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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Обработка изображения

Студент гр. 3344 _____ Клюкин А.В.

Преподаватель _____ Глазунов С.А.

Санкт-Петербург
2024

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студент Клюкин А.В.

Группа 3344

Тема работы: Обработка изображений.

Исходные данные:

- Программу требуется реализовать в виде утилиты, подобной стандартным *linux*-утилитам.
- Программа должна считать *bmp*-файл без сжатия с 24 битами на цвет
- Программа должна сохранить обработанный *bmp*-файл
- Все поля стандартных *BMP* заголовков в выходном файле должны иметь те же значения что и во входном

Содержание пояснительной записки:

- Содержание
- Введение
- Описание варианта работы
- Описание функций программы
- Описание структуры файлов программы
- Описание сборки проекта
- Примеры работы программы
- Заключение

- Список использованных источников

Предполагаемый объем пояснительной записки:

Не менее 30 страниц.

Дата выдачи задания: 18.03.2024

Дата сдачи реферата: 23.05.2024

Дата защиты реферата:
23.05.2024

Студент _____ Клюкин А.В.

Преподаватель _____ Глазунов С.А.

АННОТАЦИЯ

Программа на языке C предназначена для обработки изображений с использованием структуры Rgb для хранения информации о пикселях. В ней реализована утилита, предоставляющая инструменты для фильтрации, рисования и преобразования изображений. Фильтры позволяют изменять значения RGB-компонент цвета. Для рисования используется функция создания квадрата с диагоналями, определяемыми параметрами. Преобразование включает в себя функцию поворота фрагмента изо. Операции выполняются с использованием структуры Element, которая действует как "словарь" для хранения аргументов вызова функций. Результатом работы программы является обработанное изображение с учетом выполненных операций.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель проекта — изучение изображений формата BMP и реализация утилиты на языке C для работы с этим форматом. Задачи включают изучение структуры файла BMP, получение параметров изображения, обработку массива пикселей в соответствии с заданием, обработку крайних случаев, таких как отсутствие файла или неверный формат, и сохранение итогового изображения в новый файл. Методы будут включать в себя реализацию функций для чтения и записи файлов BMP, а также функций для обработки изображений.

1. ОПИСАНИЕ ВАРИАНТА РАБОТЫ

Программа должна иметь следующую функции по обработке изображений:

Рисование квадрата с диагоналями. Флаг для выполнения данной операции: –squared_lines. Квадрат определяется:
Координатами левого верхнего угла. Флаг –left_up, значение задаётся в формате left.up, где left – координата по x, up – координата по y. Размером стороны. Флаг –side_size. 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). Может быть залит или нет (диагонали располагаются “поверх” заливки). 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)

Для создания фильтра RGB-компоненты с заданными параметрами:
1. Флаг --rgbfilter для указания операции фильтрации RGB-компонент.
2. Флаг --component_name для выбора компоненты, которую нужно изменить (red, green или blue).
3. Флаг --component_value для установки значения выбранной компоненты в диапазоне от 0 до 255.

Для выполнения операции поворота изображения (части):
1. Флаг --rotate для указания операции поворота изображения.
2. Флаг --left_up для определения координат левого верхнего угла области. Значение задается в формате left.up, где left - координата по x, up - координата по y.
3. Флаг --right_down для определения координат правого нижнего угла области.

Значение задается в формате right.down, где right - координата по x, down - координата по y.

4. Флаг --angle для указания угла поворота. Возможные значения: 90, 180, 270.

2. ОПИСАНИЕ ПРОГРАММЫ

2.1. Описание функций программы

Функции и их краткое описание:

- int main(int argc, char** argv)

  Функция с первичной обработкой входных данных, которая передает все в функцию

- void run(Element *dict, int len_dict) - функция выполняющая основную полученную опцию, которая определяется через

- char *find_main_option(Element *dict, int len_dict, int *input_flag, int *output_flag, int *info_flag, int *help_flag, int *squared_lines_flag,int *side_size_flag, int *left_up_flag, int *right_down_flag, int *rgbfilter_flag, int *component_name_flag, int *component_value_flag, int *thickness_flag, int *color_flag, int *fill_flag, int*fill_color_flag, int *rotate_flag, int *angle_flag) - длинная функция, которая в зависимости от полученных флагов - делает проверку на корректность и возвращает основную опцию.

- void check_flags(Element *dict, int len_dict, int *input_flag, int *output_flag, int *info_flag, int *help_flag, int *squared_lines_flag, int *side_size_flag, int *left_up_flag, int *right_down_flag, int *rgbfilter_flag, int *component_name_flag, int *component_value_flag, int *thickness_flag, int *color_flag, int *fill_flag, int *fill_color_flag, int *rotate_flag, int *angle_flag) - Работает как переключатель. Проверяет в словаре наличие аргумента и в этом случае приравнивает флаг к 1.

- int check_extra_option(Element *dict, int len_dict) -
  Функция почти как предыдущая, но необходимо учитывать ограничения по количеству лишних (основных) функций.

- char *find_value(Element *dict, int len, char *key) - ищет значение в “словаре” по ключу.

  Функции на check - являются проверяющими соответствующие значения.
  Функции parse - разделяют аргументы на составляющие.

