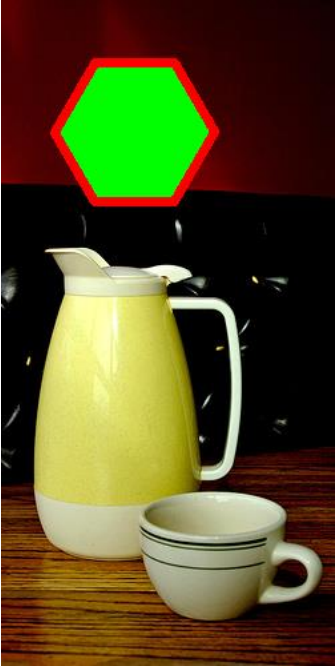
<http://www.libpng.org/pub/png/libpng-1.2.5-manual.html> (дата обращения: 25.04.2024).

2. Документация getopt // gnu.org URL:

https://www.gnu.org/software/libc/manual/html_node/Getopt.html (дата обращения: 25.04.2024).

7. ПРИМЕРЫ ТЕСТИРОВАНИЯ

№	Ввод	Вывод
1	<p>Команда ввода: ./cw –pentagram –center 100.109 –radius 100 –thickness 10 –color 255.0.0 --input img.png –output output.png</p> <p>img.png:</p> 	<p>output.png:</p> 
2	<p>Команда ввода: ./cw –mirror –left_up 10.10 –right_down 100.400 –axis y --input img.png –output output.png</p> <p>img.png:</p> 	<p>output.png:</p> 

3	<p>Команда ввода: <code>./cw -rect -left_up 110.51 -right_down 322.258 -thickness 23 -color 255.0.0 -fill -fill_color 0.255.0 --input img.png -output output.png</code></p> <p>img.png:</p> 	<p>output.png:</p> 
4	<p>Команда ввода: <code>./cw -hexagon -center 100.100 -thickness 5 -radius 60 -color 255.0.0 -fill -fill_color 0.255.0 --input img.png -output output.png</code></p> <p>img.png:</p> 	<p>output.png:</p> 

ПРИЛОЖЕНИЕ А

main.c:

```
#include <stdio.h>
#include <png.h>
#include <stdlib.h>
#include <string.h>
#include <getopt.h>
#include "drawing_line_functions.h"
#include "input_functions.h"

int main(int argc, char *argv[]) {
    const char *short_options = "i:o:h";
    Element *dict = malloc(sizeof(Element) * 200);
    int len_dict = 0;
    const struct option long_options[] = {
        {"input",      required_argument, NULL, 'i'},
        {"output",     required_argument, NULL, 'o'},
        {"info",       no_argument,      NULL, 'i'},
        {"help",       no_argument,      NULL, 'h'},
        {"mirror",     no_argument,      NULL, 0},
        {"axis",       required_argument, NULL, 0},
        {"left_up",    required_argument, NULL, 0},
        {"right_down", required_argument, NULL, 0},
        {"pentagram",  no_argument,      NULL, 0},
        {"center",     required_argument, NULL, 0},
        {"radius",     required_argument, NULL, 0},
        {"thickness",  required_argument, NULL, 0},
        {"color",      required_argument, NULL, 0},
        {"fill",       no_argument,      NULL, 0},
        {"fill_color", required_argument, NULL, 0},
        {"rect",       no_argument,      NULL, 0},
        {"hexagon",    no_argument,      NULL, 0},
        {NULL, 0,      NULL, 0}
    };

    int option;
    int option_index = 0;
    char *option_name;

    while ((option = getopt_long(argc, argv, short_options,
                                long_options, &option_index)) != -1) {
        if (option == '?') {
            printf("unknown option - %s\n", argv[optind - 1]);
            continue;
        }
        option_name = strdup(long_options[option_index].name);
        // save option and value
        if (optarg != NULL) {
            dict[len_dict].key = option_name;
            dict[len_dict++].value = optarg;
        } else {
            dict[len_dict].key = option_name;
            dict[len_dict++].value = "";
        }
    }
};
```

```

int input_flag = 0;
for (int i = 0; i < len_dict; i++) {
    if (strcmp("input", dict[i].key) == 0) {
        input_flag = 1;
    }
}
if (input_flag == 0) {
    if (optind < argc) {
        dict[len_dict].key = "input";
        dict[len_dict++].value = argv[argc - 1];
    }
}
run(dict, len_dict);
free(dict);

return 0;
}

