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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Обработка изображений

Студент гр. 3341 _____ Самокрутов А.Р.

Преподаватель _____ Глазунов С.А.

Санкт-Петербург

2024

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студент Самокрутов Артём Романович

Группа 3341

Тема работы: Обработка изображений

Вариант 4.6

Программа **обязательно должна иметь CLI** (опционально дополнительное использование GUI). Более подробно тут: **http://se.moevm.info/doku.php/courses:programming:rules_extra_kurs**

Программа должна реализовывать весь следующий функционал по обработке bmp-файла

Общие сведения

- 24 бита на цвет
- без сжатия
- файл может не соответствовать формату BMP, т.е. необходимо проверка на BMP формат (дополнительно стоит помнить, что версий у формата несколько). Если файл не соответствует формату BMP или его версии, то программа должна завершиться с соответствующей ошибкой.
- обратите внимание на выравнивание; мусорные данные, если их необходимо дописать в файл для выравнивания, должны быть нулями.
- обратите внимание на порядок записи пикселей
- все поля стандартных BMP заголовков в выходном файле должны иметь те же значения что и во входном (разумеется кроме тех, которые должны быть изменены).

Программа должна иметь следующую функции по обработке изображений:

- (1) Рисование отрезка. Флаг для выполнения данной операции: `--line`. Отрезок определяется:
  - координатами начала. Флаг `--start`, значение задаётся в

формате `x.y`, где x – координата по x, y – координата по y

- координатами конца. Флаг `--end` (аналогично флагу `--start`)
- цветом. Флаг `--color` (цвет задаётся строкой `rrr.ggg.bbb`, где rrr/ggg/bbb – числа, задающие цветовую компоненту. пример `--color 255.0.0` задаёт красный цвет)
- толщиной. Флаг `--thickness`. На вход принимает число больше 0

- (2) Инвертировать цвета в заданной окружности. Флаг для выполнения данной операции: `--inverse_circle`. Окружность 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
- (3) Обрезка изображения. Флаг для выполнения данной операции: `--trim`. Требуется обрезать изображение по заданной области. Область 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

Все подзадачи, ввод/вывод должны быть реализованы в виде отдельной функции.

Содержание пояснительной записки:

Аннотация, содержание, введение, ход выполнения, заключение, приложения.

Предполагаемый объем пояснительной записки:

Не менее 15 страниц.

Дата выдачи задания: 18.03.2024

Дата сдачи реферата: 21.05.2024

Дата защиты реферата: 23.05.2024

Студент _____ Самокрутов А.Р.

Преподаватель _____ Глазунов С.А.

АННОТАЦИЯ

Курсовая работа представляет собой программу на языке C, обрабатывающую изображения формата BMP, не имеющие сжатия, с глубиной цвета равной 24 битам.

Взаимодействие с программой осуществляется с помощью CLI (интерфейс командной строки). Программа принимает на вход флаги из командной строки, обрабатывает их и выполняет определенные манипуляции с изображением, затем сохраняет его изменённую версию, если это требуется и поданные аргументы были корректными.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью курсовой работы является изучение формата файлов BMP, а также реализация функций для работы с этими форматами файлов.

Для достижения поставленной цели требуется решить следующие задачи:

1. Изучить BMP формат изображений;
2. Получить информацию об изображении: размеры, содержимое и др.;
3. Обработать массив пикселей в соответствии с заданием;
4. Обработать исключительные случаи;
5. Сохранить итоговое изображение в новый файл.

1. СЧИТЫВАНИЕ И ОБРАБОТКА ФЛАГОВ

Для считывания и обработки аргументов командной строки, вводимых пользователем, используется функционал библиотеки *getopt.h*. Массив структур *long_options[]* хранит в себе информацию о каждом флаге, который должна обрабатывать программа, а строка *short_options* — информацию о всех возможных коротких версиях этих флагов. Для чтения и обработки флагов используется функция *getopt_long()*.

Описывается структура struct Config, которая содержит в себе информацию об изображении и действиях, которые необходимо с ним выполнить, например, название входного и выходного файла и указатель на то, какие манипуляции и с какими параметрами должны быть применены к изображению. Также реализованы структуры struct Line_config, struct Inverse_circle_config и struct Trim_config, которые содержают в себе информацию о том, с какими аргументами были вызваны функции рисования отрезка, инверсии цвета в заданной окружности и обрезания изображения соответственно. Так, структура struct Line_config содержит в себе переменные, которые указывают на то, был ли указан какой-либо параметр (--start, --end, --color, --thickness), и переменные, которые хранят в себе значения этих параметров.

Функция struct Config *get_options(int argc, char **argv) принимает на вход количество аргументов командной строки и массив со строками, в котором они хранятся, инициализирует структуру struct Config и заполняет её в помощью функции getopt_long() в цикле while. С помощью конструкции switch-case обрабатывается значение, возвращаемое этой функцией. При этом, если какое-то значение уже было записано в структуру, то при попытке записать его повторно программа завершает работу с соответствующим сообщением и кодом ошибки.

2. ОБРАБОТКА BMP ФАЙЛА

Для хранения данных об изображении была создана структура Bitmap_header_t, хранящая в себе данные из заголовка BMP файла, и Bitmap_image_t, в которой находятся описанная ранее структура, а также массив байтов собственно изображения, т.е. множества его пикселей, и число, в котором хранится размер изображения.

Чтения изображения производится с помощью функции Bitmap_image_t *load_bmp(const char *filename). Она открывает файл с названием filename, выделяет память под необходимые структуры. Далее она считывает с помощью функции fread() заголовок файла и проверяет его на корректность с помощью вспомогательной функции valid_header(), которая проверяет что считанные сигнатура, количество битов на пиксель, тип сжатия соответствуют формату BMP файла, используемому в работе. Далее считываются пиксели изображения. Если всё прошло успешно, то файл закрывается, и функция возвращает структуру со считанными данными.

Для сохранения изображения была написана функция void save_bmp(const Bitmap_image_t *bmp, const char *filename). Она записывает информацию о заголовке файла и пикселях изображения из bmp в файл с именем filename с помощью функции fread().

Были также написаны функции, позволяющие узнать цвет пикселя, расположенного по определённым координатам, и изменить его цвет. Для этого по его координатам рассчитывается положения пикселя в массиве байтов всех пикселей, после чего нужные байты либо возвращаются, либо меняются на другие.

3. ОБРАБОТКА ИЗОБРАЖЕНИЯ

3.1 Рисование отрезка

Для рисования отрезка был реализована функция void draw_line(Bitmap_image_t *image, Coord_t start, Coord_t end, RGB_t color, int thickness), являющая имплементацией модифицированного метода рисования линии Брезенхема. От обычного этот метод отличается тем, что на каждом шаге закрашивается не один пиксель, а рисуется окружность с диаметром, равным толщине линии. Для рисования окружности были реализованы две вспомогательные функции void draw_circle(Bitmap_image_t *image, Coord_t center, int radius, RGB_t color, int rim) и void fill_circle(Bitmap_image_t *image, Coord_t center, int radius, RGB_t color). Обе функции основаны на алгоритме Брезенхема для рисования окружности. Первая рисует окружность с толщиной rim, уходящей внутрь, а вторая рисует залитый круг. Они отвечают за рисование средней части отрезка и его концов соответственно. В ходе работы каждой из функций производится проверка корректности входных данных. Например, если поданная толщина линии thickness окажется отрицательной, то программа завершит работу с соответствующей ошибкой.