- Rgb **read_bmp(char file_name[], BitmapFileHeader *bmfh, BitmapInfoHeader *bmif) – функция считывания изображения.
- void write_bmp(char file_name[], Rgb **arr, int H, int W, BitmapFileHeader bmfh, BitmapInfoHeader bmif) – функция записи.
- int check_bmp_signature(FILE *file) – функция проверки файла на корректность формата.
- void draw_circle(Rgb **arr, int H, int W, int dot_x, int dot_y, int thickness, int* color) – функция рисования закрашенной окружности в точке с заданной толщиной.
- Rgb **change_red_chanel(Rgb **arr, int H, int W, int r) - изменяет значение красной компоненты пикселей. Аналогично работают с green и blue.
- void draw_line(Rgb **arr, int H, int W, int x0, int y0, int x1, int y1, int thickness, int *color, int flag) – функция для рисования вертикальной или горизонтальной линии.
- void fill_sq(Rgb **arr, int start_x, int start_y, int lenght_line, int *colors, int H, int W) - функция для закрашивания определенной области заданным цветом.
- void draw_diags(Rgb **arr, int H, int W, int start_x, int start_y, int

end_x, int end_y, int thickness, int *color) - рисование наклонной между двумя точками.

- void draw_diags(Rgb **arr, int H, int W, int start_x, int start_y, int end_x, int end_y, int thickness, int *color) - функция для рисования квадрата с диагоналями, на основе предыдущих функций.
- Rgb** turned_copy(Rgb **arr, int H, int W, int start_x, int start_y, int end_x, int end_y, int angle) - копирование области с определенной стороны, в зависимости от угла
- void replace_part(Rgb **arr, Rgb **arr_t, int start_x, int start_y, int end_x, int end_y, int H, int W) - вставка скопированной области изображения
- void turn(Rgb **arr, int H, int W, int start_x, int start_y, int end_x, int end_y, int angle) - определение центра вращения.

2.2. Описание структуры файлов программы

Программа содержит следующую структуру:

- *Makefile*: Файл для автоматизации процесса компиляции и сборки программы.
- main.cpp — файл, в котором происходит вызов остальных функций для обработки изображения
- structures.h - файл со всеми структурами
- files_action.c - файл с функциями для работы с записью, чтением и выводом информации об изображении
- actions.c - файл с функциями для обработки изображения
- input_hell.c - файл для обработки входных данных и выборе цели
- checkers.c - файл с функциями проверки на корректность отдельных значений

3. ПРИМЕРЫ РАБОТЫ ПРОГРАММЫ

Пример 1: Функция rotate

Ввод	Вывод
./cw --output ./output.bmp --input ./f.bmp --right_down 355.272 --rotate --angle 270 --left_up 241.52	Правильно измененная картинка

Пример 2: Функция squared_lines

Ввод	Вывод
./cw --input ./last.bmp --thickness 25 --side_size 281 --color 35.79.218 --fill_color 224.186.80 --squared_lines --left_up 66.100 --fill --output ./output.bmp	Правильно измененная картинка

Пример 3: Функция rgbfilter

Ввод	Вывод
./cw --rgbfilter --component_name green --component_value 255 --output output.bmp input.bmp	Правильно измененная картинка

Пример 5: Вывод информации об изображении.

Ввод	Вывод
./cw --info input.bmp	signature: 4d42 (19778) filesize: c82b8 (819896) reserved1: 0 (0) reserved2: 0 (0) pixelArrOffset: 36 (54) headerSize: 28 (40) width: 280 (640) height: 1ab (427)

	planes: 1 (1) bitsPerPixel: 18 (24) compression: 0 (0) imageSize: c8282 (819842) xPixelsPerMeter: b12 (2834) yPixelsPerMeter: b12 (2834) colorsInColorTable: 0 (0) importantColorCount: 0 (0)

Пример 6: Проверка обработки ошибок.

Ввод	Вывод
./cw --output ./output.bmp --input ./f.bmp --right_down 355.272 --rotate --angle 270 --left_up 241.-52	Coords are not correct

ЗАКЛЮЧЕНИЕ

Была успешно реализована программа на языке C для обработки изображений в формате BMP. Программа выполняет поставленные задачи, включая чтение и запись изображений, фильтрацию, рисование и поворот фрагмента изображения. Полученные результаты подтверждают успешное достижение поставленной цели. В ходе выполнения работы были приобретены навыки работы с изображениями, использования структур данных.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Базовые сведения к выполнению курсовой и лабораторных работ по дисциплине «программирование». Второй семестр: учеб.-метод. пособие др. СПб.: Изд-во СПбГЭТУ «ЛЭТИ», 2024. 36 с.
2. Geeksforgeeks. URL: https://www.geeksforgeeks.org (дата обращения: 25.04.2024).

ПРИЛОЖЕНИЕ А
КОД ПРОГРАММЫ

main.c

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

#include "structures.h"
#include "files_action.h"
#include "actions.h"
#include "input_hell.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,
1},
            {"help",             no_argument,           NULL,
'h'},
            {"squared_lines",    no_argument,           NULL,
0},
            {"side_size",        required_argument,     NULL,
0},
            {"left_up",          required_argument,     NULL,
0},
            {"right_down",       required_argument,     NULL,
0},
            {"rgbfilter",        no_argument,           NULL,
0},
            {"component_name",   required_argument,     NULL,
0},
            {"component_value",  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},
            {"rotate",           no_argument,           NULL,
0},
```

```
            {"angle",           required_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);
        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;
}
```

Makefile:

```
all: main.o files_action.o actions.o input_hell.o checkers.o
     gcc  main.o files_action.o actions.o input_hell.o
checkers.o -o cw -lm

main.o: main.c
     gcc -c main.c
files_action.o: files_action.c files_action.h
     gcc -c files_action.c
actions.o: actions.c actions.h
     gcc -c actions.c
```

```
input_hell.o: input_hell.c input_hell.h
	gcc -c input_hell.c
checkers.o: checkers.c checkers.h
	gcc -c  checkers.c
clean:
	rm *.o
```

structures.h:

```
#ifndef COURSE_WORK_STRUCTS_H
#define COURSE_WORK_STRUCTS_H

typedef struct Element {
    char *key;
    char *value;
} Element;

typedef struct {
    int x, y;
} Point;