```

check_functions.c:

```
#include "check_functions.h"
int is_inside_polygon(Point polygon[], int n, Point p) {
    int i, j, c = 0;
    for (i = 0, j = n - 1; i < n; j = i++) {
        if (((polygon[i].y > p.y) != (polygon[j].y > p.y)) &&
            (p.x < (polygon[j].x - polygon[i].x) * (p.y - polygon[i].y) /
              (polygon[j].y - polygon[i].y) + polygon[i].x))
            c = !c;
    }
    return c;
}

void check_corner_coords(char *left_up, char *right_down) {
    // check with regex that coords are like "x.y"
    regex_t regex;
    int reti = regcomp(&regex, "^([0-9]+\\.([0-9])+$", REG_EXTENDED);
    if (reti) {
        fprintf(stderr, "Could not compile regex\n");
        exit(1);
    }
    reti = regexec(&regex, left_up, 0, NULL, 0);
    if (reti) {
        printf("%s", left_up);
        fprintf(stderr, "Left up coords are not correct\n");
        exit(41);
    }
    reti = regexec(&regex, right_down, 0, NULL, 0);
    if (reti) {
        fprintf(stderr, "Right down coords are not correct\n");
        exit(41);
    }
    int *left_up_coords = parse_coords(left_up);
    int *right_down_coords = parse_coords(right_down);
    if (left_up_coords[0] > right_down_coords[0] || left_up_coords[1] >
        right_down_coords[1]) {
        char* temp = malloc(sizeof(char) * (strlen(left_up) + 1));
        strcpy(temp, left_up);
        strcpy(left_up, right_down);
        strcpy(right_down, temp);
        free(temp);
    }
    free(left_up_coords);
    free(right_down_coords);
}

void check_center_coords(char *coords) {
    regex_t regex;
    int reti = regcomp(&regex, "^([0-9]+\\.([0-9])+$", REG_EXTENDED);
    if (reti) {
        fprintf(stderr, "Could not compile regex\n");
        exit(1);
    }
    reti = regexec(&regex, coords, 0, NULL, 0);
    if (reti) {
        fprintf(stderr, "Coords are not correct\n");
        exit(41);
    }
}
```

```

void check_radius(char *radius) {
    regex_t regex;
    int reti = regcomp(&regex, "[0-9]+", REG_EXTENDED);
    if (reti) {
        fprintf(stderr, "Could not compile regex\n");
        exit(1);
    }
    reti = regexec(&regex, radius, 0, NULL, 0);
    if (reti || atoi(radius) < 0) {
        fprintf(stderr, "Radius is not correct\n");
        exit(41);
    }
}

void check_thickness(char *thickness) {
    regex_t regex;
    int reti = regcomp(&regex, "[0-9]+", REG_EXTENDED);
    if (reti) {
        fprintf(stderr, "Could not compile regex\n");
        exit(1);
    }
    reti = regexec(&regex, thickness, 0, NULL, 0);
    if (reti || atoi(thickness) < 0) {
        fprintf(stderr, "Thickness is not correct\n");
        exit(41);
    }
}

void check_color(char *color) {
    regex_t regex;
    // check with regex that color is like "x.y.z" and x,y,z from 0 to
255
    int reti = regcomp(&regex, "[0-9]+\\.\\.[0-9]+\\.\\.[0-9]+", REG_EXTENDED);
    if (reti) {
        fprintf(stderr, "Could not compile regex\n");
        exit(1);
    }
    int *color_rgb = parse_color(color);
    reti = regexec(&regex, color, 0, NULL, 0);
    if (reti || color_rgb[0] > 255 || color_rgb[1] > 255 || color_rgb[2]
> 255 || color_rgb[0] < 0 || color_rgb[1] <
0 ||
        color_rgb[2] < 0) {
        free(color_rgb);
        fprintf(stderr, "Color is not correct\n");
        exit(41);
    }
    free(color_rgb);
}

void check_axis(char *axis) {
    if (strcmp(axis, "x") != 0 && strcmp(axis, "y") != 0) {
        fprintf(stderr, "Axis is not correct\n");
        exit(41);
    }
}

```

check_functions.h:

```
#ifndef COURSE_WORK_CHECK_FUNCTIONS_H
#define COURSE_WORK_CHECK_FUNCTIONS_H
#include "structs.h"
#include <regex.h>
#include <stdlib.h>
#include <string.h>
#include "parse_functions.h"
int is_inside_polygon(Point polygon[], int n, Point p);
void check_corner_coords(char *left_up, char *right_down);
void check_center_coords(char *coords);
void check_radius(char *radius);
void check_thickness(char *thickness);
void check_color(char *color);
void check_axis(char *axis);
#endif //COURSE_WORK_CHECK_FUNCTIONS_H
```

