**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

Курсовая РАБОТА

**по дисциплине «Программирование»**

Тема: Обработка BMP изображения.

|  |  |  |
| --- | --- | --- |
| Студентка гр. 3343 |  | Ермолаева В. А. |
| Преподаватель |  | Государкин Я. С. |

Санкт-Петербург

2024

**ЗАДАНИЕ**

**на курсовую работу**

|  |  |  |
| --- | --- | --- |
| Студентка Ермолаева В. А. | | |
| Группа 3343 | | |
| Тема работы: Обработка BMP изображения. | | |
| Исходные данные:  Программа обязательно должна иметь CLI (опционально дополнительное использование GUI).  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). | | |
| Содержание пояснительной записки:   * Аннотация * Содержание * Введение * Описание задачи и требований * Описание структур * Описание реализованных функций * Тестирование * Заключение * Приложение А. Исходный код программы | | |
| Предполагаемый объем пояснительной записки:  Не менее 35 страниц. | | |
| Дата выдачи задания: 18.03.2024 | | |
| Дата сдачи реферата: 29.05.2024 | | |
| Дата защиты реферата: 29.05.2024 | | |
| Студентка |  | Ермолаева В. А. |
| Преподаватель |  | Государкин Я. С. |

**Аннотация**

Основным поставленным заданием курсовой работы является разработка программы, направленной на обработку BMP изображения, поданного на вход пользователем. Производится реализация нескольких структур для хранения информации об изображении и его дальнейшей обработки. В программу заложено три базовые функции, включающие в себя рисование квадрата с диагоналями, фильтр rgb-компонент и поворот изображения или его части на 90/180/270 градусов. Помимо этого пользователю доступны дополнительные функции, выводящие информацию о программе или поданном на вход изображении. Взаимодействие пользователя с программой осуществляется при помощи интерфейса командной строки.

**Содержание**

|  |  |  |
| --- | --- | --- |
|  | Введение | 6 |
| 1. | Описание задачи и требований | 7 |
| 2. | Описание структур | 9 |
| 3. | Описание реализованных функций | 10 |
| 4. | Тестирование | 13 |
|  | Заключение | 15 |
|  | Приложение А. Исходный код программы | 16 |

**введение**

Целью работы является разработка программы на языке Си, направленной на обработку изображений формата BMP (от англ. Bitmap Picture). В основные задачи входят изучение и практическое применение навыков, необходимых для:

* реализации структур, необходимых для хранения и обработки изображений
* успешного применения функционала библиотек языка Си для работы с изображениями к практическим задачам
* обработке ошибок, которые могут возникнуть в процессе исполнения программы
* реализации методов обработки изображений согласно поставленным задачам.

Среди задач данной курсовой работы имеется реализация интерфейса командной строки с использованием getopt, своевременное уведомление пользователя о некорректно введенных данных, а также обеспечение работоспособности программы на неожиданных данных, введенных пользователем.

**Описание задачи и требований**

Программа должна иметь следующую функции по обработке изображений:

* (1) Рисование квадрата с диагоналями. Флаг для выполнения данной операции: `--squared\_lines`. Квадрат определяется:
  + Координатами левого верхнего угла. Флаг `--left\_up`, значение задаётся в формате `left.up`, где left – координата по x, up – координата по y
  + Размером стороны. Флаг `--side\_size`. На вход принимает число больше 0
  + Толщиной линий. Флаг `--thickness`. На вход принимает число больше 0
  + Цветом линий. Флаг `--color` (цвет задаётся строкой `rrr.ggg.bbb`, где rrr/ggg/bbb – числа, задающие цветовую компоненту. пример `--color 255.0.0` задаёт красный цвет)
  + Может быть залит или нет (диагонали располагаются “поверх” заливки). Флаг `--fill`. Работает как бинарное значение: флага нет – false , флаг есть – true.
  + Цветом которым он залит, если пользователем выбран залитый. Флаг `--fill\_color` (работает аналогично флагу `--color`)
* (2) Фильтр rgb-компонент. Флаг для выполнения данной операции: `--rgbfilter`. Этот инструмент должен позволять для всего изображения либо установить в диапазоне от 0 до 255 значение заданной компоненты. Функционал определяется
  + Какую компоненту требуется изменить. Флаг `--component\_name`. Возможные значения `red`, `green` и `blue`.
  + В какой значение ее требуется изменить. Флаг `--component\_value`. Принимает значение в виде числа от 0 до 255
* (3) Поворот изображения (части) на 90/180/270 градусов. Флаг для выполнения данной операции: `--rotate`. Функционал определяется
  + Координатами левого верхнего угла области. Флаг `--left\_up`, значение задаётся в формате `left.up`, где left – координата по x, up – координата по y
  + Координатами правого нижнего угла области. Флаг `--right\_down`, значение задаётся в формате `right.down`, где right – координата по x, down – координата по y
  + Углом поворота. Флаг `--angle`, возможные значения: `90`, `180`, `270`