#pragma pack(push, 1)


typedef struct
{
    unsigned short signature;
    unsigned int filesize;
    unsigned short reserved1;
    unsigned short reserved2;
    unsigned int pixelArrOffset;
} BitmapFileHeader;

typedef struct
{
    unsigned int headerSize;
    unsigned int width;
    unsigned int height;
    unsigned short planes;
    unsigned short bitsPerPixel;
    unsigned int compression;
    unsigned int imageSize;
    unsigned int xPixelsPerMeter;
    unsigned int yPixelsPerMeter;
    unsigned int colorsInColorTable;
    unsigned int importantColorCount;
} BitmapInfoHeader;

typedef struct
{
    unsigned char b;
    unsigned char g;
    unsigned char r;
} Rgb;


#pragma pack(pop)
```

```
#endif
```

files_actions.c:

```
#include "files_action.h"

unsigned int padding(unsigned int w){
    unsigned int padding = (w * sizeof(Rgb)) % 4;
    if(padding) padding = 4 - padding;
    return padding;
}

unsigned int row_len(unsigned int w){
    return w * sizeof(Rgb) + padding(w);
}

int check_bmp_color_depth(FILE *file) {
    fseek(file, 28, SEEK_SET); // Переходим к байту, содержащему
информацию о битовой глубине цвета

    unsigned short color_depth;
    fread(&color_depth, sizeof(unsigned short), 1, file);

    // Проверяем битовую глубину цвета
    if (color_depth == 24) {
        return 1; // Битовая глубина 24 бита
    } else {
        fprintf(stderr, "Wrong color format\n");
        exit(41);
    }
}

int check_bmp_signature(FILE *file) {
    char signature[2];
    fread(signature, sizeof(char), 2, file);
    if (signature[0] == 'B' && signature[1] == 'M') {
        return 1;
    } else {
        fprintf(stderr, "Wrong format\n");
        exit(41);
    }
}

Rgb **read_bmp(char file_name[], BitmapFileHeader *bmfh,
BitmapInfoHeader *bmif)
{
    FILE *test = fopen(file_name, "rb");
    if (check_bmp_signature(test)){
        if (check_bmp_color_depth(test))
        {
            fclose(test);
        }
    }
    FILE *f = fopen(file_name, "rb");
    fread(bmfh, 1, sizeof(BitmapFileHeader), f);
    fread(bmif, 1, sizeof(BitmapInfoHeader), f);
    unsigned int H = bmif->height;
    unsigned int W = bmif->width;
```

```
    Rgb **arr = malloc(H*sizeof(Rgb*));
    for(int i=0; i<H;i++){
        arr[i] = malloc(row_len(W));
        fread(arr[i], 1, row_len(W), f);
    }
    fclose(f);
    return arr;
}

void write_bmp(char file_name[], Rgb **arr, int H, int W,
BitmapFileHeader bmfh, BitmapInfoHeader bmif)
{
    FILE *ff = fopen(file_name, "wb");
    bmif.height = H;
    bmif.width = W;
    fwrite(&bmfh, 1, sizeof(BitmapFileHeader), ff);
    fwrite(&bmif, 1, sizeof(BitmapInfoHeader), ff);
    unsigned int w = row_len(W);
    for(int i=0; i<H; i++){
        fwrite(arr[i], 1, w, ff);
    }
    fclose(ff);
}

void print_file_header(BitmapFileHeader header){
    printf("signature:\t%x (%hu)\n", header.signature,
header.signature);
    printf("filesize:\t%x (%u)\n", header.filesize,
header.filesize);
    printf("reserved1:\t%x (%hu)\n", header.reserved1,
header.reserved1);
    printf("reserved2:\t%x (%hu)\n", header.reserved2,
header.reserved2);
    printf("pixelArrOffset:\t%x (%u)\n", header.pixelArrOffset,
header.pixelArrOffset);
}

void print_info_header(BitmapInfoHeader header){
    printf("headerSize:\t%x (%u)\n", header.headerSize,
header.headerSize);
    printf("width: \t%x (%u)\n", header.width, header.width);
    printf("height: \t%x (%u)\n", header.height, header.height);
    printf("planes: \t%x (%hu)\n", header.planes,
header.planes);
    printf("bitsPerPixel:\t%x (%hu)\n", header.bitsPerPixel,
header.bitsPerPixel);
    printf("compression:\t%x (%u)\n", header.compression,
header.compression);
    printf("imageSize:\t%x (%u)\n", header.imageSize,
header.imageSize);
    printf("xPixelsPerMeter:\t%x (%u)\n",
header.xPixelsPerMeter, header.xPixelsPerMeter);
    printf("yPixelsPerMeter:\t%x (%u)\n",
header.yPixelsPerMeter, header.yPixelsPerMeter);
    printf("colorsInColorTable:\t%x (%u)\n",
header.colorsInColorTable, header.colorsInColorTable);
    printf("importantColorCount:\t%x (%u)\n",
```

```
            header.importantColorCount, header.importantColorCount);
            }
```

files_action.h:

```
            #ifndef COURSE_WORK_FILE_H
            #define COURSE_WORK_FILE_H
            #include <stdio.h>
            #include <stdlib.h>
            #include "structures.h"

            unsigned int padding(unsigned int w);
            unsigned int row_len(unsigned int w);
            int check_bmp_color_depth(FILE *file);
            int check_bmp_signature(FILE *file);
            Rgb **read_bmp(char file_name[], BitmapFileHeader *bmfh,
            BitmapInfoHeader *bmif);
            void write_bmp(char file_name[], Rgb **arr, int H, int W,
            BitmapFileHeader bmfh, BitmapInfoHeader bmif);
            void print_file_header(BitmapFileHeader header);
            void print_info_header(BitmapInfoHeader header);