Для ускорения работы функции был реализован ряд оптимизаций. Так, если какая-то часть отрезка не входит в изображение, то она не рисуется. Также именно из соображений оптимизации для рисования средней части отрезка используется функция, рисующая именно окружность, а не круг, так как это уменьшает количество манипуляций с пикселями, а значит и время выполнения функции рисования отрезка.

3.2 Инверсия цветов в заданной окружности

Для инверсии цветов в заданной окружности была написана функция void invert_circle(Bitmap_image_t *image, Coord_t center, int radius). Она проверяет, что окружность с заданными параметрами находится в пределах изображения (в ином случае она завершает работу), а далее выполняет модифицированный алгоритм Брезенхема для рисования окружности. Отличием от стандартной имплементации является то, что она рисует не пиксели, а отрезки толщиной в

один пиксель так, чтобы они не накладывались друг от друга, т.е. так, чтобы по каждому из них функция проходила единожды. Это необходимо для корректной инверсии цвета.

Для инверсии цвета была реализована вспомогательная функция, которая по координатам пикселя возвращает его инверсированный цвет. Каждая его компонента считается как разность её максимального значения (255) и её значения в пикселе-оригинале.

3.3 Обрезка изображения

Для обрезки изображения была создана функция void crop(Bitmap_image_t *image, Coord_t left_up, Coord_t right_down). Сначала она с помощью вспомогательной функции set_area() устанавливает координаты границ зоны обрезки в корректные, т.к. на вход могут быть поданы координаты меньшие нуля или большие размеров изображения, и в таком случае их необходимо «сжать» до значений границ изображения. Далее считаются новая высота и ширина изображения, новое значение количества байтов для выравнивания в изображении. После этого заголовок изображения image->hdr обновляется полученными значениями. Далее создаётся новый массив байтов пикселей, а затем в него копируются только те исходные байты пикселей, которые попадают в заданную область, и этот массив байтов сохраняется в поле image->data, а прошлое значения этого поля освобождается из памяти.

4. ФАЙЛ CW.C

В файле cw.c описана функция main(). Она создаёт структуру config с помощью функции init_config(), после чего заполняет её аргументами, введёнными пользователем в командной строке через get_options(). Далее в зависимости от содержания этой структуры выполняет определённые действия: выводит информацию о поданном изображении; выводит справку о программе; открывает изображение, рисует отрезок с заданными параметрами, инвертирует цвет в выбранной окружности либо обрезает изображение, после чего сохраняет изображение в новый файл. В конце вся динамически выделенная память, которая ещё не была очищена, наконец очищается.

5. MAKEFILE

Для сборки программы был написан Makefile. Это необходимо, так как программы разбита на несколько файлов с исходным кодом и заголовочных файлов: cwstructures.h (определения структур), cwbmp.h, cwbmp.c (чтение и сохранение BMP файлов), cwdrawing.h, cwdrawing.c (работа с изображениями), cwoptions.h, cwoptions.c (обработка флагов и выполнение команд), cwerror.h, cwerror.c (обработка ошибок), cw.c.

В файле указаны пути к файлам с исходным кодом (PREF_CORE), заголовочным файлам (PREF_HEADER) и бинарным файлам (PREF_BIN). Также были установлены компилятор (CC) и флаги для компилятора (CFLAGS).

С помощью wildcard определяются файлы с исходным кодом (CORE), из них с помощью patsubst находятся названия соответствующих объектных файлов (OBJ).

Цель all вызывает цель make_bin, создающую папку для объектных файлов, а затем вызывает цель, компилирующую программу.

Цель clean удаляет исполняемый файл cw и папку bin с её содержимым.

Ознакомиться с исходным кодом можно в приложении А, а с примерами работы программы — в приложении В.

ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы был изучен формат файлов BMP, а также реализация функций для работы с этими форматами файлов.

Решены следующие задачи:

1. Изучен BMP формат изображений;
2. Получена информация об изображении: размеры, содержимое и др.;
3. Обработан массив пикселей в соответствии с заданием;
4. Обработаны исключительные случаи;
5. Итоговое изображение сохранено в новый файл.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. М. М. Заславский, А. А. Лисс, А. В. Гаврилов, С. А. Глазунов, Я. С. Государкин, С. А. Тиняков, В. П. Голубева, К. В. Чайка, В. Е. Допира. Б17 Базовые сведения к выполнению курсовой работы по дисциплине «Программирование». Второй семестр, 2024.

2. getopt(3) – Linux manual page. URL: https://man7.org/linux/man-pages/man3/getopt.3.html

3. Bresenham J. E. Algorithm for computer control of a digital plotter // IBM Systems journal. 1965. Т. 4, №. 1. P. 25–30

ПРИЛОЖЕНИЕ В
ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл: cwstructures.h

```
# #ifndef CWSTRUCTURES_H
#define CWSTRUCTURES_H


#pragma pack(push, 1)
typedef struct {
    unsigned short signature;
    unsigned int file_size;
    unsigned short reserved1;
    unsigned short reserved2;
    unsigned int data_offset;

    unsigned int header_size;
    signed int width; //signed??
    signed int height; //signed??
    unsigned short color_planes_num;
    unsigned short bits_per_pixel;
    unsigned int compression;
    unsigned int image_size;
    unsigned int horizontal_resolution_pxpm;
    unsigned int vertical_resolution_pxpm;
    unsigned int number_of_colors;
    unsigned int number_of_important_colors;
} Bitmap_header_t;
#pragma pack(pop)

typedef struct {
    unsigned char b, g, r;
} RGB_t;

typedef struct {
    Bitmap_header_t hdr;
    unsigned char *data;
```

```
    unsigned int data_size;
} Bitmap_image_t;

typedef struct {
    int x, y;
} Coord_t;


#endif
```

Файл: cwerror.h

```
#ifndef CWERROR_H
#define CWERROR_H

#include <stdio.h>
#include <stdlib.h>

#define MEMORY_ERROR_RET_VAL 40
#define MEMORY_ERROR_MSG "Cannot allocate of free dynamic memory on heap."
#define FILE_ERROR_RET_VAL 41
#define FILE_ERROR_MSG "File cannot be opened. Perhaps, this file does not exits."
#define BMP_DATA_ERROR_RET_VAL 42
#define BMP_DATA_ERROR_MSG "Header cannot be read or written."
#define NOT_A_BMP_ERROR_RET_VAL 43
#define NOT_A_BMP_ERROR_MESSAGE "Input is either not a BMP file or incorrect BMP file format. This program only accepts 24-bit BMP files with no compression."
#define INVALID_COORDINATE_ERROR_RET_VAL 44
#define INVALID_COORDINATE_ERROR_MSG "Invalid pixel coordinate."
#define INVALID_RGB_ERROR_RET_VAL 45
#define INVALID_RGB_ERROR_MSG "Invalid pixel color."
#define INVALID_DRAWING_COORDINATES_RET_VAL 46
#define INVALID_DRAWING_COORDINATES_MSG "Invalid coordinates for a figure (e.g. start, end, center, etc.)"
#define INVALID_OPTION_ARGUMENT_ERROR_RET_VAL 47
#define INVALID_OPTION_ARGUMENT_ERROR_MSG "Option has an invalid argument."
```

```
#define OPTION_ERROR_RET_VAL 48
#define OPTION_ERROR_MSG "Unknown option, invalid option, invalid argument, missing argument, etc."
#define CONFIG_ERROR_RET_VAL 49
#define CONFIG_ERROR_MSG "Invalid configuration."


void throw_error(const char *text, int return_value);


#endif
```

Файл: cwerror.c

```
#ifndef CWERROR_C
#define CWERROR_C

#include "../cwerror/cwerror.h"


void throw_error(const char *text, int return_value)
{
    printf("ERROR: %s.\nThe program has been terminated.\n", text);
    exit(return_value);
}

#endif
```

