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

# КУРСОВАЯ РАБОТА

**по дисциплине «Программирование»**

**Тема: Обработка изображений на языке Си**

Студент гр. 3342 _____ Хайруллов Д.Л.

Преподаватель _____ Глазунов С.А.

Санкт-Петербург

2024

**ЗАДАНИЕ**
**НА КУРСОВУЮ РАБОТУ**

Студент Хайруллов Д.Л.

Группа 3342

Тема работы: Обработка изображений на языке Си

Исходные данные:

Вариант 4.10

Общие сведения:

24 бита на цвет

без сжатия

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`)

(2) Сделать рамку в виде узора. Флаг для выполнения данной операции: `--ornament`. Рамка определяется:

Узором. Флаг `--pattern`. Обязательные значения: rectangle и circle, semicircles. Также можно добавить свои узоры (красивый узор можно получить используя фракталы). Подробнее здесь:

https://se.moevm.info/doku.php/courses:programming:cw_spring_ornament

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)

Шириной. Флаг `--thickness`. На вход принимает число больше 0

Количеством. Флаг `--count`. На вход принимает число больше 0

При необходимости можно добавить дополнительные флаги для необозначенных узоров

(3) Поиск всех залитых прямоугольников заданного цвета. Флаг для выполнения данной операции: `--filled_rects`. Требуется найти все прямоугольники заданного цвета и обвести их линией. Функционал определяется:

Цветом искомых прямоугольников. Флаг `--color` (цвет задаётся строкой `rrr.ggg.bbb`, где rrr/ggg/bbb – числа, задающие цветовую компоненту. пример `--color 255.0.0` задаёт красный цвет)

Цветом линии для обводки. Флаг `--border_color` (работает аналогично флагу `--color`)

Толщиной линии для обводки. Флаг `--thickness`. На вход принимает число больше 0

Содержание пояснительной записки:

Разделы пояснительный записки: «Содержание», «Введение», «Ход работы», «Структуры», «Заключение», «Список использованных источников», «Приложение А. Результаты тестирования», «Приложение Б. Исходный код программы».

Предполагаемый объем пояснительной записки:
Не менее 37 страниц.

Дата выдачи задания: 18.03.2024

Дата сдачи реферата: 29.05.2024

Дата защиты реферата: 29.05.2024

Студент _____ Хайруллов Д.Л.

Преподаватель _____ Глазунов С.А.

## АННОТАЦИЯ

Целью данной курсовой работы является разработка программы для обработки изображений формата bmp на языке программирования C. Программа считывает изображение формата bmp, получает на вход различные опции, в зависимости от которых программа преобразует изображение. В программе используются стандартные библиотеки Си, с помощью которых были реализованы необходимые функции.

## SUMMARY

The aim of this coursework is to develop a program for processing BMP image files using the C programming language. The program reads BMP image files, takes various options as input, and based on these options, processes the image accordingly. Standard C libraries are used in the program to implement the necessary functions.

**СОДЕРЖАНИЕ**

# ВВЕДЕНИЕ

Целью данной курсовой работы является разработка программы для обработки изображения формата bmp на языке программирования C.

Основные задачи:

1) Считывание изображения
2) Считывание опций для обработки изображений
3) Обработка изображения
4) Сохранения изображения

# 1. ФУНКЦИИ ВВОДА И ВЫВОДА

## 1.1. Функция считывания изображения

Данная функция read_bmp открывает файл изображения BMP, считывает заголовок файла и информацию о изображении. Затем происходит выделение памяти под данные изображения на основе его высоты и ширины. Последующее чтение данных пикселей изображения происходит в цикле по строкам. Наконец, файл закрывается, и функция возвращает структуру BMP, содержащую данные изображения.

## 1.2. Функция записи изображения

Функция write_bmp открывает файл для записи в формате BMP, записывает заголовок файла и информацию о изображении из структуры BMP_file. Затем данные пикселей изображения записываются в файл в цикле по строкам. По завершении записи файл закрывается.

