**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МОЭВМ**

КУРСОВАЯ РАБОТА

**по дисциплине «Программирование»**

**Тема: Обработка BMP изображения**

|  |  |  |
| --- | --- | --- |
| Студент гр. 3343 |  | Отмахов Д. В. |
| Преподаватель |  | Государкин Я.С. |

Санкт-Петербург

2024

## ЗАДАНИЕ

## НА КУРСОВУЮ РАБОТУ

Студент: Отмахов Данил

Группа: 3343

Тема: Обработка BMP изображения

Условия задания (Вариант 4.2):

Программа должна иметь следующую функции по обработке изображений:

1. Заменяет все пиксели одного заданного цвета на другой цвет. Флаг для выполнения данной операции: *`--color\_replace*`. Функционал определяется:

* Цвет, который требуется заменить. Флаг *`--old\_color*` (цвет задаётся строкой `*rrr.ggg.bbb*`, где rrr/ggg/bbb – числа, задающие цветовую компоненту. пример *`--old\_color 255.0.0*` задаёт красный цвет);
* Цвет на который требуется заменить. Флаг `*--new\_color*` (работает аналогично флагу *`--old\_color*`).

1. 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.

1. Разделяет изображение на N\*M частей. Флаг для выполнения данной операции: `*--split*`. Реализация: провести линии заданной толщины. Функционал определяется:

* Количество частей по “оси” Y. Флаг `*--number\_x*`. На вход принимает число больше 1;
* Количество частей по “оси” X. Флаг `*--number\_y*`. На вход принимает число больше 1;
* Толщина линии. Флаг `*--thickness*`. На вход принимает число больше 0.
* Цвет линии. Флаг `*--color*` (цвет задаётся строкой `*rrr.ggg.bbb*`, где rrr/ggg/bbb – числа, задающие цветовую компоненту. пример `*--color 255.0.0*` задаёт красный цвет).

Дата выдачи задания: 18.03.2024

Дата сдачи реферата: 15.05.2024

Дата защиты реферата: 15.05.2024

**АННОТАЦИЯ**

В ходе выполнения курсовой работы был создан проект на языке C, обрабатывающий BMP изображения. Программа имеет CLI, позволяющий взаимодействовать с ней. Доступны следующие преобразования BMP изображений: замена цвета, фильтр RGB-компонент и разделение изображения на части. Сборка проекта осуществляется с помощью утилиты make.

**ВВЕДЕНИЕ**

Целью работы является изучение структуры BMP изображений, принципа работы с ними на языке программирования C, а также написание программы, реализующей функционал по обработке изображений, управление которой будет осуществляться при помощи командной строки.

**1. ОПИСАНИЕ СТРУКТУРЫ ПРОГРАММЫ**

Описание структур:

* *bmp\_file\_header\_t* – структура, хранящая заголовок BMP файла;
* *bmp\_info\_header\_t* – структура, хранящая информационный заголовок BMP файла;
* *rgb\_t* – структура, содержащая значения красной, зеленой и синей компоненты цвета пикселя;
* *bmp\_t* – структура, содержащая заголовок BMP файла, информационный заголовок BMP файла и массив пикселей BMP файла;
* *command\_line\_options\_t* – структура, содержащая все возможные аргументы командной строки.

Описание функций:

