**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**](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 С, 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 | 7 |
| 1. | Считывание и обработка флагов | 8 |
| 2. | Обработка BMP файла | 9 |
| 3.  3.1  3.2  3.3  4 | Обработка изображения  Рисование отрезка  Инверсия цветов в заданной окружности  Обрезка изображения  Файл cw.c | 10  10  10  11  12 |
| 5 | Makefile | 13 |
|  | Заключение | 14 |
|  | Список использованных источников | 15 |
|  | Приложение А. Примеры работы программы  Приложение B. Исходный код программы | 16  59 |

**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 B.

**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. Р. 25–30

**приложение B**

**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 B**

**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 . Исходное изображение — test1.bmp

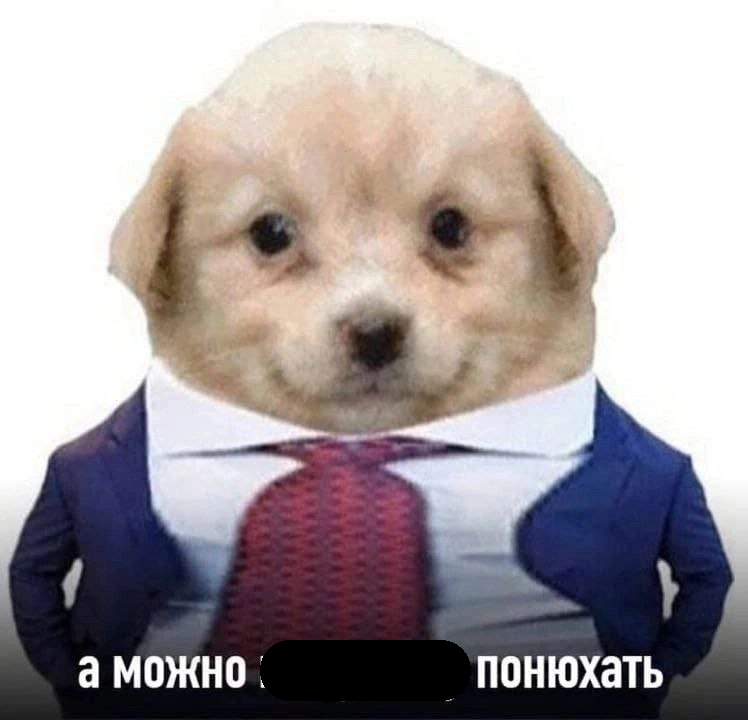


Рисунок 2. Обработанное изображение — out1.bmp

Входные данные: -i test2.bmp -o out2.bmp --inverse\_circle --center 100.425 --radius 30



Рисунок 3. Исходное изображение — test2.bmp

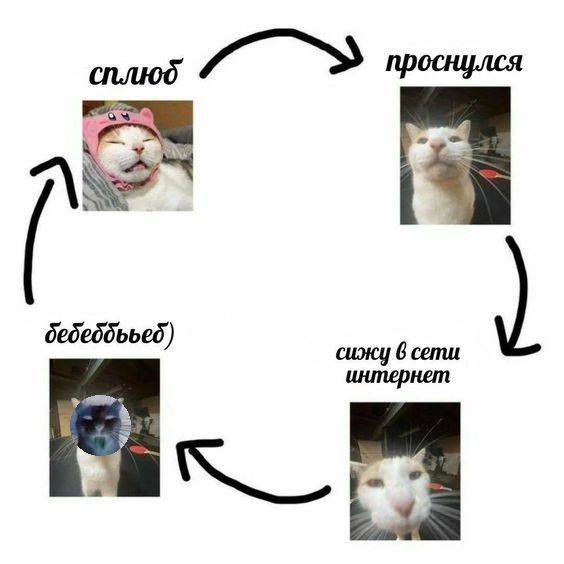


Рисунок 4. Обработанное изображение — out2.bmp

Входные данные: -i test3.bmp --output out3.bmp --left\_up 450-100 --right\_down 700.250 --trim