Файл: cwbmp.h

```
#ifndef CWBMP_H
#define CWBMP_H

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include "../cwstructures/cwstructures.h"
#include “../cwerror/cwerror.h

#define BMP_SIGNATURE 0x4D42
#define BITS_PER_PIXEL 24
#define COMPRESSION 0
#define BLUE 2
```

```
#define GREEN 1
#define RED 0


void info(const char *filename);

Bitmap_image_t *load_bmp(const char *file_name);

void save_bmp(const Bitmap_image_t *bitmap_image, const char *file_name);

void free_bmp(Bitmap_image_t *bitmap_file);

bool valid_rgb(RGB_t rgb_color);

bool valid_coordinate(Coord_t coordinate, int width, int height);

bool valid_header(const Bitmap_image_t *bitmap_image); //static

int get_padding(int width);

int get_row_size(int width);

int get_position(Coord_t coordinate, int width, int height);

RGB_t get_pixel(const Bitmap_image_t *bitmap_image, Coord_t coordinate);

void set_pixel(Bitmap_image_t *bitmap_image, Coord_t coordinate, RGB_t
color);


#endif
```

Файл: cwbmp.c

```
#ifndef CWBMP_C
#define CWBMP_C

#include "../cwbmp/cwbmp.h"


void info(const char *filename)
{
    Bitmap_image_t *bmp = load_bmp(filename);
```

```
    Bitmap_header_t *header = &bmp->hdr;

    printf("Signature -- %hu\n", header->signature);
    printf("File size -- %u\n", header->file_size);
    printf("Data start at byte №%u\n", header->data_offset);
    printf("Header size -- %u\n", header->header_size);
    printf("Size -- %i by %i\n", header->width, header->height);
    printf("Number of color planes -- %hu\n", header->color_planes_num);
    printf("Bits per pixel -- %hu\n", header->bits_per_pixel);
    printf("Compression -- %u\n", header->compression);
    printf("Resolution -- %u by %u\n", header-
>horizontal_resolution_pxpm, header->vertical_resolution_pxpm);
    printf("Number of colors -- %u\n", header->number_of_colors);
    printf("Number of important colors -- %u\n", header-
>number_of_important_colors);

    free(bmp);
}

Bitmap_image_t *load_bmp(const char *filename)
{
    FILE *file = fopen(filename, "rb");

    if (file == NULL)
    {
        throw_error(FILE_ERROR_MSG, FILE_ERROR_RET_VAL);
    }

    Bitmap_image_t *bmp = (Bitmap_image_t
*)malloc(sizeof(Bitmap_image_t));

    if (bmp == NULL)
    {
        throw_error(MEMORY_ERROR_MSG, MEMORY_ERROR_RET_VAL);
    }

    int read_hdr = fread(&bmp->hdr, sizeof(Bitmap_header_t), 1, file);
```

```
    if (read_hdr != 1)
    {
        throw_error(BMP_DATA_ERROR_MSG, BMP_DATA_ERROR_RET_VAL);
    }

    if (!valid_header(bmp))
    {
        throw_error(NOT_A_BMP_ERROR_MESSAGE, NOT_A_BMP_ERROR_RET_VAL);
    }

    bmp->data_size = bmp->hdr.file_size - sizeof(Bitmap_header_t);
    bmp->data = (unsigned char *)malloc(bmp->data_size * sizeof(unsigned
char));

    if (bmp->data == NULL) {
        throw_error(MEMORY_ERROR_MSG, MEMORY_ERROR_RET_VAL);
    }

    fseek(file, bmp->hdr.data_offset, SEEK_SET);
    int read_data = fread(bmp->data, bmp->data_size, 1, file);

    if (read_data != 1) {
        throw_error(BMP_DATA_ERROR_MSG, BMP_DATA_ERROR_RET_VAL);
    }

    fclose(file);
    return bmp;
}

void save_bmp(const Bitmap_image_t *bmp, const char *filename)
{
    FILE *file = fopen(filename, "wb");

    if (file == NULL)
    {
        throw_error(FILE_ERROR_MSG, FILE_ERROR_RET_VAL);
    }
```

```
    rewind(file);
    int written_hdr = fwrite(&bmp->hdr, sizeof(Bitmap_header_t), 1,
file);

    if (written_hdr != 1)
    {
        throw_error(BMP_DATA_ERROR_MSG, BMP_DATA_ERROR_RET_VAL);
    }

    fseek(file, bmp->hdr.data_offset, SEEK_SET);
    int written_data = fwrite(bmp->data, bmp->data_size * sizeof(unsigned
char), 1, file);

    if (written_data != 1)
    {
        throw_error(BMP_DATA_ERROR_MSG, BMP_DATA_ERROR_RET_VAL);
    }

    fclose(file);
}

void free_bmp(Bitmap_image_t *bmp)
{
    if (bmp != NULL)
    {
        if (bmp->data != NULL)
        {
            free(bmp->data);
        }
        free(bmp);
    }
}

bool valid_rgb(RGB_t color)
{
    if (color.r < 0x00 || color.r > 0xFF)
    {
        return false;
```

```
    }

    if (color.g < 0x00 || color.g > 0xFF)
    {
        return false;
    }

    if (color.b < 0x00 || color.b > 0xFF)
    {
        return false;
    }

    return true;
}

bool valid_coordinate(Coord_t coordinate, int width, int height)
{
    if (coordinate.x < 0 || coordinate.x >= width)
    {
        return false;
    }

    if (coordinate.y < 0 || coordinate.y >= height)
    {
        return false;
    }

    return true;
}

bool valid_header(const Bitmap_image_t *image)
{
    if (image->hdr.signature != BMP_SIGNATURE)
    {
        return false;
    }

    if (image->hdr.bits_per_pixel != BITS_PER_PIXEL)
```

```
    {
        return false;
    }

    if (image->hdr.compression != COMPRESSION)
    {
        return false;
    }

    return true;
}

int get_padding(int width)
{
    return (4 - (width * sizeof(RGB_t)) % 4) % 4;
}

int get_row_size(int width)
{
    return (width * sizeof(RGB_t)) + get_padding(width);
}

int get_position(Coord_t coordinate, int width, int height)
{
    if (!valid_coordinate(coordinate, width, height))
    {
        throw_error(INVALID_COORDINATE_ERROR_MSG,
INVALID_COORDINATE_ERROR_RET_VAL);
    }

    int col = coordinate.x * sizeof(RGB_t);
    int row = (height - coordinate.y - 1) * get_row_size(width);

    return col + row;
}

RGB_t get_pixel(const Bitmap_image_t *bmp, Coord_t coordinate)
{
```

```
    int width = bmp->hdr.width;
    int height = bmp->hdr.height;
    int pos = get_position(coordinate, width, height);

    RGB_t color = {
        bmp->data[pos + RED],
        bmp->data[pos + GREEN],
        bmp->data[pos + BLUE]
    };

    return color;
}

void set_pixel(Bitmap_image_t *bmp, Coord_t coordinate, RGB_t color)
{
    if (!valid_rgb(color))
    {
        throw_error(INVALID_RGB_ERROR_MSG, INVALID_RGB_ERROR_RET_VAL);
    }

    int width = bmp->hdr.width;
    int height = bmp->hdr.height;
    int pos = get_position(coordinate, width, height);

    bmp->data[pos + RED] = color.r;
    bmp->data[pos + GREEN] = color.g;
    bmp->data[pos + BLUE] = color.b;
}


#endif
```