drawing_figures_functions.c:

```
#include "drawing_figures_functions.h"
void draw_circle(PNGImage *image, int x0, int y0, int r, int thickness,
int *colors) {
    int x = 0;
    int radius;
    int y = r;
    int delta = 1 - 2 * r;
    int error = 0;
    if (thickness % 2 == 0) {
        radius = thickness / 2;
    } else if (thickness == 1) {
        radius = 0;
    } else {
        radius = (thickness + 1) / 2;
    }
    while (y >= x) {
        fill_circle(image, x0 + x, y0 + y, radius, colors);
        fill_circle(image, x0 + x, y0 - y, radius, colors);
        fill_circle(image, x0 - x, y0 + y, radius, colors);
        fill_circle(image, x0 - x, y0 - y, radius, colors);
        fill_circle(image, x0 + y, y0 + x, radius, colors);
        fill_circle(image, x0 + y, y0 - x, radius, colors);
        fill_circle(image, x0 - y, y0 + x, radius, colors);
        fill_circle(image, x0 - y, y0 - x, radius, colors);

        error = 2 * (delta + y) - 1;
        if ((delta < 0) && (error <= 0)) {
            delta += 2 * ++x + 1;
            continue;
        }
        if ((delta > 0) && (error > 0)) {
            delta -= 2 * --y + 1;
            continue;
        }
        delta += 2 * (++x - --y);
    }
}

void draw_rectangle(PNGImage *image, int x1, int y1, int x2, int y2, int
thickness, char *color, int fill,
                    char *fill_color) {
    char *copy_color = strdup(color);

    char *token = strtok(copy_color, ".");
    int colors[3];
    for (int i = 0; i < 3; i++) {
        colors[i] = atoi(token);
        token = strtok(NULL, ".");
    }
    int min_x = x1 < x2 ? x1 : x2;
    int max_x = x1 < x2 ? x2 : x1;
    int min_y = y1 < y2 ? y1 : y2;
    int max_y = y1 < y2 ? y2 : y1;
    x1 = min_x;
    x2 = max_x;
    y1 = min_y;
    y2 = max_y;
```



```

    if (fill) {
        int fill_colors[3];
        char *copy_fill_color = strdup(fill_color);
        token = strtok(copy_fill_color, ".");
        for (int i = 0; i < 3; i++) {
            fill_colors[i] = atoi(token);
            token = strtok(NULL, ".");
        }
        for (int y = y1; y < y2+1; y++) {
            for (int x = x1; x < x2+1; x++) {
                draw_pixel(image, x, y, fill_colors);
            }
        }
        free(copy_fill_color);
    }
    draw_thick_line(image, x1, y1, x2 + 1, y1, thickness, colors );
    draw_thick_line(image, x1, y1, x1, y2 + 1, thickness, colors );
    draw_thick_line(image, x1 + 1, y2, x2 + 1, y2, thickness, colors );
    draw_thick_line(image, x2, y1 + 1, x2, y2 + 1, thickness, colors );
    free(copy_color);
}

void draw_hexagon(PNGImage *image, int center_x, int center_y, int
radius, int thickness, char *color, int fill,
char *fill_color) {
    // check radius
    if (radius >= 0) {
        char *copy_color = strdup(color);
        char *token = strtok(copy_color, ".");
        int colors[3];
        for (int i = 0; i < 3; i++) {
            colors[i] = atoi(token);
            token = strtok(NULL, ".");
        }
        int x1 = round(center_x - radius/2);
        int y1 = round(center_y-sqrt(radius*radius-(radius/2)*radius/2));
        int x2 = round(center_x + radius/2);
        int y2 = y1;
        int x3 = center_x +radius;
        int y3 = center_y;
        int x4 = x2;
        int y4 = round(center_y+sqrt(radius*radius-(radius/2)*radius/2));
        int x5 = x1;
        int y5 = y4;
        int x6 = center_x-radius;
        int y6 = center_y;
        Point hexagon[] = {{x1, y1},
                           {x2, y2},
                           {x3, y3},
                           {x4, y4},
                           {x5, y5},
                           {x6, y6}};

        if (fill) {
            int fill_colors[3];
            char *copy_fill_color = strdup(fill_color);
            token = strtok(copy_fill_color, ".");
            for (int i = 0; i < 3; i++) {

```

```

        fill_colors[i] = atoi(token);
        token = strtok(NULL, ".");
    }
    for (int y = y1 - 1; y < y4 + 1; y++) {
        for (int x = x6; x < x3 + 1; x++) {
            int flag = 0;
            Point point = {x, y};
            if (y >= image->height || x >= image->width || y < 0
|| x < 0) {
                continue;
            }
            if (is_inside_polygon(hexagon, 6, point)) {
                png_byte *ptr = &(image->row_pointers[y][x * 3]);
                ptr[0] = fill_colors[0];
                ptr[1] = fill_colors[1];
                ptr[2] = fill_colors[2];
            }
        }
    }
    free(copy_fill_color);
}
draw_thick_line(image, x1, y1, x2+1, y2, thickness, colors);
draw_thick_line(image, x2, y2, x3+1, y3+1, thickness, colors);
draw_thick_line(image, x3, y3, x4-1, y4+1, thickness, colors);
draw_thick_line(image, x4, y4, x5-1, y5, thickness, colors);
draw_thick_line(image, x5, y5, x6-1, y6-1, thickness, colors);
draw_thick_line(image, x6, y6, x1+1, y1-1, thickness, colors);
free(copy_color);
}
}
void draw_pentagram(PNGImage *image, int center_x, int center_y, int
radius, int thickness, int* colors) {
    draw_circle(image, center_x, center_y, radius, thickness, colors);
    int x1 = center_x;
    int y1 = center_y - radius;
    int x2 = round(center_x + radius * sin(0.2 * M_PI));
    int y2 = round(center_y + radius * cos(0.2 * M_PI));

    int x3 = round(center_x - radius * sin(0.2 * M_PI));
    int y3 = round(center_y + radius * cos(0.2 * M_PI));

    int x4 = round(center_x + radius * sin(0.4 * M_PI));
    int y4 = round(center_y - radius * cos(0.4 * M_PI));

    int x5 = round(center_x - radius * sin(0.4 * M_PI));
    int y5 = round(center_y - radius * cos(0.4 * M_PI));

    draw_thick_line(image, x1, y1, x2+1, y2+1, thickness, colors);
    draw_thick_line(image, x5, y5, x2+1, y2+1, thickness, colors);
    draw_thick_line(image, x5, y5, x4+1, y4, thickness, colors);
    draw_thick_line(image, x4, y4, x3-1, y3+1, thickness, colors);
    draw_thick_line(image, x1, y1, x3-1, y3+1, thickness, colors);
}