## 1.3. Функции вывода информации об изображении

Функция printFileHeader выводит информацию о заголовке файла BMP, включая сигнатуру, размер файла, резервные поля и смещение массива пикселей.

Функция printInfoHeader выводит информацию о заголовке информации об изображении BMP, включая размер заголовка, ширину и высоту изображения, количество бит на пиксель, метод сжатия, размер изображения, разрешение и дополнительные параметры. Вызов функции и результат ее работы представлены в приложении А на рисунке 3.

## 1.4. Функция вывода справки

Функция printHelp выводит краткую справку о программе для выполнения курсовой работы по опции 4.10, созданной Khairullov Dinar. Выводит список доступных опций с их описанием, включая информацию о том, как использовать каждую опцию: помощь, информация о файле, установка имени входного файла, установка имени выходного файла, замена цвета пикселей, установка старого и нового цветов, рисование орнаментов, установка

шаблона орнамента, установка цвета заполненного прямоугольника, установка толщины линий, установка количества элементов орнамента, поиск заполненных прямоугольников заданного цвета и добавление границ и установка цвета границ. Вызов функции и результат ее работы представлены в приложении А на рисунке 2.

# 2. ФУНКЦИИ ОБРАБОТКИ ИЗОБРАЖЕНИЯ

## 2.1. Функция первого задания

Функция color_replace заменяет все пиксели изображения bmp_file, имеющие определенный старый цвет old_color, на новый цвет new_color. Для этого функция проходит по каждому пикселю изображения и, если его цвет соответствует старому цвету, вызывает функцию color_pixel, которая заменяет цвет пикселя на новый цвет.. Вызов функции и результат ее работы представлены в приложении А на рисунке 4 и рисунке 5.

## 2.2. Функции второго задания

Функция rectangle_ornament рисует прямоугольные орнаменты определенной толщины и количества на изображении bmp_file, используя указанный цвет color. Орнаменты направлены по диагонали. Вызов функции и результат ее работы представлены в приложении А на рисунке 6 и рисунке 7.

Функция circle_ornament рисует круглый орнамент с центром в центре изображения bmp_file и радиусом, достигающим до ближайшего края изображения. Орнамент закрашивает пиксели за пределами круга указанным цветом. Вызов функции и результат ее работы представлены в приложении А на рисунке 8 и рисунке 9.

Функция semicircle_ornament рисует полукруглые орнаменты вертикально и горизонтально на изображении bmp_file. Орнаменты имеют определенную толщину, радиусы от центра внутренних кругов и количество. Орнаменты закрашиваются указанным цветом. РВызов функции и результат ее работы представлены в приложении А на рисунке 10 и рисунке 11.

## 2.3. Функция третьего задания

Функция filled_rects предназначена для поиска и закрашивания прямоугольных областей одного цвета на изображении. Вначале функция определяет размеры изображения (H - высота, W - ширина) и начальные значения координат прямоугольника (x_left, y_down, x_right, y_up). Также выделяется память под массивы start_colored_pixels и end_colored_pixels,

используемые для хранения координат начала и конца прямоугольных областей.

Затем функция перебирает все пиксели изображения и ищет пиксели заданного цвета (color). При нахождении такого пикселя функция определяет границы прямоугольной области, заполненной цветом. Для этого определяются начальные координаты и ширина прямоугольника. Затем осуществляется проверка пикселей в пределах найденного прямоугольника.

В процессе работы функция использует вспомогательную функцию colored_pixel_check, обрабатывает флаги и осуществляет динамическое выделение памяти с помощью realloc для массивов хранения информации о прямоугольных областях. Для найденных прямоугольников рисуется рамка внутри фигуры. Вызов функции и результат ее работы представлены в приложении А на рисунке 12 и рисунке 13.

# 3. ОСТАЛЬНЫЕ ФУНКЦИИ

## 3.1. Функция закрашивания области

Функция color_area предназначена для закрашивания прямоугольной области на изображении определенным цветом. В начале функции определяются размеры изображения (H - высота, W - ширина) и проверяется корректность входных координат прямоугольника. Если координаты находятся за пределами изображения, функция завершает свою работу.