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

**Описание структур**

Программа включает в себя пять структур, необходимых для обработки изображений.

* BitmapFileHeader: описывает заголовок BMP файла. Она используется для идентификации файла как BMP, определения его размера и указания смещения, где начинается массив пикселей изображения.
* BitmapInfoHeader: храненит информацию об изображении. Содержит сведения о длине изображения, его ширине, глубине цвета, количестве цветов в цветовой таблице и т. д.
* RGB: хранит информацию об интенсивности красной, синей и зеленой компонент пикселя.
* Error: используется для более удобной обработки ошибок, хранит значение, указывающее на наличие ошибки и сообщение, которое следует вывести при ее возникновении.
* Options: хранит считанные значения аргументов, поданных в программу пользователем.

**Описание реализованных функций**

Программа состоит из функций, описание которых представлено ниже:

* int main(): главная функция программы, выводит базовую информацию о курсовой работе и вызывает функцию обработки аргументов, введенных пользователем.
* void process\_args(int argc, char\* argv[]): обрабатывает введенные пользователем аргументы, проводит предварительную обработку ошибок и при их отсутствии вызывает функцию обработки изображения.
* Error process\_image(char\* filename, Options opts): функция, которая производит считывание изображения, его преобразование и запись нового файла. На этом этапе проверяется большая часть ошибок. Возвращает структуру ошибки, хранящую информацию о том, возникли ли в процессе изменения изображения ошибки, и если да, то какие.
* int check\_options(Options\* opts, Error\* err): проверяет аргументы, введенные пользователем на корректность.
* int check\_int(char\* str): проверяет, является ли переданная строка целым числом.
* void rotate(RGB\*\* rgb, int height, int width, int x0, int y0, int x1, int y1, int angle): поворачивает изображение (часть) на 90/180/270 градусов. В случае, если повернутая часть изображения выходит за границы изображения, рисуется только вместившаяся часть.
* void swap\_channels(RGB\*\* rgb, int height, int width, char\* component\_name, unsigned int component\_value): применяет к изображению фильтр rgb-компонент.
* void draw\_square(RGB\*\* rgb, BitmapInfoHeader\* info, int x, int y, int side\_size, int thickness, char\* colour, int fill, char\* fill\_colour): рисует по указанным координатам квадрат, который может быть залит или нет. В случае, если квадрат выходит за границы изображения, отрисовывается только та часть квадрата, которая на него попадает.
* void get\_coordinates(char\* left\_up, int\* x, int\* y): записывает в переданные адреса переменных извлеченные координаты.
* RGB get\_colour(char\* colour): возвращает пиксель, содержащий цветовые компоненты, извлеченные из переданной строки.
* int check\_coordinate\_format(char\* coord): проверяет переданную строку с координатами на корректность.
* int check\_color(char\* color): проверяет переданную строку с цветом на корректность.
* void line(RGB\*\* rgb, BitmapInfoHeader\* info, int x0, int y0, int x1, int y1, int thickness, RGB pixel): рисует на изображении линию по алгоритму Брезенхэма.
* void circle(RGB\*\* rgb, BitmapInfoHeader\* info, int radius, int x1, int y1, RGB pixel): рисует на изображении окружность по алгоритму Брезенхэма.
* int check\_coordinates(int x, int y, int height, int width): проверяет координаты на то, принадлежат ли они изображению.
* void paint\_pixel(RGB\* pixel, int b, int g, int r): записывает в пиксель указанные цветовые компоненты.
* void write\_bmp(char\* file\_name, RGB\*\* rgb, BitmapFileHeader\* file, BitmapInfoHeader\* info): записывает в файл изображение.
* RGB\*\* read\_bmp(char\* file\_name, BitmapFileHeader\* file, BitmapInfoHeader\* info, Error\* err): считывает изображение из файла.
* void check\_bmp(BitmapFileHeader\* file, Error\* err): проверяет файл с изображением на соответствие BMP формату.
* void delete\_RGB(RGB\*\* rgb, int height): очищает память, выделенную под хранение изображения.
* RGB\*\* create\_RGB(int height, int width): выделяет память под хранение изображения.
* void initialize\_opts(Options\* opts): устанавливает в опции значения по умолчанию.
* void print\_info\_header(BitmapInfoHeader\* header): выводит значения всех полей структуры BitmapInfoHeader, т. е. подробную информацию об изображении.
* void print\_file\_header(BitmapFileHeader\* header): выводит значения всех полей структуры BitmapFileHeader, т. е. информацию о файле.
* void print\_help(): выводит подробную информацию о программе, ее флагах и принимаемых значениях.
* char\* strdup (const char\* s): копирует строку.