Файл: cwdrawing.h

```
#ifndef CWDRAWING_H
#define CWDRAWING_H

#include <math.h>
#include <stdio.h>
#include <stdlib.h>
```

```
#include <string.h>
#include <stdbool.h>
#include "../cwbmp/cwbmp.h"
#include "../cwerror/cwerror.h"


void draw_dot(Bitmap_image_t *image, Coord_t coordinate, RGB_t color);

void fill_circle(Bitmap_image_t *image, Coord_t center, int radius, RGB_t
color); //!!

void draw_circle(Bitmap_image_t *image, Coord_t center, int radius, RGB_t
color, int rim);

void draw_line(Bitmap_image_t *image, Coord_t start, Coord_t end, RGB_t
color, int thickness);

void invert_dot(Bitmap_image_t *image, Coord_t coordinate);

void invert_circle(Bitmap_image_t *image, Coord_t center, int radius);

void crop(Bitmap_image_t *image, Coord_t left_up, Coord_t right_down);


#endif
```

Файл: cwdrawing.c

```
#ifndef DRAWING_C
#define DRAWING_c

#include "../cwdrawing/cwdrawing.h"


static void swap_x(Coord_t *first, Coord_t *second)
{
    Coord_t tmp = *first;
    first->x = second->x;
    second->x = tmp.x;
}
```

```
static void swap_y(Coord_t *first, Coord_t *second)
{
    Coord_t tmp = *first;
    first->y = second->y;
    second->y = tmp.y;
}

static void set_area(int width, int height, Coord_t *left_up, Coord_t
*right_down)
{
    *left_up = (Coord_t){left_up->x < 0 ? 0 : left_up->x,
                         left_up->y < 0 ? 0 : left_up->y};
    *left_up = (Coord_t){left_up->x >= width ? width : left_up->x,
                         left_up->y >= height ? height : left_up->y};

    *right_down = (Coord_t){right_down->x < 0 ? 0 : right_down->x,
                         right_down->y < 0 ? 0 : right_down->y};
    *right_down = (Coord_t){right_down->x >= width ? width : right_down-
>x,
                         right_down->y >= height ? height : right_down-
>y};

    if (right_down->x < left_up->x)
    {
        swap_x(right_down, left_up);
    }

    if (left_up->y > right_down->y)
    {
        swap_y(left_up, right_down);
    }
}

void draw_dot(Bitmap_image_t *image, Coord_t coordinate, RGB_t color)
{
    if     (!valid_coordinate(coordinate,     image->hdr.width,     image-
>hdr.height))
```

```
    {
        return;
    }

    if (!valid_rgb(color))
    {
        throw_error(INVALID_COORDINATE_ERROR_MSG,
INVALID_COORDINATE_ERROR_RET_VAL);
    }

    set_pixel(image, coordinate, color);
}

static void draw_horizontal_line(Bitmap_image_t *image, Coord_t start,
Coord_t end, RGB_t color)
{
    if (end.y != start.y)
    {
        throw_error(INVALID_DRAWING_COORDINATES_MSG,
INVALID_DRAWING_COORDINATES_RET_VAL);
    }

    if (start.y < 0 || start.y >= image->hdr.height)
    {
        return;
    }

    if (start.x > end.x)
    {
        swap_x(&start, &end);
    }

    for (Coord_t current = start; current.x <= end.x; current.x++)
    {
        draw_dot(image, current, color);
    }
}
```

```
static void draw_vertical_line(Bitmap_image_t *image, Coord_t start,
Coord_t end, RGB_t color)
{
    if (end.x != start.x)
    {
        throw_error(INVALID_DRAWING_COORDINATES_MSG,
INVALID_DRAWING_COORDINATES_RET_VAL);
    }

    if (start.x < 0 || start.x >= image->hdr.width)
    {
        return;
    }

    if (start.y > end.y)
    {
        swap_y(&start, &end);
    }

    for (Coord_t current = start; current.y <= end.y; current.y++)
    {
        draw_dot(image, current, color);
    }
}

static bool circle_is_visible(Coord_t center, int radius, int width, int
height)
{
    if (center.x + radius < 0)
    {
        return false;
    }

    if (center.y + radius < 0)
    {
        return false;
    }
```

```
    if (center.x - radius >= width)
    {
        return false;
    }

    if (center.y - radius >= height)
    {
        return false;
    }

    return true;
}

void fill_circle(Bitmap_image_t *image, Coord_t center, int radius, RGB_t
color)
{
    if (radius < 0)
    {
        throw_error(INVALID_DRAWING_COORDINATES_MSG,
INVALID_DRAWING_COORDINATES_RET_VAL);
    }

    if (!valid_rgb(color))
    {
        throw_error(INVALID_RGB_ERROR_MSG, INVALID_RGB_ERROR_RET_VAL);
    }

    if  (!circle_is_visible(center,  radius,  image->hdr.width,  image-
>hdr.height))
    {
        return;
    }

    int error = 3 - (2 * radius);

    draw_dot(image, (Coord_t){center.x, center.y + radius}, color);
    draw_dot(image, (Coord_t){center.x, center.y - radius}, color);
    draw_horizontal_line(image, (Coord_t){center.x + radius, center.y},
```

```
                                (Coord_t){center.x - radius, center.y},
color);

    for (Coord_t current = {0, radius}; current.x <= current.y;
current.x++)
    {
        int vertical_step = 0;
        if (error < 0)
        {
            error += 4 * current.x + 1;
        }
        else
        {
            current.y--;
            vertical_step = 1;
            error += 4 * (current.x - current.y) + 1;
        }

        if (vertical_step == 1)
        {
            draw_horizontal_line(image, (Coord_t){center.x + current.x,
center.y - current.y},
                                        (Coord_t){center.x - current.x,
center.y - current.y}, color);
            draw_horizontal_line(image, (Coord_t){center.x + current.x,
center.y + current.y},
                                        (Coord_t){center.x - current.x,
center.y + current.y}, color);
        }
        else
        {
            draw_dot(image, (Coord_t){center.x + current.x, center.y -
current.y}, color);
            draw_dot(image, (Coord_t){center.x - current.x, center.y -
current.y}, color);
            draw_dot(image, (Coord_t){center.x + current.x, center.y +
current.y}, color);
```

```
            draw_dot(image, (Coord_t){center.x - current.x, center.y +
current.y}, color);
        }

        if (current.y != current.x)
        {
            draw_horizontal_line(image, (Coord_t){center.x + current.y,
center.y - current.x},
                                        (Coord_t){center.x - current.y,
center.y - current.x}, color);
            draw_horizontal_line(image, (Coord_t){center.x + current.y,
center.y + current.x},
                                        (Coord_t){center.x - current.y,
center.y + current.x}, color);
        }
    }
}

void draw_circle(Bitmap_image_t *image, Coord_t center, int radius, RGB_t
color, int rim) {
    if (radius < 0)
    {
        throw_error(INVALID_DRAWING_COORDINATES_MSG,
INVALID_DRAWING_COORDINATES_RET_VAL);
    }

    if (!valid_rgb(color))
    {
        throw_error(INVALID_RGB_ERROR_MSG, INVALID_RGB_ERROR_RET_VAL);
    }

    if (!circle_is_visible(center, radius, image->hdr.width, image-
>hdr.height))
    {
        return;
    }

    int error = 3 - (2 * radius);
```

```
    draw_horizontal_line(image, (Coord_t){center.x + radius, center.y},
                                (Coord_t){center.x + radius - rim,
center.y}, color);
    draw_vertical_line(image, (Coord_t){center.x, center.y + radius},
                               (Coord_t){center.x, center.y + radius -
rim}, color);
    draw_horizontal_line(image, (Coord_t){center.x - radius, center.y},
                                (Coord_t){center.x - radius + rim,
center.y}, color);
    draw_vertical_line(image, (Coord_t){center.x, center.y - radius},
                               (Coord_t){center.x, center.y - radius +
rim}, color);

    for (Coord_t current = {0, radius}; current.x <= current.y;
current.x++)
    {