            #endif
```

actions.c:

```
#include "actions.h"

Rgb **change_red_chanel(Rgb **arr, int H, int W, int r){
    for (int i = 0; i < H; i++)
    {
        for (int j = 0; j < W; j++)
        {
            arr[i][j].r = r;
        }
    }
    return arr;
}

Rgb **change_green_chanel(Rgb **arr, int H, int W, int g){
    for (int i = 0; i < H; i++)
    {
        for (int j = 0; j < W; j++)
        {
            arr[i][j].g = g;
        }
    }
    return arr;
}

Rgb **change_blue_chanel(Rgb **arr, int H, int W, int b){
    for (int i = 0; i < H; i++)
    {
        for (int j = 0; j < W; j++)
        {
            arr[i][j].b = b;
        }
    }
```

```
    return arr;
}


void swap_int(int *a, int *b){
    int t = *a;
    *a = *b;
    *b = t;
}


void draw_circle(Rgb **arr, int H, int W, int dot_x, int dot_y, int
thickness, int* color) {

    if (dot_x < 0 || dot_x > W || dot_y < 0 || dot_y > H || thickness <= 0)
{
        // fprintf(stderr, "Coords are not correct\n");
        // exit(41);
        return;
    }

    int radius = (thickness % 2 == 0) ? thickness / 2 : (thickness - 1) /
2;

    for (int y = -radius; y <= radius; y++) {
        for (int x = -radius; x <= radius; x++) {
            if (x*x + y*y <= radius*radius) {
                if (dot_y + y >= 0 && dot_y + y < H && dot_x + x >= 0 &&
dot_x + x < W) {
                    arr[dot_y + y][dot_x + x].r = *(color);
                    arr[dot_y + y][dot_x + x].g = *(color + 1);
                    arr[dot_y + y][dot_x + x].b = *(color + 2);
                }
            }
        }
    }
}




void draw_line(Rgb **arr, int H, int W, int x0, int y0, int x1, int y1, int
thickness, int *color, int flag){
    y0=H-y0;
    y1=H-y1;
    if (x0 < 0 || y0 > H || x1 < 0 || y1 > H || thickness <= 0){
        fprintf(stderr, "Coords are not correct\n");
        //printf("\n%d %d %d %d %d %d\n", x0, y0, x1, y1, H, W);
        exit(41);
        return;
    }
    //printf("Y\n");

    // if(x0>W){
    //     x0=W;
    // }
    // if(y0>H){
    //     y0=H;
```

```
    // }

    //printf("%d %d %d %d H-%d W-%d\n", x0, x1, y0, y1, H, W);
    if (x0 > W || y0 > H){
        return;
    }
    // вертикальная линия
    if(x1>W){
        x1=W;
    }
    if(y1>H){
        y1=H;
    }

    int side = 1;
    // if(thickness % 2 != 0){
    //     thickness = thickness-1;
    // }
    // if (thickness == 0){
    //     thickness = 2;
    // }
    if(y0 < 0){
        y0=0;
    }
    // if(y1 < 0){
    //     y1 = 0;
    // }
    //printf("%d %d %d %d H-%d W-%d\n", x0, x1, y0, y1, H, W);
    if (x0 == x1){
        if (y0 > y1){
            swap_int(&y0, &y1);
        }
        for (int y = y0; y <= y1; y++){
            draw_circle(arr, H, W, x0, y, thickness, color);
        }
        // горизонтальная линия
    }
    else if (y0 == y1){
        if (x0 > x1){
            swap_int(&x0, &x1);
        }
        for (int x = x0; x <= x1; x++){
            draw_circle(arr, H, W, x, y0, thickness, color);
        }
    }
}

void fill_sq(Rgb **arr, int start_x, int start_y, int lenght_line, int
*colors, int H, int W){
    //printf("%d %d", start_y, start_y-lenght_line);
    for (int i = start_x; i < start_x+lenght_line; i++){
        for (int j = start_y; j > start_y-lenght_line; j--){
            if(i >= 0 && j < H && i < W && j>=0){
                arr[j][i].r = colors[0];
                arr[j][i].g = colors[1];
                arr[j][i].b = colors[2];
            }
        }
```

```
    }
}


void draw_diags(Rgb **arr, int H, int W, int start_x, int start_y, int
end_x, int end_y, int thickness, int *color){

    start_y = H-start_y;
    end_y = H-end_y;

    if(start_y>H){
        return;
    }

    // if(end_x > W){
    //     end_x = W;
    // }
    // if(end_y > H){
    //     end_y = H;
    // }

    int dx = abs(end_x - start_x);
    int dy = abs(end_y - start_y);
    int sx = start_x < end_x ? 1 : -1;
    int sy = start_y < end_y ? 1 : -1;
    int err = (dx > dy ? dx : -dy) / 2;
    int e2;
    int x = start_x;
    int y = start_y;
    //printf("\n%d %d\n", start_y, end_y);

    //somwhere here maybe i should change
    while (x != end_x || y != end_y){
        if (x >= 0 && x < W && y >= 0 && y < H){
            // arr[y][x].r = color[0];
            // arr[y][x].g = color[1];
            // arr[y][x].b = color[2];

            draw_circle(arr, H, W, x, y, thickness, color);

        }

        e2 = err;
        if (e2 > -dx){
            err -= dy;
            x += sx;
        }
        if (e2 < dy){
            err += dx;
            y += sy;
        }
    }
}

void draw_sq(Rgb **arr, int start_x, int start_y, int lenght_line, int H,
int W, int thickness, int *color, int fill, int *fill_color, int thick){
    if(fill){
        fill_sq(arr, start_x, H-start_y, lenght_line, fill_color, H, W);
```