* *void checkBMPFileFormat(BitmapFileHeader bmfhdr)* – проверяет формат BMP файла, при неверном формате завершает программу и выводит соответствующую ошибку;
* *void printFileHeader(BitmapFileHeader header)* – выводит информацию о заголовке BMP файла;
* *void printInfoHeader(BitmapInfoHeader header)* – выводит информацию об информационном заголовке BMP файла;
* *void printBMPFileInfo(BMP\* bmp\_file)* – выводит информацию о BMP файле;
* *BMP\* readBMPFile(char file\_name[])* – чтение BMP файла по указанному пути;
* *bool compareColor(RGB first\_color, RGB second\_color)* – сравнивает 2 цвета в формате RGB;
* *void setColor(BMP\* bmp\_file, int x, int y, RGB new\_color)* – устанавливает указанный цвет по указанным координатам в указанный файл;
* *void colorReplace(BMP\* bmp\_file, CommandLineOptions\* options)* – заменяет все пиксели одного цвета на другой;
* *void rgbFilter(BMP\* bmp\_file, CommandLineOptions\* options)* – фильтр RGB-компонент;
* *void split(BMP\* bmp\_file, CommandLineOptions\* options)* – разделяет изображение на N\*M частей;
* *void writeBMPFile(char file\_name[], BMP\* bmp\_file)* – запись BMP файла по указанному пути;
* *void freeBMPFile(BMP\* bmp\_file)* – освобождение памяти из-под BMP файла;
* *void printHelp()* – выводит справку;
* *RGB convertStringToRGB(char\* color\_string)* – конвертирует строковое представление цвета в формат RGB;
* *CommandLineOptions\* initOptions()* – инициализирует поля объекта структуры *CommandLineOptions*;
* *CommandLineOptions\* parseCommandLine(int argc, char\* argv[])* – считывает аргументы командной строки;
* *void raiseError(const char\* statement, int exit\_code)* – выводит ошибку с указанным сообщением и завершает работу программы с указанным кодом.

Программа была разработана с использованием модульного подхода, что обеспечило ее структурированность. Программа собирается с использованием *Makefile*.

Разработанный код см. в приложении А.

**ТЕСТИРОВАНИЕ**



Рисунок 1 – изображение для тестирования

1. Функция *colorReplace*:

Аргументы запуска: *./cw --color\_replace --old\_color 217.220.216 --new\_color 255.0.0 --input input.bmp --output output.bmp*



Рисунок 2 – результат вызова функции *colorReplace*

1. Функция *rgbFilter*:

Аргументы запуска: *./cw --rgbfilter --component\_name blue --component\_value 67 --input input.bmp --output output.bmp*



Рисунок 3 – результат вызова функции *rgbFilter*

1. Функция *split*:

Аргументы запуска: *./cw --split --number\_x 6 --number\_y 5 --thickness 5 --color 255.0.0 --input input.bmp --output output.bmp*

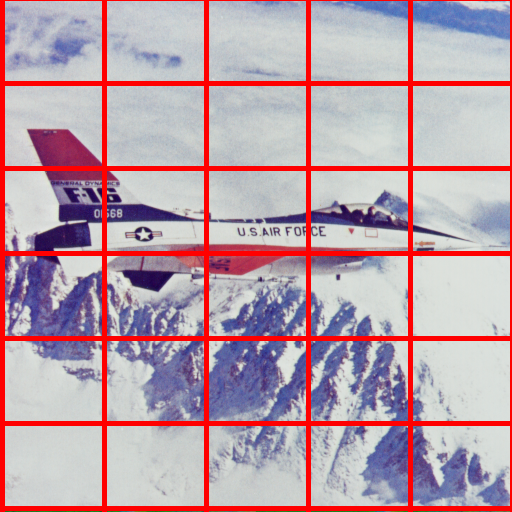


Рисунок 4 – результат вызова функции *split*

**ЗАКЛЮЧЕНИЕ**

В ходе выполнения курсовой работы был создан проект на языке C, обрабатывающий BMP изображения. Сборка проекта реализована при помощи утилиты make. Взаимодействие с программой, а именно запуск и выбор опций осуществляется через CLI.

# Приложение А Исходный код программы

Файл *bmp.h*:

#ifndef BMP\_H

#define BMP\_H

#include <stdio.h>

#include <stdlib.h>

#include "structs.h"

#include "command\_line.h"

void checkBMPFileFormat(BitmapFileHeader bmfhdr);

void printFileHeader(BitmapFileHeader header);

void printInfoHeader(BitmapInfoHeader header);

void printBMPFileInfo(BMP\* bmp\_file);

BMP\* readBMPFile(char file\_name[]);

bool compareColor(RGB first\_color, RGB second\_color);

void setColor(BMP\* bmp\_file, int x, int y, RGB new\_color);

void colorReplace(BMP\* bmp\_file, CommandLineOptions\* options);

void rgbFilter(BMP\* bmp\_file, CommandLineOptions\* options);

void split(BMP\* bmp\_file, CommandLineOptions\* options);

void writeBMPFile(char file\_name[], BMP\* bmp\_file);

void freeBMPFile(BMP\* bmp\_file);