        if (error < 0) {
            error += 4 * current.x + 1;
        }
        else
        {
            current.y--;
            error += 4 * (current.x - current.y) + 1;
        }

        draw_horizontal_line(image, (Coord_t){center.x + current.x,
center.y + current.y},
                                    (Coord_t){center.x + current.x - rim,
center.y + current.y}, color);
        draw_vertical_line(image, (Coord_t){center.x + current.y,
center.y + current.x},
                                  (Coord_t){center.x + current.y,
center.y + current.x - rim}, color);
        draw_vertical_line(image, (Coord_t){center.x - current.y,
center.y + current.x},
                                  (Coord_t){center.x - current.y,
center.y + current.x - rim}, color);
```

```
        draw_horizontal_line(image,   (Coord_t){center.x   -   current.x,
center.y + current.y},
                                     (Coord_t){center.x - current.x + rim,
center.y + current.y}, color);
        draw_horizontal_line(image,   (Coord_t){center.x   -   current.x,
center.y - current.y},
                                     (Coord_t){center.x - current.x + rim,
center.y - current.y}, color);
        draw_vertical_line(image,    (Coord_t){center.x    -    current.y,
center.y - current.x},
                                   (Coord_t){center.x     -     current.y,
center.y - current.x + rim}, color);
        draw_vertical_line(image,    (Coord_t){center.x    +    current.y,
center.y - current.x},
                                   (Coord_t){center.x     +     current.y,
center.y - current.x + rim}, color);
        draw_horizontal_line(image,   (Coord_t){center.x   +   current.x,
center.y - current.y},
                                     (Coord_t){center.x + current.x - rim,
center.y - current.y}, color);
    }
}

void draw_line(Bitmap_image_t *image, Coord_t start, Coord_t end, RGB_t
color, int thickness)
{
    if (thickness < 0)
    {
        throw_error(INVALID_COORDINATE_ERROR_MSG,
INVALID_COORDINATE_ERROR_RET_VAL);
    }

    int delta_x = abs(end.x - start.x);
    int delta_y = -abs(end.y - start.y);

    int step_x = end.x > start.x ? 1 : -1;
    int step_y = end.y > start.y ? 1 : -1;
```

```
int error = delta_x + delta_y;

Coord_t current = start;
while (1)
{
    if (thickness == 1)
    {
        draw_dot(image, current, color);
    }
    else
    {
        draw_circle(image, current, thickness/2, color, 2);
    }

    if (current.x == end.x && current.y == end.y)
    {
        break;
    }

    if (2 * error >= delta_y)
    {
        if (current.x == end.x)
        {
            break;
        }
        error += delta_y;
        current.x += step_x;
    }
    if (2 * error <= delta_x)
    {
        if (current.y == end.y)
        {
            break;
        }
        error += delta_x;
        current.y += step_y;
    }
}
```

```
    fill_circle(image, start, thickness/2, color);
    fill_circle(image, end, thickness/2, color);
}

static RGB_t invert_color(RGB_t color)
{
    if (!valid_rgb(color))
    {
        throw_error(INVALID_COORDINATE_ERROR_MSG,
INVALID_COORDINATE_ERROR_RET_VAL);
    }

    RGB_t inverted = {0xFF - color.r, 0xFF - color.g, 0xFF - color.b};
    return inverted;
}

void invert_dot(Bitmap_image_t *image, Coord_t coordinate)
{
    if     (!valid_coordinate(coordinate,     image->hdr.width,     image-
>hdr.height))
    {
        return;
    }

    set_pixel(image,      coordinate,      invert_color(get_pixel(image,
coordinate)));
}

static void invert_horizontal_line(Bitmap_image_t *image, Coord_t start,
Coord_t end)
{
    if (end.y != start.y)
    {
        throw_error(INVALID_COORDINATE_ERROR_MSG,
INVALID_DRAWING_COORDINATES_RET_VAL);
    }
```

```
    if (start.y < 0 || start.y >= image->hdr.height)
    {
        return;
    }

    if (start.x > end.x)
    {
        swap_x(&start, &end);
    }

    for (Coord_t current = start; current.x <= end.x; current.x++)
    {
        invert_dot(image, current);
    }
}

void invert_circle(Bitmap_image_t *image, Coord_t center, int radius)
{
    if (radius < 0)
    {
        throw_error(INVALID_COORDINATE_ERROR_MSG,
INVALID_COORDINATE_ERROR_RET_VAL);
    }

    if   (!circle_is_visible(center,   radius,   image->hdr.width,   image-
>hdr.height))
    {
        return;
    }

    int error = 3 - (2 * radius);

    invert_dot(image, (Coord_t){center.x, center.y + radius});
    invert_dot(image, (Coord_t){center.x, center.y - radius});
    invert_horizontal_line(image, (Coord_t){center.x + radius, center.y},
                                  (Coord_t){center.x      -      radius,
center.y});
```

```
    for (Coord_t current = {0, radius}; current.x < current.y;
current.x++)
    {
        int vertical_step = 0;

        if (error < 0)
        {
            error += 4 * current.x + 1;
        }
        else
        {
            current.y--;
            vertical_step = 1;
            error += 4 * (current.x - current.y) + 1;
        }

        if (vertical_step == 1)
        {
            invert_horizontal_line(image, (Coord_t){center.x + current.x,
center.y - current.y},
                                          (Coord_t){center.x - current.x,
center.y - current.y});
            invert_horizontal_line(image, (Coord_t){center.x + current.x,
center.y + current.y},
                                          (Coord_t){center.x - current.x,
center.y + current.y});
        }
        else
        {
            invert_dot(image, (Coord_t){center.x + current.x, center.y -
current.y});
            invert_dot(image, (Coord_t){center.x - current.x, center.y -
current.y});
            invert_dot(image, (Coord_t){center.x + current.x, center.y +
current.y});
            invert_dot(image, (Coord_t){center.x - current.x, center.y +
current.y});
        }
```

```
        if (current.y != current.x)
        {
            invert_horizontal_line(image, (Coord_t){center.x + current.y,
center.y - current.x},
                                           (Coord_t){center.x - current.y,
center.y - current.x});
            invert_horizontal_line(image, (Coord_t){center.x + current.y,
center.y + current.x},
                                           (Coord_t){center.x - current.y,
center.y + current.x});
        }
    }
}

void crop(Bitmap_image_t *image, Coord_t left_up, Coord_t right_down)
{
    set_area(image->hdr.width, image->hdr.height, &left_up, &right_down);

    int new_height = right_down.y - left_up.y;
    int new_width = right_down.x - left_up.x;

    int old_width = image->hdr.width;
    int old_height = image->hdr.height;
    image->hdr.width = new_width;
    image->hdr.height = new_height;

    int new_padding = get_padding(new_width);

    image->data_size = new_height * (new_width * sizeof(RGB_t) +
new_padding);
    image->hdr.image_size = image->data_size;
    image->hdr.file_size = image->data_size + sizeof(Bitmap_header_t);

    unsigned char *new_data = (unsigned char *)calloc(image->data_size,
sizeof(unsigned char));
    for (int row = 0; row < new_height; row++)
    {
```

```
        memcpy(new_data  +  get_position((Coord_t){0,  row},  new_width,
new_height),
               image->data + get_position((Coord_t){left_up.x, left_up.y
+ row}, old_width, old_height),
               new_width * sizeof(RGB_t));
    }

    free(image->data);
    image->data = new_data;
}


#endif
```