Затем функция проверяет правильность порядка координат прямоугольника и, если необходимо, меняет их местами. После этого функция проходит по всем пикселям в границах указанной области и закрашивает их определенным цветом с помощью вспомогательной функции color_pixel.

## 3.2. Функция замены цвета

Функция color_replace предназначена для замены определенного цвета на другой на всем изображении. В начале функции определяются размеры изображения (H - высота, W - ширина). Затем функция проходит по всем пикселям изображения, и если цвет текущего пикселя совпадает с цветом old_color, то этот пиксель заменяется на цвет new_color с помощью вспомогательной функции color_pixel.

## 3.3. Функция получения цвета

Функция get_color преобразует строковое значение цвета в структуру Rgb, представляющую цвет в формате RGB.

## 3.4. Функция обмена значениями

Функция swap выполняет обмен значениями двух целочисленных переменных, переданных по указателям.

## 3.5. Функция main

Функция main представляет точку входа в программу и обрабатывает переданные аргументы командной строки. Она использует библиотеку getopt_long для разбора опций командной строки и выполнения соответствующих действий в зависимости от переданных параметров.

В этой функции определены различные опции для работы с изображением, такие как замена цвета, добавление узора, вывод информации о изображении и другие операции. Она также обрабатывает ошибки ввода и выводит сообщения об ошибке при необходимости.

После обработки опций командной строки, функция main читает изображение из файла, получает его высоту и ширину, а затем в зависимости от выбранной опции выполняет соответствующие операции над изображением.

# ЗАКЛЮЧЕНИЕ

При выполнении курсовой работы были изучены основные методы работы с изображениями формата bmp в языке программирования C. Для выполнения заданий работы была написана программа, использующая стандартные библиотеки и функции данного языка программирования. Тестирование программы показало, что все функции программы работают корректно и выполняют необходимые задачи в соответствии с условиями.

Была создана программа, выполняющая все необходимые задачи корректно. Тем самым можно сказать, что все поставленные в работе цели были достигнуты.

В приложении А представлены результаты тестирования программы.

В приложении Б представлен код программы.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Курс “Программирование на Си. Практические задания. Второй семестр”. URL https://e.moevm.info/course/view.php?id=18

2. Язык программирования С / Керниган Брайан, Ритчи Деннис. СПб.: "Финансы и статистика", 2003.

# ПРИЛОЖЕНИЕ А
# РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ

Изначальное изображение

Рисунок 1 – начальное изображение

Тест 1 – вывод справки

```
user@user-VirtualBox:~/Khairullov_Dinar_cw/src$ ./cw --help
Course work for option 4.10, created by Khairullov Dinar

--help, -h: displaying help information
--info, -i: displaying  file information
--input, -I: setting the input file name
--output, -o: setting the output file name
--color_replace, -C: changing pixels' color
--old_color, -O: setting old_color value
--new_color, -n: setting new color value
--ornament, -r: drawing ornament
--pattern, -p: setting pattern of ornament
--color, -c: setting color of filled_rectangle
--thickness, -t: setting thickness of lines
--count, -u: setting number of ornament elements
--filled_rects, -f: finding rectangles filled with given color and making borders
--border_color, -b : setting border color value
```

Рисунок 2 – вывод справки в консоль

Тест 2 – вывод информации об изображении

```
user@user-VirtualBox:~/Khairullov_Dinar_cw/src$ ./cw --input ./TEST.bmp --info
signature:      4d42 (19778)
filesize:       c7b8a (818058)
reserved1:      0 (0)
reserved2:      0 (0)
pixelArrOffset: 8a (138)
headerSize:     7c (124)
width:          280 (640)
height:         1aa (426)
planes:         1 (1)
bitsPerPixel:   18 (24)
compression:    0 (0)
imageSize:      c7b00 (817920)
xPixelsPerMeter:        0 (0)
yPixelsPerMeter:        0 (0)
colorsInColorTable:     0 (0)
importantColorCount:    0 (0)
```