Более подробно с функциями и их содержанием можно ознакомиться в приложении А.

**тестирование**

Исходное изображение:

Рис. 1: Исходное изображение

1. Тестирование функции squared\_lines.

Аргументы для запуска: --fill\_color 56.98.217 --thickness 11 --side\_size 362 --color 110.35.109 --output ./output.bmp --fill --input ./food.bmp --left\_up 199.347 —squared\_lines

Рис. 2: Тестирование функции squared\_lines

1. Тестирование функции rgbfilter.

Аргументы для запуска: --component\_value 69 --output ./output.bmp --rgbfilter --input ./food.bmp --component\_name "green"

1. Тестирование функции rgbfilter.

Рис. 3: Результат работы функции rgbfilter

Аргументы для запуска: --angle 90 --left\_up 200.100 --right\_down 400.400 --input ./food.bmp --output ./output.bmp —rotate

Рис. 4: Результат работы функции rotate

**заключение**

В ходе выполнения курсовой работы были изучены и применены на практике знания о формате BMP, были освоены навыки работы с изображениями формата BMP на языке Си, их изменением, созданием и считыванием. Были обработаны неожиданный ввод пользователя, потенциальные ошибки, которые могут возникнуть в процессе выполнения программы, а также реализован интерфейс командной строки с использованием getopt.

**приложение А**

**Исходный код программы**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <getopt.h>

#include <ctype.h>

typedef struct {

int value;

char\* message;

} Error;

typedef struct {

int rgb\_filter\_flag;

int squared\_lines\_flag;

int fill\_flag;

char\* output\_name;

char\* input\_name;

int rotate\_flag;

int info\_flag;

char\* component\_name;

char\* component\_value;

char\* left\_up;

char\* right\_down;

char\* side\_size;

char\* thickness;

char\* angle;

char\* color;

char\* fill\_color;

} Options;

#pragma pack (push, 1)

typedef struct {

unsigned short signature;

unsigned int filesize;

unsigned short reserved1;

unsigned short reserved2;

unsigned int pixelArrOffset;

} BitmapFileHeader;

typedef struct {

unsigned int headerSize;

unsigned int width;

unsigned int height;

unsigned short planes;

unsigned short bitsPerPixel;

unsigned int compression;

unsigned int imageSize;

unsigned int xPixelsPerMeter;

unsigned int yPixelsPerMeter;

unsigned int coloursIncolourTable;

unsigned int importantcolourCount;

} BitmapInfoHeader;

typedef struct {

unsigned char b;

unsigned char g;

unsigned char r;

} RGB;

#pragma pack(pop)

char\* strdup (const char\* s) {

size\_t slen = strlen(s);

char\* result = malloc(slen + 1);

if(result == NULL) return NULL;

memcpy(result, s, slen + 1);

return result;

}

void print\_help() {

printf("Флаги, используемые в программе:\n"

"--help: Выводит справочную информацию.\n"

"--input: Задаёт имя входного изображения.\n"

"--output: Задаёт имя выходного изображения.\n"

"--info: Вывод информации об изображении.\n"