#endif

Файл *command\_line.h*:

#ifndef COMMAND\_LINE\_OPTIONS\_OPERATIONS

#define COMMAND\_LINE\_OPTIONS\_OPERATIONS

#include <stdio.h>

#include <stdlib.h>

#include <getopt.h>

#include <string.h>

#include "structs.h"

void printHelp();

RGB convertStringToRGB(char\* color\_string);

CommandLineOptions\* initOptions();

CommandLineOptions\* parseCommandLine(int argc, char\* argv[]);

#endif

Файл *errors.h*:

#ifndef ERRORS\_H

#define ERRORS\_H

#include <stdio.h>

#include <stdlib.h>

#include "structs.h"

#define ERROR\_INCORRECT\_FILE\_FORMAT 40

#define ERROR\_FILE\_READ\_FAILURE 41

#define ERROR\_FILE\_WRITE\_FAILURE 42

#define ERROR\_INVALID\_ARGUMENT 43

#define ERROR\_MULTIPLE\_FUCTION\_REQUEST 44

#define ERROR\_MISSING\_REQUIRED\_ARGUMENT 45

#define ERROR\_MISSING\_REQUIRED\_OPTION 46

#define ERROR\_UNEXPECTED\_OPTION 47

#define ERROR\_UNKNOWN\_OPTION 48

extern const char\* input\_file\_not\_found;

extern const char\* same\_input\_and\_output\_files;

extern const char\* incorrect\_file\_format;

extern const char\* can\_not\_read\_the\_file;

extern const char\* can\_not\_write\_to\_the\_file;

extern const char\* invalid\_argument;

extern const char\* missing\_required\_argument;

extern const char\* missing\_required\_option;

extern const char\* unexpected\_option;

extern const char\* multiple\_function\_request;

extern const char\* unknown\_option;

extern const char\* incorrect\_rgb\_format;

extern const char\* invalid\_rgb\_component\_value;

extern const char\* invalid\_rgb\_component\_name;

extern const char\* invalid\_parts\_number;

extern const char\* invalid\_thickness;

extern const char\* hello\_message;

void raiseError(const char\* statement, int exit\_code);

#endif

Файл *structs.h*:

#ifndef STRUCTS\_H

#define STRUCTS\_H

#include <stdbool.h>

#pragma pack(push, 1)

typedef struct bmp\_file\_header\_t {

unsigned short signature;

unsigned int filesize;

unsigned short reserved1;

unsigned short reserved2;

unsigned int pixelArrOffset;

} BitmapFileHeader;

typedef struct bmp\_info\_header\_t {

unsigned int headerSize;

unsigned int width;

unsigned int height;

unsigned short planes;

unsigned short bitsPerPixel;

unsigned int compression;

unsigned int imageSize;

unsigned int xPixelsPerMeter;

unsigned int yPixelsPerMeter;

unsigned int colorsInColorTable;

unsigned int importantColorCount;

} BitmapInfoHeader;

#pragma pack(pop)

typedef struct rgb\_t {

unsigned char b;

unsigned char g;

unsigned char r;

} RGB;

typedef struct bmp\_t {

char\* fileName;

BitmapFileHeader bmfhdr;

BitmapInfoHeader bmihdr;

RGB\*\* pixel\_array;

} BMP;

typedef struct command\_line\_options\_t {

char\* input;

char\* output;

bool info;

bool help;

bool colorReplace;

char\* oldColor;

char\* newColor;

bool rgbFilter;

char\* componentName;

unsigned char componentValue;

bool split;

int numberX;

int numberY;

int thickness;

char\* color;

} CommandLineOptions;

#endif

Файл *bmp.c*:

#include "bmp.h"

#include "errors.h"

void checkBMPFileFormat(BitmapFileHeader bmfhdr)

{

if (bmfhdr.signature != 0x4D42) {

raiseError(incorrect\_file\_format, ERROR\_INCORRECT\_FILE\_FORMAT);

}

}

void printFileHeader(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 printInfoHeader(BitmapInfoHeader header)

