

**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 следующие 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`).
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. 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*

2. Функция *rgbFilter*:



Аргументы запуска: `./cw --rgbfilter --component_name blue --component_value 67 --input input.bmp --output output.bmp`



Рисунок 3 – результат вызова функции *rgbFilter*

### 3. 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;
}
```