"--squared\_lines: Рисование квадрата с диагоналями.\n"

"--side\_size: Размер стороны. Принимает число больше 0.\n"

"--thickness: Толщина линии. Принимает число больше 0.\n"

"--color: Цвет линии, задаётся строкой \'rrr.ggg.bbb\', где rrr/ggg/bbb – числа, задающие цветовую компоненту.\n"

"--fill: Может быть залит или нет. Флаг работает как бинарное значение: флага нет – false , флаг есть – true.\n"

"--fill\_color: Цветом которым он залит, если пользователем выбран залитый. работает аналогично флагу \'--color\'.\n"

"--rgbfilter: Фильтр rgb-компонент.\n"

"--component\_name: Какую компоненту требуется изменить. Возможные значения \'red\', \'green\' и \'blue\'\n"

"--component\_value: В какое значение ее требуется изменить. Принимает значение в виде числа от 0 до 255.\n"

"--rotate: Поворот изображения (части) на 90/180/270 градусов.\n"

"--left\_up: Координаты левого верхнего угла области. Значение задаётся в формате \'left.up\', где left – координата по x, up – координата по y.\n"

"--right\_down: Координаты правого нижнего угла области. Значение задаётся в формате \'right.down\', где right – координата по x, down – координата по y.\n"

"--angle: Угол поворота. Возможные значения: \'90\', \'180\', \'270\'.\n");

}

void print\_file\_header(BitmapFileHeader\* header) {

printf("signature:\t%x (%hu)\n", header->signature, header->signature);

printf("filesize:\t%x (%u)\n", header->filesize, header->filesize);

printf("reserved1:\t%x (%hu)\n", header->reserved1, header->reserved1);

printf("reserved2:\t%x (%hu)\n", header->reserved2, header->reserved2);

printf("pixelArrOffset:\t%x (%u)\n", header->pixelArrOffset, header->pixelArrOffset);

}

void print\_info\_header(BitmapInfoHeader\* header) {

printf("headerSize:\t%x (%u)\n", header->headerSize, header->headerSize);

printf("width: \t%x (%u)\n", header->width, header->width);

printf("height: \t%x (%u)\n", header->height, header->height);

printf("planes: \t%x (%hu)\n", header->planes, header->planes);

printf("bitsPerPixel:\t%x (%hu)\n", header->bitsPerPixel, header->bitsPerPixel);

printf("compression:\t%x (%u)\n", header->compression, header->compression);

printf("imageSize:\t%x (%u)\n", header->imageSize, header->imageSize);

printf("xPixelsPerMeter:\t%x (%u)\n", header->xPixelsPerMeter, header->xPixelsPerMeter);

printf("yPixelsPerMeter:\t%x (%u)\n", header->yPixelsPerMeter, header->yPixelsPerMeter);

printf("coloursIncolourTable:\t%x (%u)\n", header->coloursIncolourTable, header->coloursIncolourTable);

printf("importantcolourCount:\t%x (%u)\n", header->importantcolourCount, header->importantcolourCount);

}

void initialize\_opts(Options\* opts) {

opts->rgb\_filter\_flag = 0;

opts->squared\_lines\_flag = 0;

opts->fill\_flag = 0;

opts->rotate\_flag = 0;

opts->info\_flag = 0;

opts->input\_name = "";

opts->output\_name = "out.bmp";

opts->fill\_color = NULL;

opts->component\_name = "";

opts->component\_value = NULL;

opts->left\_up = NULL;

opts->right\_down = NULL;

opts->side\_size = NULL;

opts->thickness = NULL;

opts->angle = NULL;

opts->color = NULL;

opts->fill\_color = NULL;

}

RGB\*\* create\_RGB(int height, int width) {

RGB\*\* rgb = malloc(height \* sizeof(RGB \*));

for(int i = 0; i < height; i++)

rgb[i] = malloc(width \* sizeof(RGB) + ((4 - (width \* sizeof(RGB)) % 4) % 4));

return rgb;

}

void delete\_RGB(RGB\*\* rgb, int height) {

for(int i = 0; i < height; i++)

free(rgb[i]);

free(rgb);

}

void check\_bmp(BitmapFileHeader\* file, Error\* err) {

if (file->signature != 0x4d42 && file->signature != 0x424d) {

err->value = 1;

err->message = "Given file is not a BMP.";