{

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

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

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

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

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

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

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

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

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

printf(" colorsInColorTable:\t%x (%u)\n", header.colorsInColorTable, header.colorsInColorTable);

printf(" importantColorCount:\t%x (%u)\n", header.importantColorCount, header.importantColorCount);

}

void printBMPFileInfo(BMP\* bmp\_file)

{

printf("File %s:\n", bmp\_file->fileName);

printFileHeader(bmp\_file->bmfhdr);

printInfoHeader(bmp\_file->bmihdr);

}

BMP\* readBMPFile(char file\_name[])

{

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

if (!file)

raiseError(can\_not\_read\_the\_file, ERROR\_FILE\_READ\_FAILURE);

BMP\* bmp\_file = (BMP\*)malloc(sizeof(BMP));

bmp\_file->fileName = file\_name;

fread(&bmp\_file->bmfhdr, 1, sizeof(BitmapFileHeader), file);

checkBMPFileFormat(bmp\_file->bmfhdr);

fread(&bmp\_file->bmihdr, 1, sizeof(BitmapInfoHeader), file);

unsigned int H = bmp\_file->bmihdr.height;

unsigned int W = bmp\_file->bmihdr.width;

bmp\_file->pixel\_array = (RGB\*\*)malloc(sizeof(RGB\*) \* H);

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

bmp\_file->pixel\_array[i] = (RGB\*)malloc(sizeof(RGB) \* W + (4 - (sizeof(RGB) \* W) % 4) % 4);

fread(bmp\_file->pixel\_array[i], 1, sizeof(RGB) \* W + (4 - (sizeof(RGB) \* W) % 4) % 4, file);

}

fclose(file);

return bmp\_file;

}

bool compareColor(RGB first\_color, RGB second\_color)

{

if (first\_color.r == second\_color.r && first\_color.g == second\_color.g && first\_color.b == second\_color.b)

return true;

return false;

}

void setColor(BMP\* bmp\_file, int x, int y, RGB new\_color)

{

if (x < 0 || x >= bmp\_file->bmihdr.width || y < 0 || y >= bmp\_file->bmihdr.height)

return;

bmp\_file->pixel\_array[y][x].r = new\_color.r;

bmp\_file->pixel\_array[y][x].g = new\_color.g;

bmp\_file->pixel\_array[y][x].b = new\_color.b;

}

void colorReplace(BMP\* bmp\_file, CommandLineOptions\* options)

{

RGB old\_color = convertStringToRGB(options->oldColor);

RGB new\_color = convertStringToRGB(options->newColor);

unsigned int W = bmp\_file->bmihdr.width;

unsigned int H = bmp\_file->bmihdr.height;

for (int y = 0; y < H; y++) {

for (int x = 0; x < W; x++) {

if (compareColor(bmp\_file->pixel\_array[y][x], old\_color)) {

setColor(bmp\_file, x, y, new\_color);

}

}

}

}

void rgbFilter(BMP\* bmp\_file, CommandLineOptions\* options)

{

if (strcmp(options->componentName, "red") && strcmp(options->componentName, "green") && strcmp(options->componentName, "blue"))

raiseError(invalid\_rgb\_component\_name, ERROR\_INVALID\_ARGUMENT);

if (options->componentValue < 0 || options->componentValue > 255)

raiseError(invalid\_rgb\_component\_value, ERROR\_INVALID\_ARGUMENT);

unsigned int H = bmp\_file->bmihdr.height;

unsigned int W = bmp\_file->bmihdr.width;

for (int y = 0; y < H; y++) {

for (int x = 0; x < W; x++) {

if (strcmp(options->componentName, "red") == 0) {

bmp\_file->pixel\_array[y][x].r = options->componentValue;

}

else if (strcmp(options->componentName, "green") == 0) {

bmp\_file->pixel\_array[y][x].g = options->componentValue;

}

else if (strcmp(options->componentName, "blue") == 0) {

bmp\_file->pixel\_array[y][x].b = options->componentValue;

}

}

}

}

void split(BMP\* bmp\_file, CommandLineOptions\* options)