```

drawing_figures_functions.h:

```
#ifndef COURSE_WORK_DRAWING_FIGURES_FUNCTIONS_H
#define COURSE_WORK_DRAWING_FIGURES_FUNCTIONS_H
#include <math.h>
#include <string.h>
#include "structs.h"
#include "drawing_line_functions.h"
#include "check_functions.h"

void draw_circle(PNGImage *image, int x0, int y0, int r, int thickness,
int *colors);
void draw_rectangle(PNGImage *image, int x1, int y1, int x2, int y2, int
thickness, char *color, int fill,
char *fill_color);
void draw_hexagon(PNGImage *image, int center_x, int center_y, int
radius, int thickness, char *color, int fill,
char *fill_color);
void draw_pentagram(PNGImage *image,int center_x, int center_y,int
radius,int thickness,int* colors);
#endif //COURSE_WORK_DRAWING_FIGURES_FUNCTIONS_H
```

drawing_line_functions.c:

```
#include "drawing_line_functions.h"

void draw_pixel(PNGImage *image, int x, int y, int *colors) {
    if (!(x < 0 || y < 0 || x >= image->width || y >= image->height)) {
        image->row_pointers[y][x * 3] = colors[0];
        image->row_pointers[y][x * 3 + 1] = colors[1];
        image->row_pointers[y][x * 3 + 2] = colors[2];
    }
}

void fill_circle(PNGImage *image, int x0, int y0, int r, int *colors) {
    int x = 0;
    int y = r;
    int delta = 3 - 2 * y;
    while (y >= x) {
        draw_pixel(image, x0 + x, y0 + y, colors);
        draw_pixel(image, x0 + x, y0 - y, colors);
        draw_pixel(image, x0 - x, y0 + y, colors);
        draw_pixel(image, x0 - x, y0 - y, colors);
        draw_pixel(image, x0 + y, y0 + x, colors);
        draw_pixel(image, x0 + y, y0 - x, colors);
        draw_pixel(image, x0 - y, y0 + x, colors);
        draw_pixel(image, x0 - y, y0 - x, colors);
        delta += delta < 0 ? 4 * x + 6 : 4 * (x - y--) + 10;
        ++x;
    }
    for (int y = -r; y <= r; y++) {
        if ((y0+y)<0 || (y0+y)>=image->height){
            continue;
        }
        for (int x = -r; x <= r; x++) {
            if (((x0+x)>=0) && ((x0+x)<image->width)&& (x * x + y * y <=
r * r)) {
                draw_pixel(image, x0 + x, y0 + y, colors);
            }
        }
    }
}

void draw_thick_line(PNGImage *image, int x1, int y1, int x2, int y2, int
thickness, int *colors) {
    int dx = abs(x2 - x1);
    int dy = abs(y2 - y1);
    int sx = x1 < x2 ? 1 : -1;
    int sy = y1 < y2 ? 1 : -1;
    int err = dx - dy;
    int e2;
    int x = x1;
    int y = y1;
    while (x != x2 || y != y2) {
        draw_pixel(image, x, y, colors);
        if (thickness % 2 == 0) {
            fill_circle(image, x, y, thickness / 2, colors);
        } else if (thickness == 1) {
            fill_circle(image, x, y, 0, colors);
        } else {

```

```

        fill_circle(image, x, y, (thickness + 1) / 2, colors);
    }
    e2 = 2 * err;
    if (e2 > -dy) {
        err -= dy;
        x += sx;
    }
    if (e2 < dx) {
        err += dx;
        y += sy;
    }
}
}

```

drawing_line_functions.h:

```
#ifndef COURSE_WORK_DRAWING_LINE_FUNCTIONS_H
#define COURSE_WORK_DRAWING_LINE_FUNCTIONS_H
#include <png.h>
#include "structs.h"
#include <stdlib.h>
void draw_pixel(PNGImage *image, int x, int y, int *colors);
void fill_circle(PNGImage *image, int x0, int y0, int r, int *colors);
void draw_thick_line(PNGImage *image, int x1, int y1, int x2, int y2, int
thickness, int *colors);
#endif //COURSE_WORK_DRAWING_LINE_FUNCTIONS_H
```