}

}

RGB\*\* read\_bmp(char\* file\_name, BitmapFileHeader\* file, BitmapInfoHeader\* info, Error\* err) {

FILE \*f = fopen(file\_name, "rb");

if (!f) {

err->value = 1;

err->message = "Couldn't open given file.";

return create\_RGB(0, 0);

}

fread(file, 1, sizeof(BitmapFileHeader), f);

fread(info, 1, sizeof(BitmapInfoHeader), f);

check\_bmp(file, err);

if (err->value) {

fclose(f);

return create\_RGB(0, 0);

}

fseek(f, file->pixelArrOffset, SEEK\_SET);

int height = info->height;

int width = info->width;

RGB\*\* rgb = create\_RGB(height, width);

for(int i = 0; i < height; i++)

fread(rgb[i], 1, width \* sizeof(RGB) + ((4 - (info->width \* sizeof(RGB)) % 4) % 4), f);

fclose(f);

return rgb;

}

void write\_bmp(char\* file\_name, RGB\*\* rgb, BitmapFileHeader\* file, BitmapInfoHeader\* info) {

FILE\* f = fopen(file\_name, "wb");

fwrite(file, 1, sizeof(BitmapFileHeader), f);

fwrite(info, 1, sizeof(BitmapInfoHeader), f);

fseek(f, file->pixelArrOffset, SEEK\_SET);

for(int i = 0; i < info->height; i++)

fwrite(rgb[i], 1, info->width \* sizeof(RGB) + ((4 - (info->width \* sizeof(RGB)) % 4) % 4), f);

fclose(f);

}

void paint\_pixel(RGB\* pixel, int b, int g, int r) {

pixel->b = b;

pixel->g = g;

pixel->r = r;

}

int check\_coordinates(int x, int y, int height, int width) {

if (x >= 0 && y >= 0 && x < width && y < height) return 1;

return 0;

}

void circle(RGB\*\* rgb, BitmapInfoHeader\* info, int radius, int x1, int y1, RGB pixel) {

int x = 0;

int y = radius;

int delta = 1 - 2 \* radius;

int error = 0;

while (y >= x) {

for (int i = x1 - x; i <= x1 + x; i++) {

if (check\_coordinates(i, y1+y, info->height, info->width)) rgb[y1 + y][i] = pixel;

if (check\_coordinates(i, y1-y, info->height, info->width))rgb[y1 - y][i] = pixel;

}

for (int i = x1 - y; i <= x1 + y; i++) {

if (check\_coordinates(i, y1+x, info->height, info->width))rgb[y1 + x][i] = pixel;

if (check\_coordinates(i, y1-x, info->height, info->width))rgb[y1 - x][i] = pixel;

}

error = 2 \* (delta + y) - 1;

if ((delta < 0) && (error <= 0)) {

delta += 2 \* ++x + 1;

continue;

}

if ((delta > 0) && (error > 0)) {

delta -= 2 \* --y + 1;

continue;

}

delta += 2 \* (++x - --y);

}

}

void line(RGB\*\* rgb, BitmapInfoHeader\* info, int x0, int y0, int x1, int y1, int thickness, RGB pixel) {

int dx = abs(x1 - x0);

int dy = abs(y1 - y0);

int sx = x0 < x1 ? 1 : -1;

int sy = y0 < y1 ? 1 : -1;

int err = (dx > dy ? dx : -dy) / 2;

while(1) {

circle(rgb, info, (thickness + 1)/2, x0, y0, pixel);

if (x0 == x1 && y0 == y1)

break;

if (err > -dx) {

err -= dy;

x0 += sx;

}

if (err < dy) {

err += dx;

y0 += sy;

}

}

}

int check\_color(char\* color) {

int dot\_count = 0;

int num = 0;

for (int i = 0; i < strlen(color); i++) {

if (!isdigit(color[i]) && color[i] != '.') return 0;

if (color[i] == '.') dot\_count++;

}

if (dot\_count != 2)

return 0;

if (color[0] == '.' || color[strlen(color) - 1] == '.')