{

unsigned int W = bmp\_file->bmihdr.width;

unsigned int H = bmp\_file->bmihdr.height;

if (options->numberX <= 1 || options->numberX > W || options->numberY <= 1 || options->numberY > H)

raiseError(invalid\_parts\_number, ERROR\_INVALID\_ARGUMENT);

RGB color = convertStringToRGB(options->color);

if (options->thickness <= 0)

raiseError(invalid\_thickness, ERROR\_INVALID\_ARGUMENT);

unsigned int step\_x = W / options->numberY;

unsigned int step\_y = H / options->numberX;

for (int x = 0; x < W; x += step\_x) {

for (int y = H - 1; y >= 0; y--) {

for (int i = 0; i < options->thickness; i++)

setColor(bmp\_file, x + i, y, color);

}

}

for (int y = H - 1; y >= 0; y -= step\_y) {

for (int x = 0; x < W; x++) {

for (int j = 0; j < options->thickness; j++)

setColor(bmp\_file, x, y + j, color);

}

}

}

void writeBMPFile(char file\_name[], BMP\* bmp\_file)

{

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

if (!file)

raiseError(can\_not\_write\_to\_the\_file, ERROR\_FILE\_WRITE\_FAILURE);

fwrite(&bmp\_file->bmfhdr, 1, sizeof(BitmapFileHeader), file);

fwrite(&bmp\_file->bmihdr, 1, sizeof(BitmapInfoHeader), file);

unsigned int H = bmp\_file->bmihdr.height;

unsigned int W = bmp\_file->bmihdr.width;

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

fwrite(bmp\_file->pixel\_array[i], 1, sizeof(RGB) \* W + (4 - (sizeof(RGB) \* W) % 4) % 4, file);

fclose(file);

}

void freeBMPFile(BMP\* bmp\_file)

{

if(bmp\_file) {

if(bmp\_file->pixel\_array) {

for (int i = 0; i < bmp\_file->bmihdr.height; i++)

free(bmp\_file->pixel\_array[i]);

free(bmp\_file->pixel\_array);

}

free(bmp\_file);

}

}

Файл *command\_line.c*:

#include "command\_line.h"

#include "errors.h"

void printHelp()

{

printf("%s\n"

"Flags:\n"

" -h --help: Вывод справочной информации;\n"

" -i --input: Имя входного файла;\n"

" -o --output: Имя выходного файла;\n"

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

" --color\_replace: Заменяет все пиксели одного заданного цвета на другой цвет;\n"

" --old\_color: Цвет, который требуется заменить;\n"

" --new\_color: Цвет, на который требуется заменить;\n\n"

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

" --component\_name: Какую компоненту требуется изменить;\n"

" --component\_value: На какое значение ее требуется изменить;\n\n"

" --split: Разделяет изображение на N\*M частей;\n"

" --number\_x: Количество частей по “оси” Y;\n"

" --number\_y: Количество частей по “оси” X;\n"

" --thickness: Толщина линии;\n"

" --color: Цвет линии.\n", hello\_message);

}

RGB convertStringToRGB(char\* color\_string)

{

int r, g, b;

sscanf(color\_string, "%d.%d.%d", &r, &g, &b);

if (r < 0 || r > 255 || g < 0 || g > 255 || b < 0 || b > 255)

raiseError(incorrect\_rgb\_format, ERROR\_INVALID\_ARGUMENT);

RGB color;

color.r = r;

color.g = g;

color.b = b;

return color;

}

CommandLineOptions\* initOptions()

{

CommandLineOptions\* options = (CommandLineOptions\*)calloc(1, sizeof(CommandLineOptions));

options->input = NULL;

options->output = "out.bmp";

options->info = false;

options->colorReplace = false;

options->rgbFilter = false;

options->split = false;

return options;

}

CommandLineOptions\* parseCommandLine(int argc, char\* argv[])

{

opterr = 0;

if (argc == 1) {

printHelp();

exit(EXIT\_SUCCESS);