input_functions.c:

```
#include "input_functions.h"
char *find_value(Element *dict, int len, char *key) {
    for (int i = 0; i < len; i++) {
        if (strcmp(key, dict[i].key) == 0) {
            return dict[i].value;
        }
    }
}

void check_output_and_input_match(char* input, char* output){
    if (strcmp(input,output)==0){
        fprintf(stderr,"Input and output files are the same\n");
        exit(41);
    }
}

int check_extra_option(Element *dict, int len_dict) {
    int len_main_options = 6;
    char *main_options[] = {"mirror", "pentagram", "rect", "hexagon",
    "info", "input"};
    int count_main_options = 0;
    for (int i = 0; i < len_dict; i++) {
        for (int j = 0; j < len_main_options; j++) {
            if (strcmp(dict[i].key, main_options[j]) == 0) {
                count_main_options++;
            }
        }
    }
    return count_main_options;
}

void check_flags(Element *dict, int len_dict, int *input_flag, int
*output_flag, int *info_flag, int *help_flag, int *
mirror_flag, int *axis_flag,
                int *left_up_flag, int *right_down_flag, int
*pentagram_flag, int *center_flag, int *radius_flag, int *
thickness_flag, int *color_flag, int *fill_flag, int *fill_color_flag,
int *rect_flag, int *hexagon_flag) {

    for (int i = 0; i < len_dict; i++) {
        if (strcmp("input", dict[i].key) == 0) {
            *input_flag = 1;
        } else if (strcmp("info", dict[i].key) == 0) {
            *info_flag = 1;
        } else if (strcmp("help", dict[i].key) == 0) {
            *help_flag = 1;
        } else if (strcmp("mirror", dict[i].key) == 0) {
            *mirror_flag = 1;
        } else if (strcmp("axis", dict[i].key) == 0) {
            *axis_flag = 1;
        } else if (strcmp("left_up", dict[i].key) == 0) {
            *left_up_flag = 1;
        } else if (strcmp("right_down", dict[i].key) == 0) {
            *right_down_flag = 1;
        } else if (strcmp("pentagram", dict[i].key) == 0) {
            *pentagram_flag = 1;
        } else if (strcmp("center", dict[i].key) == 0) {
            *center_flag = 1;
        }
    }
}
```

```

    } else if (strcmp("radius", dict[i].key) == 0) {
        *radius_flag = 1;
    } else if (strcmp("thickness", dict[i].key) == 0) {
        *thickness_flag = 1;
    } else if (strcmp("color", dict[i].key) == 0) {
        *color_flag = 1;
    } else if (strcmp("fill", dict[i].key) == 0) {
        *fill_flag = 1;
    } else if (strcmp("fill_color", dict[i].key) == 0) {
        *fill_color_flag = 1;
    } else if (strcmp("rect", dict[i].key) == 0) {
        *rect_flag = 1;
    } else if (strcmp("hexagon", dict[i].key) == 0) {
        *hexagon_flag = 1;
    }
    else if (strcmp("output", dict[i].key) == 0) {
        *output_flag = 1;
    }
}

char *find_main_option(Element *dict, int len_dict, int *input_flag, int
*output_flag, int *info_flag, int *help_flag, int *
mirror_flag, int *axis_flag, int *left_up_flag, int *right_down_flag, int
*pentagram_flag, int
        *center_flag, int *radius_flag, int
*thickness_flag, int *color_flag, int *fill_flag, int
        *fill_color_flag, int *rect_flag, int
*hexagon_flag) {
    check_flags(dict, len_dict, input_flag, output_flag, info_flag,
help_flag, mirror_flag, axis_flag,
        left_up_flag, right_down_flag, pentagram_flag,
center_flag, radius_flag,
        thickness_flag, color_flag, fill_flag, fill_color_flag,
rect_flag, hexagon_flag);
    int count_main_options = check_extra_option(dict, len_dict);
    if (count_main_options == 2) {

        // переделать проверки
        if ((*mirror_flag)==1) {
            if ((*axis_flag)==1 && (*left_up_flag)==1 &&
(*right_down_flag)==1 && (*output_flag)==1 && (*input_flag)
==1) {
                check_corner_coords(find_value(dict, len_dict,
"left_up"), find_value(dict, len_dict, "right_down"));
                check_axis(find_value(dict, len_dict, "axis"));
                check_output_and_input_match(find_value(dict, len_dict,
"input"), find_value(dict, len_dict, "output"));
                return "mirror";
            } else {
                fprintf(stderr, "Недостаточное количество флагов для
функции mirror\n");
                exit(41);
            }
        }
        else if ((*pentagram_flag)==1) {
            if ((*center_flag)==1 && (*radius_flag)==1 &&
(*thickness_flag)==1 && (*color_flag)==1 && (*output_flag)