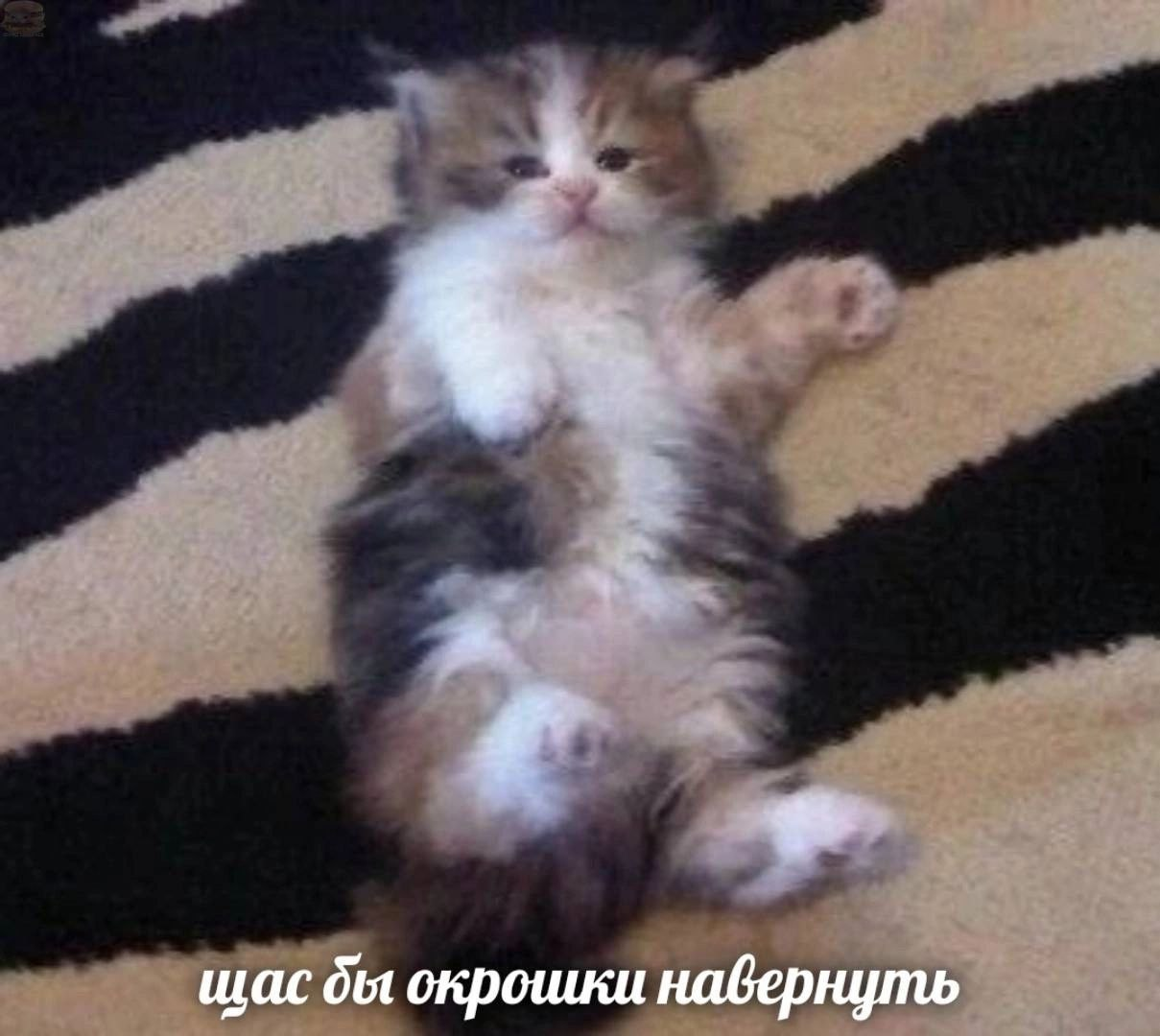


Рисунок 5. Исходное изображение — test3.bmp



Рисунок 6 Обработанное изображение — out3.bmp