Рисунок 3 – вывод информации об изображении в консоль

Тест 3 – функция замена цвета

```
user@user-VirtualBox:~/Khairullov_Dinar_cw/src$ ./cw --input ./TEST.bmp --color_replace --old_color 0.0.0 --new_color 25
5.0.0
user@user-VirtualBox:~/Khairullov_Dinar_cw/src$
```

Рисунок 4 – вызов функции замены цвета в консоли

Рисунок 5 – результат работы функции замены цвета

Тест 4 – рисование прямоугольной рамки

```
user@user-VirtualBox:~/Khairullov_Dinar_cw/src$ ./cw --input ./TEST.bmp --ornament --pattern rectangle --count 4 --thick
ness 4 --color 234.12.63
user@user-VirtualBox:~/Khairullov_Dinar_cw/src$
```

Рисунок 6 – вызов функции рисования прямоугольной рамки

Рисунок 7 – результат работы функции рисования прямоугольной рамки

Тест 5 – рисование круглой рамки

```
user@user-VirtualBox:~/Khairullov_Dinar_cw/src$ ./cw --input ./TEST.bmp --ornament --pattern circle --color 234.12.63
user@user-VirtualBox:~/Khairullov_Dinar_cw/src$
```

Рисунок 8 – вызов функции рисования круглой рамки

Рисунок 9 – результат работы функции рисования круглой рамки

Тест 6 – рисование рамки из полукругов

```
user@user-VirtualBox:~/Khairullov_Dinar_cw/src$ ./cw --input ./TEST.bmp --ornament --pattern semicircles --count 4 --thi
ckness 4 --color 234.12.63
user@user-VirtualBox:~/Khairullov_Dinar_cw/src$
```

Рисунок 10 – вызов функции рисования рамки из полукругов

Рисунок 11 – результат работы функции рисования рамки из полукругов

Тест 7 – нахождения прямоугольников одного цвета

```
user@user-VirtualBox:~/Khairullov_Dinar_cw/src$ ./cw --input ./TEST.bmp --filled_rects --thickness 4 --color 0.0.0 --bor
der_color 0.255.255
user@user-VirtualBox:~/Khairullov_Dinar_cw/src$
```

Рисунок 12 – вызов нахождения прямоугольников одного цвета

Рисунок 13 – результат работы функции нахождения прямоугольников одного цвета

# ПРИЛОЖЕНИЕ Б
# ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <getopt.h>
#include <math.h>
#include <stdint.h>
#include <ctype.h>

#define ERROR_OPENING_FILE_EXIT 40
#define ERROR_REALOCATING_MEMORY_EXIT 41
#define WRONG_BMP_DATA_EXIT 42
#define WRONG_OPTION_EXIT 43
#define ERROR_COLOR_EXIT 44
#define ERROR_FEW_ARGS_EXIT 45
#define ERROR_INVALID_COUNT_VALUE_EXIT 46
#define ERROR_INVALID_THICKNESS_VALUE_EXIT 47
#define ERROR_INVALID_PATTERN_VALUE_EXIT 48
#define ERROR_INVALID_VALUE_EXIT 49


#define ERROR_OPENING_FILE_PRINT "Error: file opening error\n"
#define  ERROR_REALOCATING_MEMORY_PRINT    "Error:  realocating  memory
error\n"
#define WRONG_BMP_DATA_PRINT "Error: wrong bmp data\n"
#define WRONG_OPTION_PRINT "Error: wrong option\n"
#define ERROR_COLOR_PRINT "Error: invalid color data\n"
#define ERROR_FEW_ARGS_PRINT "Error: too few arguments were given\n"
#define ERROR_INVALID_COUNT_VALUE_PRINT "Error: invalid count value\n"
#define  ERROR_INVALID_THICKNESS_VALUE_PRINT  "Error:  invalid  thickness
value\n"
#define   ERROR_INVALID_PATTERN_VALUE_PRINT   "Error:   invalid   pattern
value\n"
#define ERROR_INVALID_VALUE_PRINT "Error: invalid value\n"