return 0;

char\* token = strtok(strdup(color), ".");

for (int i = 0; i < 3; i++) {

if (token == NULL || !isdigit(token[0])) return 0;

num = atoi(token);

if (num < 0 || num > 255) return 0;

token = strtok(NULL, ".");

}

return 1;

}

int check\_coordinate\_format(char\* coord) {

int dot\_count = 0;

for (int i = 0; i < strlen(coord); i++) {

if (!isdigit(coord[i]) && coord[i] != '.') return 0;

if (coord[i] == '.') dot\_count++;

}

if (dot\_count != 1)

return 0;

if (coord[0] == '.' || coord[strlen(coord) - 1] == '.')

return 0;

return 1;

}

RGB get\_colour(char\* colour) {

RGB pixel;

char\* str = strdup(colour);

char\* end;

int red = strtol(str, &end, 10);

int green = strtol(end + 1, &end, 10);

int blue = strtol(end + 1, NULL, 10);

paint\_pixel(&pixel, blue, green, red);

return pixel;

}

void get\_coordinates(char\* left\_up, int\* x, int\* y) {

char\* str = strdup(left\_up);

char\* end;

\*x = strtol(str, &end, 10);

\*y = strtol(end + 1, NULL, 10);

}

void draw\_square(RGB\*\* rgb, BitmapInfoHeader\* info, int x, int y, int side\_size, int thickness, char\* colour, int fill, char\* fill\_colour) {

RGB pixel = get\_colour(colour);

if (fill) {

RGB pixel\_filled = get\_colour(fill\_colour);

for (int i = y+thickness/2; i < y+side\_size-thickness/2; i++) {

for (int j = x+thickness/2; j < x+side\_size-thickness/2; j++) {

if (check\_coordinates(j, i, info->height, info->width)) rgb[i][j] = pixel\_filled;

}

}

}

line(rgb, info, x, y, x + side\_size, y, thickness, pixel);

line(rgb, info, x, y, x, y + side\_size, thickness, pixel);

line(rgb, info, x, y + side\_size, x + side\_size, y + side\_size, thickness, pixel);

line(rgb, info, x + side\_size, y, x + side\_size, y + side\_size, thickness, pixel);

line(rgb, info, x, y, x + side\_size, y + side\_size, thickness, pixel);

line(rgb, info, x + side\_size, y, x, y + side\_size, thickness, pixel);

}

void swap\_channels(RGB\*\* rgb, int height, int width, char\* component\_name, unsigned int component\_value) {

for(int i = 0; i < height; i++) {

for(int j = 0; j < width; j++) {

if (!strcmp(component\_name, "green"))

rgb[i][j].g = component\_value;

else if (!strcmp(component\_name, "blue"))

rgb[i][j].b = component\_value;

else

rgb[i][j].r = component\_value;

}

}

}

void rotate(RGB\*\* rgb, int height, int width, int x0, int y0, int x1, int y1, int angle) {

RGB\*\* rgb\_rotated = create\_RGB(height, width);

for (int i = 0; i < height; i++) {

for (int j = 0; j < width; j++) {

rgb\_rotated[i][j] = rgb[i][j];

}

}

int xc = (x1 + x0) / 2; int yc = (y1 + y0) / 2;

int x; int y;

int xr; int yr;

for (int i = y0; i < y1; i++) {

for (int j = x0; j < x1; j++) {

switch(angle) {

case 90:

xr = j - xc; yr = i - yc;

x = xc - yr; y = yc + xr;

if (!((y1 + y0) % 2 == 0 && (x1 + x0) % 2 == 0)) {

if ((y1 + y0) % 2 == 0) y -= 1;

if ((y1 + y0) % 2 != 0) y += 1;

}

if ((x1 + x0) % 2 != 0) x -= 1;

if (check\_coordinates(j, i, height, width) && check\_coordinates(x, y, height, width))

rgb[y][x] = rgb\_rotated[i][j];

break;

case 180:

x = x1 - j - 1 + x0; y = y1 - i - 1 + y0;

if (check\_coordinates(j, i, height, width) && check\_coordinates(x, y, height, width))

rgb[i][j] = rgb\_rotated[y][x];

break;

case 270:

xr = j - xc; yr = i - yc;

x = xc + yr; y = yc - xr;

if ((y1 + y0) % 2 == 0) y -= 1;

if (check\_coordinates(j, i, height, width) && check\_coordinates(x, y, height, width))

rgb[y][x] = rgb\_rotated[i][j];

break;