```
    }
    draw_line(arr, H, W, start_x, start_y, start_x + lenght_line, start_y,
thickness, color, 0);
    draw_line(arr, H, W, start_x, start_y, start_x, start_y + lenght_line,
thickness, color, 1);

    draw_line(arr, H, W, start_x, start_y + lenght_line, start_x +
lenght_line, start_y + lenght_line, thickness, color, 1);
    draw_line(arr, H, W, start_x + lenght_line, start_y, start_x +
lenght_line, start_y + lenght_line, thickness, color, 0);

    draw_diags(arr, H, W, start_x, start_y, start_x+lenght_line,
start_y+lenght_line, thick, color);
    draw_diags(arr, H, W, start_x+lenght_line, start_y, start_x,
start_y+lenght_line, thick, color);

}


Rgb** turned_copy(Rgb **arr, int H, int W, int start_x, int start_y, int
end_x, int end_y, int angle){
    int x = 0;
    int y = 0;

    int Verticals = 0; // how much lines
    int Horizontal = 0; // how long 1 line in horizontal

    if(angle == 90){
        Verticals = end_x-start_x+3; // turned matrix reading at right-up
position to right-down pos
        Horizontal = end_y-start_y+3;
    }else if (angle == 180){
        Verticals = end_y-start_y+3; // turned M reading at right-down pos
to left-down pos
        Horizontal = end_x-start_x+3;
    }else if (angle = 270){
        Verticals = end_x-start_x+3;
        Horizontal = end_y-start_y+3;
    }


    Rgb **arr_t = (Rgb **)malloc(Verticals* sizeof(Rgb *)); // make turned
matrix
    for (int i = 0; i < Verticals; i++) {
        arr_t[i] = (Rgb *)malloc(Horizontal * sizeof(Rgb));
    }
    //printf("%d %d %d %d,  H=%d W=%d\n",  start_x, start_y, end_x, end_y,
H, W);

    start_y = H-start_y; // translate y chords, because it's inverted
    end_y = H-end_y;
    //printf("%d %d %d %d  Vert=%d Horiz=%d\n",  start_x, start_y, end_x,
end_y, Verticals, Horizontal);
    //exit(41);

    switch (angle){
    case 90:               //turned matrix reading at right-up position to
right-down pos
```

```
        for(int i=end_x-1; i>=start_x-1;i--){
            for(int j=start_y-1;j>=end_y-1;j--){
                if(j < H && j>=0 && x < W && x >= 0){
                    arr_t[y][x] = arr[j][i];
                }
                //printf("%d %d, j=%d, i=%d, x=%d, y=%d, H=%d, W=%d\n",
arr_t[y][x].r, arr[j][i].r, j, i, x, y, H, W);
                x=x+1;
            }
            y=y+1;
            x=0;
        }

        break;
    case 180:              // turned matrix reading at right-down to left-
down pos
        for(int i=end_x-1; i>=start_x-1;i--){ // -1 fish test
            for(int j=end_y;j<=start_y;j++){
                //printf("%d %d, j=%d, i=%d, x=%d, y=%d, H=%d, W=%d\n",
arr[j][i].r, arr[j][i].r, j, i, x, y, H, W);
                arr_t[y][x] = arr[j][i];
                y=y+1;
            }
            x=x+1;
            y=0;
        }
        break;
    case 270: // left-down to left-up

        for(int i=start_x; i<=end_x;i++){
            for(int j=end_y;j<=start_y;j++){
                arr_t[y][x] = arr[j][i];
                x=x+1;
            }
            y=y+1;
            x=0;
        }
        //printf("End");
        break;

    default:
        break;
    }
    return arr_t;
}

void replace_part(Rgb **arr, Rgb **arr_t, int start_x, int start_y, int
end_x, int end_y, int H, int W){
    int x = 0;
    int y = 0;// ДОБАВИТЬ ОГРАНИЧЕНИЯ ПО end_y


    printf("\nx1-%d y1-%d x2-%d y2-%d  H=%d W=%d   - delta = %d\n\n",
start_x+x, H-start_y-y-1, end_x-x+1, H-end_y+y, H, W, x);

    if(end_y >= H){
        end_y = H;
```

```
    }
    if(start_y < 0){
        start_y = 0;
    }
    // if(end_x > W){
    //     end_x = W;
    // }
    printf("\nx1-%d y1-%d x2-%d y2-%d  H=%d W=%d   - deltass = %d\n\n",
start_x+x, start_y, end_x-x+1, H-end_y+y, H, W, x);
    //return;
        for(int i=start_x; i<=end_x+1;i++){
            for(int j=H-start_y-1; j>=H-end_y; j--){
                if(i >= 0 && j < H && i < W && j>=0){
                    //printf("%d %d    %d %d\n", y, x, j, i);
                    arr[j][i] = arr_t[y][x];
                    //printf("D\n");
                }

                // arr[j][i].r = x;
                // arr[j][i].g = y;
                // arr[j][i].b = 0;
                y = y+1;
            }

            x = x+1;
            //printf("%d %d \n", x, y);
            y = 0;
        }
        //printf("\nx1-%d y1-%d x2-%d y2-%d  H=%d W=%d   - delta = %d\n\n",
start_x, H-start_y-1, end_x+1, H-end_y, H, W, x);

}

void turn(Rgb **arr, int H, int W, int start_x, int start_y, int end_x, int
end_y, int angle){ //arr[y][x], where x < W, x>=0, y >= 0, y < H

    //printf("%d %d %d %d  H=%d W=%d   - arr = %d",  start_x, start_y,
end_x, end_y, H, W, arr[0][680].r);
    // arr[426][0].r = 0;
    // arr[426][0].g = 0;
    // arr[426][0].b = 255;

    // exit(41);