Файл: cwoptions.h

```
#ifndef CWOPTIONS_H
#define CWOPTIONS_H

#include <stdlib.h>
#include <string.h>
#include <getopt.h>
#include <stdio.h>
#include <stdbool.h>
#include "../cwstructures/cwstructures.h"
#include "../cwdrawing/cwdrawing.h"
#include "../cwerror/cwerror.h"

#define INFO_OPT 1000
#define LINE_OPT 1001
#define INVERSE_CIRCLE_OPT 1002
#define TRIM_OPT 1003
#define START_OPT 1004
#define END_OPT 1005
#define COLOR_OPT 1006
#define THICKNESS_OPT 1007
#define CENTER_OPT 1008
#define RADIUS_OPT 1009
#define LEFT_UP_OPT 1010
#define RIGHT_DOWN_OPT 1011
```

```
#define CONTRAST_OPT 1012
#define ALPHA_OPT 1013
#define BETA_OPT 1014

#define MAX_FILENAME_LEN 127


struct Line_config {
    bool flag_start;
    bool flag_end;
    bool flag_color;
    bool flag_thickness;

    Coord_t start;
    Coord_t end;
    RGB_t color;
    int thickness;
};

struct Inverse_circle_config {
    bool flag_center;
    bool flag_radius;

    Coord_t center;
    int radius;
};

struct Trim_config {
    bool flag_left_up;
    bool flag_right_down;

    Coord_t left_up;
    Coord_t right_down;
};

struct Contrast_config {
    bool flag_alpha;
    bool flag_beta;
```

```
    float alpha;
    int beta;
};

struct Config {
    char *input_file;
    char *output_file;

    bool flag_help;
    bool flag_info;

    bool flag_line;
    bool flag_inverse_circle;
    bool flag_trim;
    bool flag_contrast;

    struct Line_config *line;
    struct Inverse_circle_config *inverse_circle;
    struct Trim_config *trim;
    struct Contrast_config *contrast;
};

struct Config *init_config();

void destroy_config(struct Config *config);

bool big_flag_is_set(const struct Config *config);

struct Config *get_options(int argc, char **argv);


#endif
```