```



```

==1 && (*input_flag)==1){
    check_center_coords(find_value(dict, len_dict,
"center"));
    check_radius(find_value(dict, len_dict, "radius"));
    check_thickness(find_value(dict, len_dict, "thickness"));
    check_color(find_value(dict, len_dict, "color"));
    check_output_and_input_match(find_value(dict, len_dict,
"input"), find_value(dict, len_dict, "output"));
    return "pentagram";
} else {
    fprintf(stderr, "Недостаточное количество флагов для
функции pentagram\n");
    exit(41);
}
} else if ((*rect_flag)==1) {
    if ((*left_up_flag)==1 && (*right_down_flag)==1 &&
(*thickness_flag)==1 && (*color_flag)==1 &&
(*output_flag)==1 && (*input_flag)==1){
        check_corner_coords(find_value(dict, len_dict,
"left_up"), find_value(dict, len_dict, "right_down"));
        check_thickness(find_value(dict, len_dict, "thickness"));
        check_color(find_value(dict, len_dict, "color"));
        check_output_and_input_match(find_value(dict, len_dict,
"input"), find_value(dict, len_dict, "output"));
        if ((*fill_flag) == 1 && (*fill_color_flag) == 1){
            check_color(find_value(dict, len_dict,
"fill_color"));
        }
        if ((*fill_flag) == 1 && (*fill_color_flag) == 0){
            fprintf(stderr, "Не задан цвет заливки\n");
            exit(41);
        }
        return "rect";
    } else {
        fprintf(stderr, "Недостаточное количество флагов для
функции rect\n");
        exit(41);
    }
} else if ((*hexagon_flag)==1) {
    if ((*center_flag)==1 && (*radius_flag)==1 &&
(*thickness_flag)==1 && (*color_flag)==1 && (*output_flag)
==1 && (*input_flag)==1) {
        check_center_coords(find_value(dict, len_dict,
"center"));
        check_radius(find_value(dict, len_dict, "radius"));
        check_thickness(find_value(dict, len_dict, "thickness"));
        check_color(find_value(dict, len_dict, "color"));
        check_output_and_input_match(find_value(dict, len_dict,
"input"), find_value(dict, len_dict, "output"));
        if ((*fill_flag) == 1 && (*fill_color_flag) == 1){
            check_color(find_value(dict, len_dict,
"fill_color"));
        }
        if ((*fill_flag) == 1 && (*fill_color_flag) == 0){
            fprintf(stderr, "Не задан цвет заливки\n");

```

```

        exit(41);
    }
    return "hexagon";
} else {
    fprintf(stderr, "Недостаточное количество флагов для
функции hexagon\n");
    exit(41);
}
} else if ((*info_flag)==1) {
    return "info";
}
} else if (count_main_options > 2) {
    fprintf(stderr, "Можно выполнить только 1 основную функцию, а не
несколько\n");
    exit(41);
} else {
    return "Not main option";
}
}

```

```

void run(Element *dict, int len_dict) {
    int input_flag = 0;
    int output_flag = 0;
    int info_flag = 0;
    int help_flag = 0;
    int mirror_flag = 0;
    int axis_flag = 0;
    int left_up_flag = 0;
    int right_down_flag = 0;
    int pentagram_flag = 0;
    int center_flag = 0;
    int radius_flag = 0;
    int thickness_flag = 0;
    int color_flag = 0;
    int fill_flag = 0;
    int fill_color_flag = 0;
    int rect_flag = 0;
    int hexagon_flag = 0;

    char *main_option = find_main_option(dict, len_dict, &input_flag,
&output_flag, &info_flag, &help_flag, &mirror_flag, &axis_flag,
&left_up_flag, &right_down_flag,
&pentagram_flag, &center_flag, &radius_flag, &thickness_flag,
&color_flag,
&fill_flag, &fill_color_flag,
&rect_flag, &hexagon_flag);

    if (strcmp(main_option, "mirror") == 0) {
        PNGImage image;
        read_png_file(find_value(dict, len_dict, "input"), &image);
        char *axis = find_value(dict, len_dict, "axis");
        char *left_up = find_value(dict, len_dict, "left_up");
        char *right_down = find_value(dict, len_dict, "right_down");
        int *left_up_coords = parse_coords(left_up);
        int *right_down_coords = parse_coords(right_down);
    }
}

```

```

        mirror_image(&image, axis[0], left_up_coords[0],
left_up_coords[1], right_down_coords[0], right_down_coords[1]);
        write_png_file(find_value(dict, len_dict, "output"), &image);
        free(left_up_coords);
        free(right_down_coords);
    }
    else if(strcmp(main_option, "pentagram")==0){
        PNGImage image;
        read_png_file(find_value(dict, len_dict, "input"), &image);
        char *center = find_value(dict, len_dict, "center");
        char *radius = find_value(dict, len_dict, "radius");
        char *thickness = find_value(dict, len_dict, "thickness");
        char *color = find_value(dict, len_dict, "color");
        int *center_coords = parse_coords(center);
        int *colors = parse_color(color);

draw_pentagram(&image, center_coords[0], center_coords[1], atoi(radius), atoi
(thickness), colors);
        write_png_file(find_value(dict, len_dict, "output"), &image);
        free(center_coords);
        free(colors);
    }
    else if(strcmp(main_option, "rect")==0){
        PNGImage image;
        read_png_file(find_value(dict, len_dict, "input"), &image);
        char *left_up = find_value(dict, len_dict, "left_up");
        char *right_down = find_value(dict, len_dict, "right_down");
        char *thickness = find_value(dict, len_dict, "thickness");
        char *color = find_value(dict, len_dict, "color");
        char* fill_color = "0.0.0";
        if (fill_flag){
            fill_color = find_value(dict, len_dict, "fill_color");
        }
        int *left_up_coords = parse_coords(left_up);
        int *right_down_coords = parse_coords(right_down);

draw_rectangle(&image, left_up_coords[0], left_up_coords[1], right_down_coor
ds[0], right_down_coords[1], atoi
(thickness), color, fill_flag, fill_color);
        write_png_file(find_value(dict, len_dict, "output"), &image);
        free(left_up_coords);
        free(right_down_coords);
    }
    else if(strcmp(main_option, "hexagon")==0){
        PNGImage image;
        read_png_file(find_value(dict, len_dict, "input"), &image);
        char *center = find_value(dict, len_dict, "center");
        char *radius = find_value(dict, len_dict, "radius");
        char *thickness = find_value(dict, len_dict, "thickness");
        char *color = find_value(dict, len_dict, "color");
        int fill = fill_flag == 1 ? 1 : 0;
        char *fill_color = "0.0.0";
        if (fill){
            fill_color = find_value(dict, len_dict, "fill_color");
        }
    }
}