    // H = 471;
    // W = 500;

    if(start_x > end_x){
        swap_int(&start_x, &end_x);
        swap_int(&start_y, &end_y);
        //printf("%d %d %d %d\n", start_x, start_y, end_x, end_y);
        if(start_y > end_y){
            fprintf(stderr, "Wrong dots\n");
            exit(41);
        }
    }
    if(start_y > end_y){
        swap_int(&start_x, &end_x);
```

```
        swap_int(&start_y, &end_y);
        if(start_x > end_x){
            fprintf(stderr, "Wrong dots\n");
            exit(41);

        }
    }
                        // swap coords

    Rgb **arr_t = turned_copy(arr, H, W, start_x, start_y, end_x, end_y,
angle);
    int sq_flag = (end_x-start_x)-(end_y-start_y);

    int lenght_x = end_x-start_x+1; // +1 becouse including end's dot
    int lenght_y = end_y-start_y+1;



    switch (angle)
    {
    case 90:

        if(sq_flag == 0){
            replace_part(arr, arr_t, start_x, start_y-1, end_x-1, end_y, H,
W); //done
            break;
        }
        if((lenght_x-lenght_y)%2==0){
            // printf("OFF");
            int x = (lenght_x-lenght_y)/2;
            int y = (lenght_x-lenght_y)/2;
            replace_part(arr, arr_t, start_x+x, start_y-y, end_x-x-2,
end_y+y, H, W); //done over
            break;
        }
        if((lenght_x-lenght_y)%2==1){
            // printf("YE");
            int x = abs((lenght_x-lenght_y-1)/2);
            int y = abs((lenght_x-lenght_y-1)/2);
            replace_part(arr, arr_t, start_x+x, start_y-y-1, end_x-x-2,
end_y+y, H, W); // done BUT start_X + 2
            break;
        }
        if((lenght_x-lenght_y)%2==-1){
            // printf("L");
            int x = abs((lenght_x-lenght_y+1)/2);
            int y = abs((lenght_x-lenght_y+1)/2);
            replace_part(arr, arr_t, start_x-x, start_y+y, end_x+x, end_y-
y, H, W); //done
            break;
        }


    case 180:
        replace_part(arr, arr_t, start_x, start_y, end_x-1, end_y+1, H, W);
// x without change  НЕ ДОЛЖНО БЫТЬ ПРОБЛЕМ
        break;
```

```
    case 270:
        if(sq_flag == 0){
            replace_part(arr, arr_t, start_x, start_y-1, end_x-1, end_y, H,
W);
            break;
        }
        if((lenght_x-lenght_y)%2==0){
            //printf("%d %d-------\n", lenght_x, lenght_y);
            int x = ((lenght_x-lenght_y))/2;
            int y = ((lenght_x-lenght_y))/2;
            //printf("%d %d %d %d  H=%d W=%d   - delta = %d\n",  start_x+x,
start_y-y, end_x-x, end_y+y, H, W, x);
            //printf("S");

            replace_part(arr, arr_t, start_x+x, start_y-y, end_x-x-2,
end_y+y, H, W);
            break;
        }
        if((lenght_x-lenght_y)%2==1){

            int x = abs((lenght_x-lenght_y-1)/2);
            int y = abs((lenght_x-lenght_y-1)/2);
            replace_part(arr, arr_t, start_x+x, start_y-y, end_x-x-3,
end_y+y, H, W); //done over
            break;
        }

        if((lenght_x-lenght_y)%2==-1){

            int x = abs((lenght_x-lenght_y+1)/2);
            int y = abs((lenght_x-lenght_y+1)/2);
            replace_part(arr, arr_t, start_x-x-1, start_y+y+1, end_x+x-3,
end_y-y, H, W);
            break;
        }
    default:
        break;
    }
}
```

actions.h:

```
#ifndef COURSE_WORK_DRAWING_LINE_FUNCTIONS_H
#define COURSE_WORK_DRAWING_LINE_FUNCTIONS_H
#include <stdio.h>
#include <stdlib.h>
#include "structures.h"

void draw_circle(Rgb **arr, int H, int W, int dot_x, int dot_y, int
thickness, int* color);

Rgb **change_red_chanel(Rgb **arr, int H, int W, int r);
Rgb **change_green_chanel(Rgb **arr, int H, int W, int g);
Rgb **change_blue_chanel(Rgb **arr, int H, int W, int b);
void swap_int(int *a, int *b);
void draw_line(Rgb **arr, int H, int W, int x0, int y0, int x1, int y1, int
thickness, int *color, int flag);
void fill_sq(Rgb **arr, int start_x, int start_y, int lenght_line, int
```

```
*colors, int H, int W);
void draw_diags(Rgb **arr, int H, int W, int start_x, int start_y, int
end_x, int end_y, int thickness, int *color);
void draw_sq(Rgb **arr, int start_x, int start_y, int lenght_line, int H,
int W, int thickness, int *color, int fill, int *fill_color, int thick);
Rgb** turned_copy(Rgb **arr, int H, int W, int start_x, int start_y, int
end_x, int end_y, int angle);
void replace_part(Rgb **arr, Rgb **arr_t, int start_x, int start_y, int
end_x, int end_y, int H, int W);
void turn(Rgb **arr, int H, int W, int start_x, int start_y, int end_x, int
end_y, int angle);

#endif
```

input_hell.c:

```
#include "input_hell.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[] = {"squared_lines", "rgbfilter", "rotate", "info",
"input", "output"};
    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++;
            }
            if (strcmp(dict[i].key, main_options[3]) == 0) {
                count_main_options = -1;
                return 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 *squared_lines_flag, int
*side_size_flag, int *left_up_flag, int *right_down_flag, int
*rgbfilter_flag, int *component_name_flag, int *component_value_flag, int
*thickness_flag, int *color_flag, int *fill_flag, int *fill_color_flag, int
*rotate_flag, int *angle_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("squared_lines", dict[i].key) == 0) {
            *squared_lines_flag = 1;
        } else if (strcmp("side_size", dict[i].key) == 0) {
            *side_size_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("rgbfilter", dict[i].key) == 0) {
            *rgbfilter_flag = 1;
        } else if (strcmp("component_name", dict[i].key) == 0) {
            *component_name_flag = 1;
        } else if (strcmp("component_value", dict[i].key) == 0) {
            *component_value_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("rotate", dict[i].key) == 0) {
            *rotate_flag = 1;
        } else if (strcmp("angle", dict[i].key) == 0) {
            *angle_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 *
squared_lines_flag,int *side_size_flag, int *left_up_flag, int
*right_down_flag, int *rgbfilter_flag, int
                       *component_name_flag, int *component_value_flag, int
*thickness_flag, int *color_flag, int *fill_flag, int
                       *fill_color_flag, int *rotate_flag, int *angle_flag)
{
    check_flags(dict, len_dict, input_flag, output_flag, info_flag,
help_flag, squared_lines_flag, side_size_flag,
                left_up_flag, right_down_flag, rgbfilter_flag,
component_name_flag, component_value_flag,
                thickness_flag, color_flag, fill_flag, fill_color_flag,
rotate_flag, angle_flag);
    int count_main_options = check_extra_option(dict, len_dict);
    // printf("%d", *output_flag);
```

```
    if (count_main_options == 3) {
        // переделать проверки
        if ((*squared_lines_flag)==1) {
            if ((*left_up_flag)==1 && (*side_size_flag==1) &&
(*thickness_flag)==1 && (*color_flag)==1 && (*output_flag)==1 &&
(*input_flag) ==1){
                check_one_coords(find_value(dict, len_dict, "left_up"));
                check_size(find_value(dict, len_dict, "side_size"));
//maybe i need to check >0
                check_color(find_value(dict, len_dict, "color"));
                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);
                }
                check_output_and_input_match(find_value(dict, len_dict,
"input"), find_value(dict, len_dict, "output"));
                return "squared_lines";
            } else {
                fprintf(stderr, "Недостаточное количество флагов для
функции squared_lines\n");
                exit(41);
            }
        } else if ((*rgbfilter_flag)==1) {
            if ((*component_name_flag)==1 && (*component_value_flag)==1 &&
(*output_flag)==1 && (*input_flag)==1){

                check_component_name(find_value(dict, len_dict,
"component_name"));
                check_size(find_value(dict, len_dict, "component_value"));
                check_output_and_input_match(find_value(dict, len_dict,
"input"), find_value(dict, len_dict, "output"));
                return "rgbfilter";
            } else {
                fprintf(stderr, "Недостаточное количество флагов для
функции rgbfilter_flag\n");
                exit(41);
            }
        } else if ((*rotate_flag)==1) {
            if ((*left_up_flag)==1 && (*right_down_flag)==1 &&
(*angle_flag)==1 && (*output_flag)==1 && (*input_flag)==1){
                check_one_coords(find_value(dict, len_dict, "left_up"));
                check_one_coords(find_value(dict, len_dict, "right_down"));
                check_angle(find_value(dict, len_dict, "angle"));
                check_output_and_input_match(find_value(dict, len_dict,
"input"), find_value(dict, len_dict, "output"));
                return "rotate";
            } else {
                fprintf(stderr, "Недостаточное количество флагов для
функции rotate\n");
                exit(41);
            }
        } else if ((*info_flag)==1) {
            return "info";
        }
    } else if (count_main_options > 3) {
```

```
        fprintf(stderr, "Можно выполнить только 1 основную функцию, а не
несколько\n");
        exit(41);
    } else if(count_main_options == -1 && (*input_flag)==1){return "info";}
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 squared_lines_flag = 0;
    int side_size_flag = 0;
    int left_up_flag = 0;
    int right_down_flag = 0;
    int rgbfilter_flag = 0;
    int component_name_flag = 0;
    int component_value_flag = 0;
    int thickness_flag = 0;
    int color_flag = 0;
    int fill_flag = 0;
    int fill_color_flag = 0;
    int rotate_flag = 0;
    int angle_flag = 0;

    char *main_option = find_main_option(dict, len_dict, &input_flag,
&output_flag, &info_flag, &help_flag, &squared_lines_flag, &side_size_flag,
                                         &left_up_flag, &right_down_flag,
&rgbfilter_flag, &component_name_flag, &component_value_flag,
&thickness_flag, &color_flag,
                                         &fill_flag, &fill_color_flag,
&rotate_flag, &angle_flag);

    //printf("%s", main_option);
    if (strcmp(main_option, "squared_lines") == 0) {
        //printf("SQUARED\n");

        BitmapFileHeader bmfh;
        BitmapInfoHeader bmif;
        Rgb **arr;
        arr = read_bmp(find_value(dict, len_dict, "input"), &bmfh, &bmif);
        //print_info_header(bmif);
        int H = bmif.height;
        int W = bmif.width;

        int *coords_all = parse_coords(find_value(dict, len_dict,
"left_up"));
        int start_x = coords_all[0];
        int start_y = coords_all[1];
        int len_s = atoi(find_value(dict, len_dict, "side_size"));
        int th = atoi(find_value(dict, len_dict, "thickness"));
        int *col = parse_color(find_value(dict, len_dict, "color"));
        // if(start_x+len_s > W){
```