Файл: cwoptions.c

```
#ifndef CWOPTIONS_C
#define CWOPTIONS_C

#include "../cwoptions/cwoptions.h"
```

```
static int arg_to_int(const char *arg)
{
	if (arg == NULL)
	{
		throw_error(INVALID_OPTION_ARGUMENT_ERROR_MSG,
INVALID_OPTION_ARGUMENT_ERROR_RET_VAL);
	}

	int num;
	int read = sscanf(arg, "%d", &num);

	if (read != 1)
	{
		throw_error(INVALID_OPTION_ARGUMENT_ERROR_MSG,
INVALID_OPTION_ARGUMENT_ERROR_RET_VAL);
	}

	return num;
}
static Coord_t arg_to_coord(const char *arg)
{
	if (arg == NULL)
	{
		throw_error(INVALID_OPTION_ARGUMENT_ERROR_MSG,
INVALID_OPTION_ARGUMENT_ERROR_RET_VAL);
	}

	int x, y;
	int read = sscanf(arg, "%d.%d", &x, &y);

	if (read != 2)
	{
		throw_error(INVALID_OPTION_ARGUMENT_ERROR_MSG,
INVALID_OPTION_ARGUMENT_ERROR_RET_VAL);
	}

	return (Coord_t){x, y};
```

```
}

static RGB_t arg_to_rgb(const char *arg)
{
      if (arg == NULL)
      {
            throw_error(INVALID_OPTION_ARGUMENT_ERROR_MSG,
INVALID_OPTION_ARGUMENT_ERROR_RET_VAL);
      }

      int r, g, b;
      int read = sscanf(arg, "%d.%d.%d", &r, &g, &b);

      if (read != 3)
      {
            throw_error(INVALID_OPTION_ARGUMENT_ERROR_MSG,
INVALID_OPTION_ARGUMENT_ERROR_RET_VAL);
      }

      return (RGB_t){r, g, b};
}

static float arg_to_float(const char *arg)
{
      if (arg == NULL)
      {
            throw_error(INVALID_OPTION_ARGUMENT_ERROR_MSG,
INVALID_OPTION_ARGUMENT_ERROR_RET_VAL);
      }

      float f;
      int read = sscanf(arg, "%f", &f);

      if (read != 1)
      {
            throw_error(INVALID_OPTION_ARGUMENT_ERROR_MSG,
INVALID_OPTION_ARGUMENT_ERROR_RET_VAL);
      }
```

```
	return f;
}

struct Config *init_config()
{
	struct Config *config = (struct Config *)calloc(1, sizeof(struct
Config));

	if (config == NULL)
	{
		throw_error(MEMORY_ERROR_MSG, MEMORY_ERROR_RET_VAL);
	}

	config->input_file = (char *)calloc(128, sizeof(char));
	config->output_file = (char *)calloc(128, sizeof(char));
	config->line = (struct Line_config *)calloc(1, sizeof(struct
Line_config));
	config->inverse_circle = (struct Inverse_circle_config *)calloc(1,
sizeof(struct Inverse_circle_config));
	config->trim = (struct Trim_config *)calloc(1, sizeof(struct
Trim_config));
	config->contrast = (struct Contrast_config *)calloc(1,
sizeof(struct Contrast_config));

	if (config->input_file == NULL ||
	   config->output_file == NULL ||
		config->line == NULL ||
		config->inverse_circle == NULL ||
		config->trim == NULL ||
		config->contrast == NULL)
	{
		throw_error(MEMORY_ERROR_MSG, MEMORY_ERROR_RET_VAL);
	}

	return config;
}
```

```
void destroy_config(struct Config *config)
{
      if (config != NULL)
      {
            if (config->input_file != NULL)
            {
                  free(config->input_file);
            }

            if (config->output_file != NULL)
            {
                  free(config->output_file);
            }

            if (config->line != NULL)
            {
                  free(config->line);
            }

            if (config->inverse_circle != NULL)
            {
                  free(config->inverse_circle);
            }

            if (config->trim != NULL)
            {
                  free(config->trim);
            }

            if (config->contrast != NULL)
            {
                  free(config->contrast);
            }
      }
}

bool big_flag_is_set(const struct Config* config)
{
```

```
	if (config->flag_line)
	{
		return true;
	}

	if (config->flag_inverse_circle)
	{
		return true;
	}

	if (config->flag_trim)
	{
		return true;
	}

	if (config->flag_contrast)
	{
		return true;
	}

	return false;
}

struct Config *get_options(int argc, char **argv)
{
	const char *short_options = "ho:i:";

	const struct option long_options[] = {
		{"help", no_argument, NULL, 'h'},
		{"input", required_argument, NULL, 'i'},
		{"output", required_argument, NULL, 'o'},
		{"info", no_argument, NULL, INFO_OPT},
		{"line", no_argument, NULL, LINE_OPT},
		{"inverse_circle", no_argument, NULL, INVERSE_CIRCLE_OPT},
		{"trim", no_argument, NULL, TRIM_OPT},
		{"start", required_argument, NULL, START_OPT},
		{"end", required_argument, NULL, END_OPT},
		{"color", required_argument, NULL, COLOR_OPT},
```

```
            {"thickness", required_argument, NULL, THICKNESS_OPT},
            {"center", required_argument, NULL, CENTER_OPT},
            {"radius", required_argument, NULL, RADIUS_OPT},
            {"left_up", required_argument, NULL, LEFT_UP_OPT},
            {"right_down", required_argument, NULL, RIGHT_DOWN_OPT},
            {"contrast", no_argument, NULL, CONTRAST_OPT},
            {"alpha", required_argument, NULL, ALPHA_OPT},
            {"beta", required_argument, NULL, BETA_OPT},
            {NULL, no_argument, NULL, 0}
      };

      struct Config *config = init_config();

      int opt, option_index = -1;
      char *last_argument = (char *)malloc((MAX_FILENAME_LEN + 1) *
sizeof(char));

      if (last_argument == NULL)
      {
            throw_error(MEMORY_ERROR_MSG, MEMORY_ERROR_RET_VAL);
      }

      strncpy(last_argument, argv[argc - 1], MAX_FILENAME_LEN);

      while ((opt = getopt_long(argc, argv, short_options, long_options,
&option_index)) != -1)
      {
            switch(opt)
            {
                  case 'h':
                  {
                        if (config->flag_help)
                        {
                              throw_error(OPTION_ERROR_MSG,
OPTION_ERROR_RET_VAL);
                        }
                        config->flag_help = true;
                  }
```

```
                break;

                case 'o':
                {
                     if (strlen(config->output_file) != 0)
                     {
                          throw_error(OPTION_ERROR_MSG,
OPTION_ERROR_RET_VAL);
                     }
                     strncpy(config->output_file,                optarg,
MAX_FILENAME_LEN);
                }
                break;

                case 'i':
                {
                     if (strlen(config->input_file) != 0)
                     {
                          throw_error(OPTION_ERROR_MSG,
OPTION_ERROR_RET_VAL);
                     }
                     strncpy(config->input_file,                 optarg,
MAX_FILENAME_LEN);
                }
                break;

                case INFO_OPT:
                {
                     if (config->flag_info)
                     {
                          throw_error(OPTION_ERROR_MSG,
OPTION_ERROR_RET_VAL);
                     }
                     config->flag_info = true;
                }
                break;

                case LINE_OPT:
```

```
                {
                      if (big_flag_is_set(config))
                      {
                            throw_error(OPTION_ERROR_MSG,
OPTION_ERROR_RET_VAL);
                      }
                      config->flag_line = true;
                }
                break;

                case INVERSE_CIRCLE_OPT:
                {
                      if (big_flag_is_set(config))
                      {
                            throw_error(OPTION_ERROR_MSG,
OPTION_ERROR_RET_VAL);
                      }
                      config->flag_inverse_circle = true;
                }
                break;

                case TRIM_OPT:
                {
                      if (big_flag_is_set(config))
                      {
                            throw_error(OPTION_ERROR_MSG,
OPTION_ERROR_RET_VAL);
                      }
                      config->flag_trim = true;
                }
                break;

                case CONTRAST_OPT:
                {
                      if (big_flag_is_set(config))
                      {
                            throw_error(OPTION_ERROR_MSG,
OPTION_ERROR_RET_VAL);
```

```
                    }
                    config->flag_contrast = true;
                }
                break;

                case START_OPT:
                {
                    if (config->line->flag_start)
                    {
                        throw_error(OPTION_ERROR_MSG,
OPTION_ERROR_RET_VAL);
                    }
                    config->line->start = arg_to_coord(optarg);
                    config->line->flag_start = true;
                }
                break;

                case END_OPT:
                {
                    if (config->line->flag_end) {
                        throw_error(OPTION_ERROR_MSG,
OPTION_ERROR_RET_VAL);
                    }
                    config->line->end = arg_to_coord(optarg);
                    config->line->flag_end = true;
                }
                break;

                case COLOR_OPT:
                {
                    if (config->line->flag_color)
                    {
                        throw_error(OPTION_ERROR_MSG,
OPTION_ERROR_RET_VAL);
                    }
                    config->line->color = arg_to_rgb(optarg);
                    config->line->flag_color = true;
                }
```

```
                break;

                case THICKNESS_OPT:
                {
                     if (config->line->flag_thickness)
                     {
                          throw_error(OPTION_ERROR_MSG,
OPTION_ERROR_RET_VAL);
                     }
                     config->line->thickness = arg_to_int(optarg);
                     config->line->flag_thickness = true;
                }
                break;

                case CENTER_OPT:
                {
                     if (config->inverse_circle->flag_center)
                     {
                          throw_error(OPTION_ERROR_MSG,
OPTION_ERROR_RET_VAL);
                     }
                     config->inverse_circle->center                    =
arg_to_coord(optarg);
                     config->inverse_circle->flag_center = true;
                }
                break;

                case RADIUS_OPT:
                {
                     if (config->inverse_circle->flag_radius)
                     {
                          throw_error(OPTION_ERROR_MSG,
OPTION_ERROR_RET_VAL);
                     }
                     config->inverse_circle->radius                    =
arg_to_int(optarg);
                     config->inverse_circle->flag_radius = true;
                }
```

```
                break;

                case LEFT_UP_OPT:
                {
                     if (config->trim->flag_left_up)
                     {
                          throw_error(OPTION_ERROR_MSG,
OPTION_ERROR_RET_VAL);
                     }
                     config->trim->left_up = arg_to_coord(optarg);
                     config->trim->flag_left_up = true;
                }
                break;

                case RIGHT_DOWN_OPT:
                {
                     if (config->trim->flag_right_down)
                     {
                          throw_error(OPTION_ERROR_MSG,
OPTION_ERROR_RET_VAL);
                     }
                     config->trim->right_down = arg_to_coord(optarg);
                     config->trim->flag_right_down = true;
                }
                break;

                case ALPHA_OPT:
                {
                     if (config->contrast->flag_alpha)
                     {
                          throw_error(OPTION_ERROR_MSG,
OPTION_ERROR_RET_VAL);
                     }
                     config->contrast->alpha = arg_to_float(optarg);
                     config->contrast->flag_alpha = true;
                }
                break;
```

```
                case BETA_OPT:
                {
                      if (config->contrast->flag_beta)
                      {
                            throw_error(OPTION_ERROR_MSG,
OPTION_ERROR_RET_VAL);
                      }
                      config->contrast->beta = arg_to_int(optarg);
                      config->contrast->flag_beta = true;
                }
                break;

                case '?':
                default:
                {
                      throw_error(OPTION_ERROR_MSG,
OPTION_ERROR_RET_VAL);
                }
           }
           option_index = -1;
     }

     if (strlen(config->input_file) == 0)
     {
           if (optind == argc - 1 && strncmp(argv[optind], last_argument,
MAX_FILENAME_LEN + 1) == 0)
           {
                strcpy(config->input_file, last_argument);
           }
           else if (optind < argc)
           {
                throw_error(OPTION_ERROR_MSG, OPTION_ERROR_RET_VAL);
           }
     }

     return config;
}
```

```
#endif
```