}

}

}

delete\_RGB(rgb\_rotated, height);

}

int check\_int(char\* str) {

if (!str) return 0;

while (\*str) {

if (\*str < '0' || \*str > '9') return 0;

str++;

}

return 1;

}

int check\_options(Options\* opts, Error\* err) {

if (opts->rgb\_filter\_flag) {

if (!opts->component\_name || !opts->component\_value) {

err->value = 1;

err->message = "Some required flags are missing.";

return 1;

}

if (!opts->component\_name || (strcmp(opts->component\_name, "green") && strcmp(opts->component\_name, "blue") && strcmp(opts->component\_name, "red"))) {

err->value = 1;

err->message = "Incorrect component name, must be blue, red or green.";

return 1;

}

if (!check\_int(opts->component\_value)) {

err->value = 1;

err->message = "Incorrect component value, must be an integer.";

return 1;

}

if (atoi(opts->component\_value) < 0 || atoi(opts->component\_value) > 255) {

err->value = 1;

err->message = "Incorrect component value, must be between 0 and 255.";

return 1;

}

}

if (opts->squared\_lines\_flag) {

if (!opts->left\_up || !opts->side\_size || !opts->thickness || !opts->color || ((opts->fill\_flag && !opts->fill\_color))) {

err->value = 1;

err->message = "Some required flags are missing.";

return 1;

}

if(!check\_coordinate\_format(opts->left\_up)) {

err->value = 1;

err->message = "Given coordinates must be in format \"x.y\".";

return 1;

}

if (!check\_int(opts->side\_size) || !check\_int(opts->thickness)) {

err->value = 1;

err->message = "Incorrect side\_size or thickness values, must be integers.";

return 1;

}

if (atoi(opts->side\_size) < 0 || atoi(opts->thickness) < 0) {

err->value = 1;

err->message = "Numeric values must be positive.";

return 1;

}

if (!check\_color(opts->color) || (opts->fill\_flag && !check\_color(opts->fill\_color))) {

err->value = 1;

err->message = "Color must be given in format \"0.0.0\".";

return 1;

}

}

if (opts->rotate\_flag) {

if(!opts->left\_up || !opts->right\_down || !opts->angle) {

err->value = 1;

err->message = "Some required flags are missing.";

return 1;

}

if(!check\_coordinate\_format(opts->left\_up) || !check\_coordinate\_format(opts->right\_down)) {

err->value = 1;

err->message = "Given coordinates must be in format \"x.y\".";

return 1;

}

if (!check\_int(opts->angle)) {

err->value = 1;

err->message = "Incorrect angle value, must be an integer.";

return 1;

}

if (atoi(opts->angle) % 90 != 0 || atoi(opts->angle) < 0 || atoi(opts->angle) > 270) {

err->value = 1;

err->message = "Rotation angle must be 90, 180 or 270 degrees.";

return 1;

}

}

return 0;

}

Error process\_image(char\* filename, Options opts) {

BitmapFileHeader file;

BitmapInfoHeader info;

Error err;

err.value = 0;

RGB\*\* rgb = read\_bmp(filename, &file, &info, &err);

if (err.value == 1) return err;

if (opts.info\_flag) {

print\_file\_header(&file);

print\_info\_header(&info);

}

if (opts.rgb\_filter\_flag) {

if(!check\_options(&opts, &err)) swap\_channels(rgb, info.height, info.width, opts.component\_name, atoi(opts.component\_value));

}

if (opts.squared\_lines\_flag) {

if (!check\_options(&opts, &err)) {

int x; int y;

get\_coordinates(opts.left\_up, &x, &y);

if (opts.fill\_flag)

draw\_square(rgb, &info, x, info.height - y - atoi(opts.side\_size), atoi(opts.side\_size), atoi(opts.thickness), opts.color, 1, opts.fill\_color);

else

draw\_square(rgb, &info, x, info.height - y - atoi(opts.side\_size), atoi(opts.side\_size), atoi(opts.thickness), opts.color, 0, NULL);