}

const char\* short\_options = "i:o:h";

const struct option long\_options[] = {

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

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

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

{"info", no\_argument, NULL, 256},

{"color\_replace", no\_argument, NULL, 257},

{"old\_color", required\_argument, NULL, 258},

{"new\_color", required\_argument, NULL, 259},

{"rgbfilter", no\_argument, NULL, 260},

{"component\_name", required\_argument, NULL, 261},

{"component\_value", required\_argument, NULL, 262},

{"split", no\_argument, NULL, 263},

{"number\_x", required\_argument, NULL, 264},

{"number\_y", required\_argument, NULL, 265},

{"thickness", required\_argument, NULL, 266},

{"color", required\_argument, NULL, 267},

{NULL, 0, NULL, 0}

};

CommandLineOptions\* options = initOptions();

int res = 0;

while ((res = getopt\_long(argc, argv, short\_options, long\_options, NULL)) != -1) {

switch(res)

{

case 'i':

if (!optarg || optarg[0] == '-'){

raiseError(missing\_required\_argument, ERROR\_MISSING\_REQUIRED\_ARGUMENT);

}

options->input = optarg;

break;

case 'o':

if (!optarg || optarg[0] == '-'){

raiseError(missing\_required\_argument, ERROR\_MISSING\_REQUIRED\_ARGUMENT);

}

options->output = optarg;

break;

case 'h':

printHelp();

exit(EXIT\_SUCCESS);

break;

case 256:

if (options->colorReplace || options->rgbFilter || options->split)

raiseError(unexpected\_option, ERROR\_UNEXPECTED\_OPTION);

options->info = true;

break;

case 257:

if (options->info)

raiseError(unexpected\_option, ERROR\_UNEXPECTED\_OPTION);

if (!options->rgbFilter && !options->split)

options->colorReplace = true;

else

raiseError(multiple\_function\_request, ERROR\_MULTIPLE\_FUCTION\_REQUEST);

break;

case 258:

if (!optarg || optarg[0] == '-') {

raiseError(missing\_required\_argument, ERROR\_MISSING\_REQUIRED\_ARGUMENT);

}

options->oldColor = optarg;

break;

case 259:

if (!optarg || optarg[0] == '-') {

raiseError(missing\_required\_argument, ERROR\_MISSING\_REQUIRED\_ARGUMENT);

}

options->newColor = optarg;

break;

case 260:

if (options->info)

raiseError(unexpected\_option, ERROR\_UNEXPECTED\_OPTION);

if (!options->colorReplace && !options->split)

options->rgbFilter = true;

else

raiseError(multiple\_function\_request, ERROR\_MULTIPLE\_FUCTION\_REQUEST);

break;

case 261:

if (!optarg || optarg[0] == '-')

raiseError(missing\_required\_argument, ERROR\_MISSING\_REQUIRED\_ARGUMENT);

options->componentName = optarg;

break;

case 262:

if (!optarg || optarg[0] == '-')

raiseError(missing\_required\_argument, ERROR\_MISSING\_REQUIRED\_ARGUMENT);

sscanf(optarg, "%hhu", &options->componentValue);

break;

case 263:

if (options->info)

raiseError(unexpected\_option, ERROR\_UNEXPECTED\_OPTION);

if (!options->colorReplace && !options->rgbFilter)

options->split = true;

else

raiseError(multiple\_function\_request, ERROR\_MULTIPLE\_FUCTION\_REQUEST);

break;

case 264:

if (!optarg || optarg[0] == '-')

raiseError(missing\_required\_argument, ERROR\_MISSING\_REQUIRED\_ARGUMENT);

options->numberX = strtol(optarg, NULL, 10);

break;

case 265:

if (!optarg || optarg[0] == '-')

raiseError(missing\_required\_argument, ERROR\_MISSING\_REQUIRED\_ARGUMENT);

options->numberY = strtol(optarg, NULL, 10);

break;

case 266:

if (!optarg || optarg[0] == '-')

raiseError(missing\_required\_argument, ERROR\_MISSING\_REQUIRED\_ARGUMENT);

options->thickness = strtol(optarg, NULL, 10);

break;

case 267:

if (!optarg || optarg[0] == '-')

raiseError(missing\_required\_argument, ERROR\_MISSING\_REQUIRED\_ARGUMENT);

options->color = optarg;

break;

default:

raiseError(unknown\_option, ERROR\_UNKNOWN\_OPTION);

break;