Файл: cw.c

```
##include "cwstructures/cwstructures.h"
#include "cwbmp/cwbmp.h"
#include "cwdrawing/cwdrawing.h"
#include "cwoptions/cwoptions.h"


void help()
{
     printf("This is a bitmap image processing program\n");
     printf("It accepts the followng command line arguments:\n");
     printf("\t--line: draws a line between two points of set color and
thickness.\n");
     printf("\t\t--start x.y : sets the start at point (x, y);\n");
     printf("\t\t--end x.y : sets the end at (x, y);\n");
     printf("\t\t--color rrr.ggg.bbb : sets the color to an RGB with
corresponding components;\n");
     printf("\t\t--thickness t : sets the thickness to t, has to be a
positive number;\n");
     printf("\n");
     printf("\t--inverse_circle: inverts the pixels within a given
circle.\n");
     printf("\t\t--center x.y : sets the center at point (x, y);\n");
     printf("\t\t--radius r : sets the radius to r, has to be a positive
number;\n");
     printf("\n");
     printf("\t--trim: crops the image to a rectangle given by two
points;\n");
     printf("\t\t--left_up x.y : sets the upper left point at (x,
y);\n");
     printf("\t\t--right_down x.y : sets the down right point at (x,
y);\n");
}

int main(int argc, char **argv) {
    printf("Course work for option 4.6, created by Artem Samokrutov\n");
```

```
struct Config *config = get_options(argc, argv);

 if (config->flag_help == 1)
 {
      help();
      return 0;
 }

 if (config->flag_info == 1)
 {
      if (big_flag_is_set(config) ||
           strlen(config->output_file) != 0)
      {
           throw_error(CONFIG_ERROR_MSG, CONFIG_ERROR_RET_VAL);
      }

      info(config->input_file);
      return 0;
 }

 if (strlen(config->input_file) == 0)
 {
      throw_error(CONFIG_ERROR_MSG, CONFIG_ERROR_RET_VAL);
 }

 if (strcmp(config->input_file, config->output_file) == 0 ||
     strcmp(config->input_file, "./out.bmp") == 0)
 {
      throw_error(CONFIG_ERROR_MSG, CONFIG_ERROR_RET_VAL);
 }

 Bitmap_image_t *image = load_bmp(config->input_file);

 if (config->flag_line)
 {
      draw_line(image,
                  config->line->start,
                  config->line->end,
```

```
                    config->line->color,
                    config->line->thickness);
    }

    if (config->flag_inverse_circle)
    {
        invert_circle(image,
                    config->inverse_circle->center,
                        config->inverse_circle->radius);
    }

    if (config->flag_trim)
    {
        crop(image,
            config->trim->left_up,
             config->trim->right_down);
    }

    if (config->flag_contrast)
    {
        contrast(image,
                config->contrast->alpha,
                   config->contrast->beta);
    }

    if (strlen(config->output_file) == 0)
    {
        strcpy(config->output_file, "./out.bmp");
    }

    save_bmp(image, config->output_file);
    free_bmp(image);

  destroy_config(config);

  return 0;
}
```

Файл: Makefile

```
TARGET = cw
CC = gcc

LIBS = ./bmp ./drawing ./error ./options ./structures

CFLAGS = -Wall -std=gnu99 $(foreach lib, $(LIBS), -I$(lib))

CORE = $(wildcard *.c) $(wildcard **/*.c)
OBJ = $(patsubst */%.c, $(PREF_BIN)%.o, $(CORE))

all : $(TARGET)

$(TARGET) : $(OBJ)
	$(CC) $(CFLAGS) $^ -o $@

clean :
	rm -rf $(TARGET)
```

ПРИЛОЖЕНИЕ В
ПРИМЕРЫ РАБОТЫ ПРОГРАММЫ

Рисование отрезка: исходное изображение — рис. 1, обработанное — рис. 2.

Входные данные: `--line --input test1.bmp --output out1.bmp --start 290.670 --end 440.670 --color 0.0.0 --thickness 60`

Рисунок 1. Исходное изображение — test1.bmp

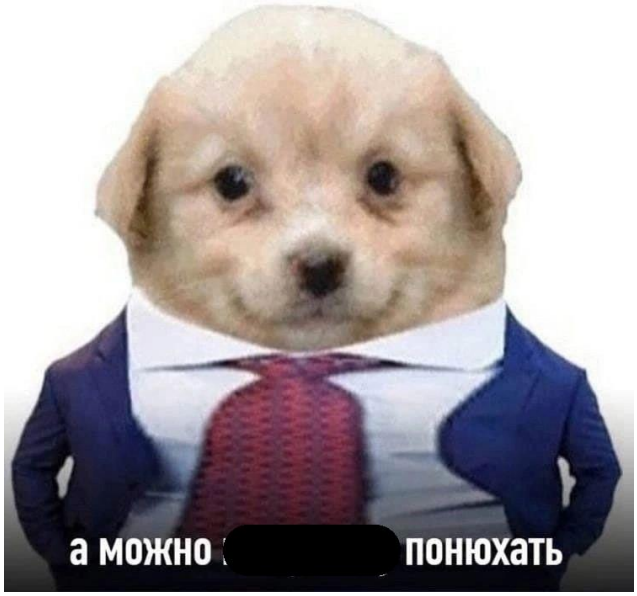

Рисунок 2. Обработанное изображение — out1.bmp

Входные данные: `-i test2.bmp -o out2.bmp --inverse_circle --center 100.425 --radius 30`

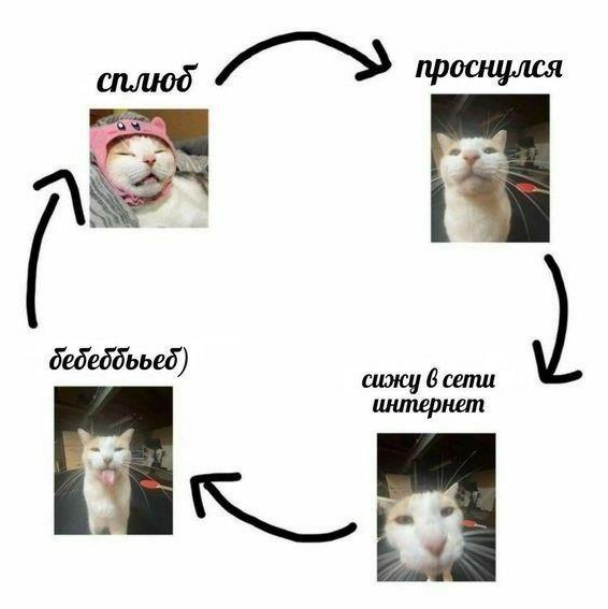

Рисунок 3. Исходное изображение — test2.bmp

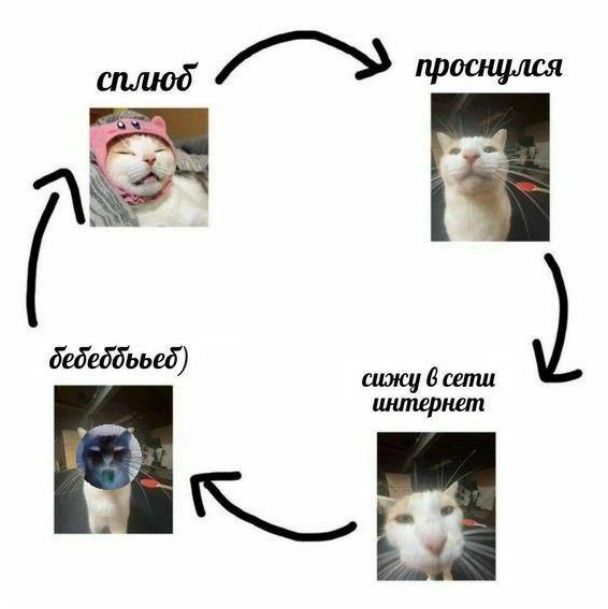

Рисунок 4. Обработанное изображение — out2.bmp

Входные данные: `-i test3.bmp --output out3.bmp --left_up 450-100 --right_down 700.250 --trim`

Рисунок 5. Исходное изображение — test3.bmp


Рисунок 6 Обработанное изображение — out3.bmp