}

}

if (opts.rotate\_flag) {

if(!check\_options(&opts, &err)) {

int x0, y0, x1, y1;

get\_coordinates(opts.left\_up, &x0, &y0);

get\_coordinates(opts.right\_down, &x1, &y1);

int side\_size = abs(y1 - y0);

if (y1 < y0 || x1 < x0) {

int temp\_x = x0; int temp\_y = y0;

x0 = x1; y0 = y1;

x1 = temp\_x; y1 = temp\_y;

}

rotate(rgb, info.height, info.width, x0, info.height - y0 - side\_size, x1, info.height - y1 + side\_size, atoi(opts.angle));

}

}

write\_bmp(opts.output\_name, rgb, &file, &info);

delete\_RGB(rgb, info.height);

return err;

}

int check\_file(char\* filename) {

BitmapFileHeader file;

BitmapInfoHeader info;

Error err;

err.value = 0;

RGB\*\* rgb = read\_bmp(filename, &file, &info, &err);

if (err.value == 0) {

delete\_RGB(rgb, info.height);

return 1;

}

delete\_RGB(rgb, info.height);

return 0;

}

void process\_args(int argc, char\* argv[]) {

struct option long\_options[] = {

{"help", no\_argument, 0, 'h'},

{"output", required\_argument, 0, 'o'},

{"info", no\_argument, 0, 5},

{"input", required\_argument, 0, 'i'},

{"rgbfilter", no\_argument, 0, 1},

{"component\_name", required\_argument, 0, 'n'},

{"component\_value", required\_argument, 0, 'v'},

{"squared\_lines", no\_argument, 0, 2},

{"left\_up", required\_argument, 0, 'l'},

{"side\_size", required\_argument, 0, 's'},

{"thickness", required\_argument, 0, 't'},

{"color", required\_argument, 0, 'c'},

{"fill", no\_argument, 0, 3},

{"fill\_color", required\_argument, 0, 'f'},

{"rotate", no\_argument, 0, 4},

{"right\_down", required\_argument, 0, 'r'},

{"angle", required\_argument, 0, 'a'},

{0, 0, 0, 0}

};

char\* extra\_args[argc];

int extra\_args\_count = 0;

Options opts;

initialize\_opts(&opts);

int opt;

opterr = 0;

while ((opt = getopt\_long(argc, argv, "hi:o:n:v:l:s:t:c:f:r:a:", long\_options, NULL)) != -1) {

switch (opt) {

case 1:

opts.rgb\_filter\_flag = 1;

break;

case 2:

opts.squared\_lines\_flag = 1;

break;

case 3:

opts.fill\_flag = 1;

break;

case 4:

opts.rotate\_flag = 1;

break;

case 5:

opts.info\_flag = 1;

break;

case 'n':

opts.component\_name = optarg;

break;

case 'v':

opts.component\_value = optarg;

break;

case 'l':

opts.left\_up = optarg;

break;

case 's':

opts.side\_size = optarg;

break;

case 't':

opts.thickness = optarg;

break;

case 'c':

opts.color = optarg;

break;

case 'f':

opts.fill\_color = optarg;

break;

case 'r':

opts.right\_down = optarg;

break;

case 'a':

opts.angle = optarg;

break;

case 'h':

print\_help();

break;

case 'o':

opts.output\_name = optarg;

break;

case 'i':

opts.input\_name = optarg;

break;

case '?':

extra\_args[extra\_args\_count++] = argv[optind - 1];

break;

}

}

while (optind < argc) {

extra\_args[extra\_args\_count++] = argv[optind++];

}

Error err;

err.value = 0;

if (optind == 1) print\_help();

if (extra\_args\_count > 0) {

for (int i = 0; i < extra\_args\_count; i++) {

if (check\_file(extra\_args[i])) {

opts.input\_name = extra\_args[i];

}

else printf("Extra argument \'%s\' will be ignored.\n", extra\_args[i]);

}

}

if (strstr(opts.input\_name, opts.output\_name)) {

err.value = 1;

err.message = "Can't use the same input and output file names.";

}

if (!err.value) err = process\_image(opts.input\_name, opts);

if (err.value) printf("%s\n", err.message);

}

int main(int argc, char\* argv[]) {

printf("Course work for option 4.12, created by Vera Ermolaeva.\n");

process\_args(argc, argv);

return 0;

}