```
        //     fprintf(stderr, "wrong size\n");
        //     exit(41);
        // }
        // if(start_y + len_s > H){
        //     fprintf(stderr, "wrong size\n");
        //     exit(41);
        // }
        if(fill_flag){
            int *col_f = parse_color(find_value(dict, len_dict,
"fill_color"));
            draw_sq(arr, start_x, start_y, len_s, H, W, th, col, 1, col_f,
th-1); // check coords 0
        }else{
            // printf("%d-x\n %d-y\n %d-len\n %d-th\n", start_x,
start_y,len_s, th);
            int col_f[3] = {0, 0, 0};
            draw_sq(arr, start_x, start_y, len_s, H, W, th, col, 0, col_f,
th-1);
        }
        write_bmp(find_value(dict, len_dict, "output"), arr, H, W, bmfh,
bmif);

        for (int i = 0; i < H; i++){
            free(arr[i]);
        }
        free(arr);

    }
    else if(strcmp(main_option,"rgbfilter")==0){
        //printf("FILTERS");
        BitmapFileHeader bmfh;
        BitmapInfoHeader bmif;
        Rgb **arr;
        arr = read_bmp(find_value(dict, len_dict, "input"), &bmfh, &bmif);
        //print_info_header(bmif);
        int H = bmif.height;
        int W = bmif.width;
        char* color = find_value(dict, len_dict, "component_name");
        int val = atoi(find_value(dict, len_dict, "component_value"));
        //printf("_%c_", color[0]);
        if(color[0] == 'r'){
            change_red_chanel(arr, H, W, val);
        }else if(color[0] == 'g'){
            change_green_chanel(arr, H, W, val);
        }else if (color[0] == 'b'){
            change_blue_chanel(arr, H, W, val);
        }

        write_bmp(find_value(dict, len_dict, "output"), arr, H, W, bmfh,
bmif);

        for (int i = 0; i < H; i++){
            free(arr[i]);
        }
        free(arr);

    }
    else if(strcmp(main_option,"rotate")==0){
```

```
        //printf("ROTATOS\n");

        BitmapFileHeader bmfh;
        BitmapInfoHeader bmif;
        Rgb **arr;
        arr = read_bmp(find_value(dict, len_dict, "input"), &bmfh, &bmif);
        //print_info_header(bmif);
        int H = bmif.height;
        int W = bmif.width;
        //int col[3] = {255, 255, 255};
        int col_f[3] = {255, 0, 0};

        int *coords_all = parse_coords(find_value(dict, len_dict,
"left_up"));
        int *coords_all_down = parse_coords(find_value(dict, len_dict,
"right_down"));
        int start_x = coords_all[0];
        int start_y = coords_all[1];
        int end_x = coords_all_down[0];
        int end_y = coords_all_down[1];
        int a = atoi(find_value(dict, len_dict, "angle"));

        turn(arr, H, W, start_x, start_y, end_x, end_y, a);
        //turn(arr, H, W, start_x, start_y, end_x, end_y, a);
        //turn(arr, H, W, start_x, start_y, end_x, end_y, a);
        //turn(arr, H, W, start_x, start_y, end_x, end_y, a);


        write_bmp(find_value(dict, len_dict, "output"), arr, H, W, bmfh,
bmif);

        for (int i = 0; i < H; i++){
            free(arr[i]);
        }
        free(arr);

    }


    else if(strcmp(main_option,"info")==0){
        //printf("INFOS");
        BitmapFileHeader bmfh;
        BitmapInfoHeader bmif;
        Rgb **arr;
        arr = read_bmp(find_value(dict, len_dict, "input"), &bmfh, &bmif);
        int H = bmif.height;
        int W = bmif.width;
        print_file_header(bmfh);
        print_info_header(bmif);
        for (int i = 0; i < H; i++){
            free(arr[i]);
        }
        free(arr);

    }
    else{
        if (help_flag){
            printf("Course work for option 4.12, created by Aleksandr
```

```
Klyukin.\n");
        }
    }
}
```

input_hell.h:

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <getopt.h>

#include <regex.h>

#include "structures.h"

#include "actions.h"

#include "files_action.h"

#include "checkers.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 *squared_lines_flag, int
*side_size_flag, int *left_up_flag, int *right_down_flag, int
*rgbfilter_flag, int *component_name_flag, int *component_value_flag, int
*thickness_flag, int *color_flag, int *fill_flag, int *fill_color_flag, int
*rotate_flag, int *angle_flag);

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

squared_lines_flag,int *side_size_flag, int *left_up_flag, int
*right_down_flag, int *rgbfilter_flag, int

                        *component_name_flag, int *component_value_flag, int
*thickness_flag, int *color_flag, int *fill_flag, int

                        *fill_color_flag, int *rotate_flag, int
*angle_flag);

void run(Element *dict, int len_dict);
```

checkers.c:

```
#include "checkers.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;
}

void check_one_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(41);
    }
    reti = regexec(&regex, coords, 0, NULL, 0);
    if (reti) {
        fprintf(stderr, "Coords are not correct\n");
        exit(41);
    }
}

void check_size(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, "Value 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;
    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_component_name(char *color){
    if(strcmp(color, "red") == 0 || strcmp(color, "green") == 0 ||
strcmp(color, "blue") == 0){
       int c = 0;
```

```
    }else{
        fprintf(stderr, "Component_name is not correct\n");
        exit(41);
    }
}

void check_angle(char *angle){
    if(strcmp(angle, "90") == 0 || strcmp(angle, "180") == 0 ||
strcmp(angle, "270") == 0){
       int c = 0;
    }else{
        fprintf(stderr, "Angle is not correct\n");
        exit(41);
    }
}
```

checkers.h:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <getopt.h>
#include <regex.h>
#include "structures.h"

int *parse_coords(char *coords);
int *parse_color(char *color);
void check_one_coords(char *coords);
void check_size(char *radius);
void check_thickness(char *thickness);
void check_color(char *color);
```

```
void check_component_name(char *color);

void check_angle(char *angle);
```