```

```

    }
    int *center_coords = parse_coords(center);

    draw_hexagon(&image, center_coords[0], center_coords[1], atoi(radius), atoi(thickness), color, fill, fill_color);
    write_png_file(find_value(dict, len_dict, "output"), &image);
}
else if(strcmp(main_option, "info") == 0) {
    PNGImage image;
    read_png_file(find_value(dict, len_dict, "input"), &image);
    printf("Width: %d\n", image.width);
    printf("Height: %d\n", image.height);
    printf("Color type: %d\n", image.color_type);
    printf("Bit depth: %d\n", image.bit_depth);
}
else {
    if (help_flag) {
        printf("Course work for option 5.15, created by Danila Pachev.\n");
    }
}
}

```

input_functions.h:

```
#ifndef COURSE_WORK_INPUT_FUNCTIONS_H
#define COURSE_WORK_INPUT_FUNCTIONS_H
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "structs.h"
#include "png_functions.h"
#include "drawing_figures_functions.h"
#include "parse_functions.h"
#include "mirror_function.h"
char *find_value(Element *dict, int len, char *key);
void check_output_and_input_match(char* input, char* output);
int check_extra_option(Element *dict, int len_dict);
void check_flags(Element *dict, int len_dict, int *input_flag, int
*output_flag, int *info_flag, int *help_flag, int *
    mirror_flag, int *axis_flag, int *left_up_flag, int
*right_down_flag, int *pentagram_flag, int *center_flag,
    int *radius_flag, int *thickness_flag, int *color_flag, int
*fill_flag, int *fill_color_flag, int *rect_flag,
    int *hexagon_flag);

char *find_main_option(Element *dict, int len_dict, int *input_flag, int
*output_flag, int *info_flag, int *help_flag, int *
    mirror_flag, int *axis_flag, int *left_up_flag, int *right_down_flag, int
*pentagram_flag, int
    *center_flag, int *radius_flag, int
*thickness_flag, int *color_flag, int *fill_flag, int
    *fill_color_flag, int *rect_flag, int
*hexagon_flag);
void run(Element *dict, int len_dict);
#endif //COURSE_WORK_INPUT_FUNCTIONS_H
```

mirror_function.c:

```
#include "mirror_function.h"

void mirror_image(PNGImage *image, char axis, int x1, int y1, int x2, int
y2) {
    if (axis == 'x') {
        if (y2>image->height){
            y2 = image->height;
        }
        if (x2>image->width){
            x2 = image->width;
        }
        if (y1<0){
            y1 = 0;
        }
        if (x1<0){
            x1 = 0;
        }
        if (x1>=image->width){
            return;
        }
        if (y1>=image->height){
            return;
        }

        for (int y = y1; y < y2; y++) {
            png_byte *row = image->row_pointers[y];
            int i = 0;
            for (int x = x1; x < round((x1+x2)/2); x++) {
                png_byte *ptr1 = &(row[x * 3]);

                int color1 = ptr1[0];
                int color2 = ptr1[1];
                int color3 = ptr1[2];
                png_byte *ptr2 = &(row[(x2 - i++) * 3]);
                ptr1[0] = ptr2[0];
                ptr1[1] = ptr2[1];
                ptr1[2] = ptr2[2];
                ptr2[0] = color1;
                ptr2[1] = color2;
                ptr2[2] = color3;
            }
        }
    } else if (axis == 'y') {
        if (y2>image->height){
            y2 = image->height-1;
        }
        if (x2>image->width){
            x2 = image->width-1;
        }
        if (y1<0){
            y1 = 0;
        }
        if (x1<0){
            x1 = 0;
        }
    }
}
```

```

if (x1>=image->width){
    return;
}
if (y1>=image->height){
    return;
}

for (int y = y1; y < round((y2+y1)/2); y++) {
    png_byte *row = image->row_pointers[y];
    png_byte *row2 = image->row_pointers[y2 - y+y1];
    for (int x = x1; x < x2; x++) {
        png_byte *ptr1 = &(row[x * 3]);
        int color1 = ptr1[0];
        int color2 = ptr1[1];
        int color3 = ptr1[2];
        png_byte *ptr2 = &(row2[x * 3]);
        ptr1[0] = ptr2[0];
        ptr1[1] = ptr2[1];
        ptr1[2] = ptr2[2];
        ptr2[0] = color1;
        ptr2[1] = color2;
        ptr2[2] = color3;
    }
}
}
}