}

}

if (!options->input) {

if (optind == argc - 1)

options->input = argv[optind];

else

raiseError(input\_file\_not\_found, ERROR\_INVALID\_ARGUMENT);

}

if (!strcmp(options->input, options->output))

raiseError(same\_input\_and\_output\_files, ERROR\_INVALID\_ARGUMENT);

int task\_number = 0;

if (options->colorReplace || options->oldColor || options->newColor) {

if (task\_number != 0)

raiseError(unexpected\_option, ERROR\_UNEXPECTED\_OPTION);

task\_number = 1;

}

if (options->rgbFilter || options->componentName || options->componentValue) {

if (task\_number != 0)

raiseError(unexpected\_option, ERROR\_UNEXPECTED\_OPTION);

task\_number = 2;

}

if (options->split || options->numberX || options->numberY || options->thickness || options->color) {

if (task\_number != 0)

raiseError(unexpected\_option, ERROR\_UNEXPECTED\_OPTION);

task\_number = 3;

}

int options\_cnt;

if (task\_number == 1) {

options\_cnt = 0;

if (options->colorReplace) options\_cnt++;

if (options->oldColor) options\_cnt++;

if (options->newColor) options\_cnt++;

if (options\_cnt != 3)

raiseError(missing\_required\_option, ERROR\_MISSING\_REQUIRED\_OPTION);

}

if (task\_number == 2) {

options\_cnt = 0;

if (options->rgbFilter) options\_cnt++;

if (options->componentName) options\_cnt++;

if (options->componentValue) options\_cnt++;

if (options\_cnt != 3)

raiseError(missing\_required\_option, ERROR\_MISSING\_REQUIRED\_OPTION);

}

if (task\_number == 3) {

options\_cnt = 0;

if (options->split) options\_cnt++;

if (options->numberX) options\_cnt++;

if (options->numberY) options\_cnt++;

if (options->thickness) options\_cnt++;

if (options->color) options\_cnt++;

if (options\_cnt != 5)

raiseError(missing\_required\_option, ERROR\_MISSING\_REQUIRED\_OPTION);

}

return options;

}

Файл *errors.c*:

#include "errors.h"

const char\* input\_file\_not\_found = "No input file!";

const char\* same\_input\_and\_output\_files = "Input and output files are the same!";

const char\* incorrect\_file\_format = "Incorrect file format!";

const char\* can\_not\_read\_the\_file = "Can't read the file!";

const char\* can\_not\_write\_to\_the\_file = "Can't write to the file!";

const char\* invalid\_argument = "Invalid argument!";

const char\* missing\_required\_argument = "Missing required argument!";

const char\* missing\_required\_option = "Missing required option!";

const char\* unexpected\_option = "Unexpected option!";

const char\* multiple\_function\_request = "Only one function can be called!";

const char\* unknown\_option = "Unknown option!";

const char\* incorrect\_rgb\_format = "Incorrect RGB format";

const char\* invalid\_rgb\_component\_value = "Wrong RGB component value!";

const char\* invalid\_rgb\_component\_name = "Companent name must be 'red', 'green' or 'blue'!";

const char\* invalid\_parts\_number = "Wrong parts number!";

const char\* invalid\_thickness = "Wrong thickness!";

const char\* hello\_message = "Course work for option 4.2, created by Otmakhov Danil.";

void raiseError(const char\* statement, int exit\_code)

{

fprintf(stderr, "Error: %s\n", statement);

exit(exit\_code);

}

Файл *main.c*:

#include "bmp.h"

#include "errors.h"

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

{

puts(hello\_message);

CommandLineOptions\* options = parseCommandLine(argc, argv);

BMP\* bmp\_file = readBMPFile(options->input);

if (options->info) {

printBMPFileInfo(bmp\_file);

exit(EXIT\_SUCCESS);

}

if (options->colorReplace)

colorReplace(bmp\_file, options);

if (options->rgbFilter)

rgbFilter(bmp\_file, options);

if (options->split)

split(bmp\_file, options);

writeBMPFile(options->output, bmp\_file);

free(options);

freeBMPFile(bmp\_file);

return 0;

}