```

mirror_function.h:

```
#ifndef COURSE_WORK_MIRROR_FUNCTION_H
#define COURSE_WORK_MIRROR_FUNCTION_H
#include <math.h>
#include "structs.h"
void mirror_image(PNGImage *image, char axis, int x1, int y1, int x2, int
y2);
#endif //COURSE_WORK_MIRROR_FUNCTION_H
```


parse_functions.c:

```
#include "parse_functions.h"

int *parse_coords(char *coords) {
    int *result = malloc(sizeof(int) * 2);
    char *copy_coords = strdup(coords);
    char *token = strtok(copy_coords, ".");
    result[0] = atoi(token);
    token = strtok(NULL, ".");
    result[1] = atoi(token);
    free(copy_coords);
    return result;
}

int *parse_color(char *color) {
    int *result = malloc(sizeof(int) * 3);
    char *copy_color = strdup(color);
    char *token = strtok(copy_color, ".");
    for (int i = 0; i < 3; i++) {
        result[i] = atoi(token);
        token = strtok(NULL, ".");
    }
    free(copy_color);
    return result;
}
```

parse_functions.h:

```
#ifndef COURSE_WORK_PARSE_FUNCTIONS_H
#define COURSE_WORK_PARSE_FUNCTIONS_H
#include <stdlib.h>
#include <string.h>
int *parse_coords(char *coords);
int *parse_color(char *color);
#endif //COURSE_WORK_PARSE_FUNCTIONS_H
```

png_functions.c:

```
#include "png_functions.h"

void read_png_file(char *file_name, PNGImage *image) {
    int x, y;
    char header[8]; // 8 is the maximum size that can be checked

    FILE *fp = fopen(file_name, "rb");
    if (!fp) {
        fprintf(stderr, "Error: file can not be open\n");
        exit(45);
    }

    fread(header, 1, 8, fp);
    if (png_sig_cmp((png_const_bytep) header, 0, 8)) {
        fprintf(stderr, "Error: File is not recognized as a PNG.\n");
        exit(45);
    }

    image->png_ptr = png_create_read_struct(PNG_LIBPNG_VER_STRING, NULL,
    NULL, NULL);
    if (!image->png_ptr) {
        fprintf(stderr, "Error: png_create_read_struct failed\n");
        exit(45);
    }

    image->info_ptr = png_create_info_struct(image->png_ptr);
    if (!image->info_ptr) {
        png_destroy_read_struct(&image->png_ptr, NULL, NULL);
        exit(0);
    }

    if (setjmp(png_jmpbuf(image->png_ptr))) {
        fprintf(stderr, "Error: error during init_io\n");
        exit(45);
    }

    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);
    if (setjmp(png_jmpbuf(image->png_ptr))) {
        fprintf(stderr, "Похоже в картинке битые пиксели\n");
        exit(45);
    }
    image->row_pointers = (png_bytep *) malloc(sizeof(png_bytep) * image-
```

```

>height);
    for (y = 0; y < image->height; y++)
        image->row_pointers[y] = (png_byte *)
malloc(png_get_rowbytes(image->png_ptr, image->info_ptr));
    png_read_image(image->png_ptr, image->row_pointers);
}
void write_png_file(char *file_name, PNGImage *image) {
    FILE *fp = fopen(file_name, "wb");
    if (!fp) {
        fprintf(stderr, "Can not open file for write\n");
        exit(45);
    }
    image->png_ptr = png_create_write_struct(PNG_LIBPNG_VER_STRING, NULL,
NULL, NULL);
    if (!image->png_ptr) {
        fprintf(stderr, "Error: png_create_write_struct failed\n");
        exit(45);
    }
    image->info_ptr = png_create_info_struct(image->png_ptr);
    if (!image->info_ptr) {
        fprintf(stderr, "Error: png_create_info_struct failed\n");
        exit(45);
    }
    png_init_io(image->png_ptr, fp);
    if (setjmp(png_jmpbuf(image->png_ptr))) {
        fprintf(stderr, "Error: error during writing header\n");
        exit(45);
    }
    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))) {
        fprintf(stderr, "Error: error during writing bytes\n");
        exit(45);
    }
    png_write_image(image->png_ptr, image->row_pointers);
    if (setjmp(png_jmpbuf(image->png_ptr))) {
        fprintf(stderr, "Error: error during end of write\n");
        exit(45);
    }
    png_write_end(image->png_ptr, NULL);
    for (int y = 0; y < image->height; y++) {
        free(image->row_pointers[y]);
    }
    free(image->row_pointers);
    fclose(fp);
}

```

png_functions.h:

```
#ifndef COURSE_WORK_PNG_FUNCTIONS_H
#define COURSE_WORK_PNG_FUNCTIONS_H
#include "structs.h"
#include <stdlib.h>
#include <png.h>
void read_png_file(char *file_name, PNGImage *image);
void write_png_file(char *file_name, PNGImage *image);
#endif //COURSE_WORK_PNG_FUNCTIONS_H
```

structs.h:

```
#ifndef COURSE_WORK_STRUCTS_H
#define COURSE_WORK_STRUCTS_H
#include <png.h>
typedef struct Element {
    char *key;
    char *value;
} Element;
typedef struct PNGImage {
    int width, height; // width and height of the image
    png_byte color_type; // color type of the image
    png_byte bit_depth; // bit depth of the image
    png_structp png_ptr; // pointer to the png struct
    png_infop info_ptr; // pointer to the png info struct
    int number_of_passes;
    png_bytep *row_pointers;
} PNGImage;
typedef struct Point{
    int x, y;
} Point;
#endif //COURSE_WORK_STRUCTS_H
```