#pragma pack(push, 1)


typedef struct
{
    unsigned short signature;    // определение типа файла
    unsigned int filesize;       // размер файла
    unsigned short reserved1;    // должен быть 0
    unsigned short reserved2;    // должен быть 0
    unsigned int pixelArrOffset; // начальный адрес байта, в котором
находятся данные изображения (массив пикселей)
} BitmapFileHeader;

typedef struct
{
    unsigned int headerSize;          // размер этого заголовка в байтах
    unsigned int width;               // ширина изображения в пикселях
    unsigned int height;              // высота изображения в пикселях
```

```
    unsigned short planes;                // кол-во цветовых плоскостей
(должно быть 1)
    unsigned short bitsPerPixel;      // глубина цвета изображения
    unsigned int compression;             // тип сжатия; если сжатия не
используется, то здесь должен быть 0
    unsigned int imageSize;           // размер изображения
    unsigned int xPixelsPerMeter;         // горизонтальное разрешение
(пиксель на метр)
    unsigned int yPixelsPerMeter;     // вертикальное разрешение (пиксель
на метр)
    unsigned int colorsInColorTable;  // кол-во цветов в цветовой палитре
    unsigned int importantColorCount; // кол-во важных цветов (или 0,
если каждый цвет важен)
} BitmapInfoHeader;

typedef struct
{
    unsigned char b;
    unsigned char g;
    unsigned char r;
} Rgb;

#pragma pack(pop)

typedef struct
{
    BitmapInfoHeader bmih;
    BitmapFileHeader bmfh;
    Rgb **img;
} BMP;

void printFileHeader(BitmapFileHeader header);
void printInfoHeader(BitmapInfoHeader header);
void printHelp();
int getOffset(size_t width);
void swap(int *a, int *b);
void color_pixel(BMP* bmp_file, int x, int y, Rgb *color);
void check_bmp_data(BMP* bmp_file);
Rgb get_color(char* char_color);
BMP read_bmp(char* file_name);
void write_bmp(char file_name[], BMP BMP_file);
void color_area(BMP* bmp_file, int x_left, int y_down, int x_right, int
y_up, Rgb* color);
void draw_rectangle(BMP* bmp_file, Rgb* color, int thickness, int x_left,
int y_down, int x_right, int y_up);
void check_color_replace(char* old_color, char* new_color);
void color_replace(BMP* bmp_file, Rgb old_color, Rgb new_color);
void rectangle_ornament(BMP* bmp_file, Rgb* color, int thickness, int
count);
void circle_ornament(BMP* bmp_file, Rgb* color);

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 printHelp()
{
    printf("Course   work   for   option   4.10,   created   by   Khairullov
Dinar\n\n");

    printf("--help, -h: displaying help information\n");
    printf("--info, -i: displaying  file information\n");
    printf("--input, -I: setting the input file name\n");
    printf("--output, -o: setting the output file name\n");
    printf("--color_replace, -C: changing pixels' color\n");
    printf("--old_color, -O: setting old_color value\n");
    printf("--new_color, -n: setting new color value\n");
    printf("--ornament, -r: drawing ornament\n");
    printf("--pattern, -p: setting pattern of ornament\n");
    printf("--color, -c: setting color of filled_rectangle\n");
    printf("--thickness, -t: setting thickness of lines\n");
    printf("--count, -u: setting number of ornament elements\n");
    printf("--filled_rects,  -f:  finding  rectangles  filled  with  given
color and making borders\n");
    printf("--border_color, -b : setting border color value\n");
}

void image_free(BMP* bmp_file)
{
    unsigned int H = bmp_file->bmih.height;

    for(int i = 0; i < H; i++){
          free(bmp_file->img[i]);
     }

     free(bmp_file->img);
```

```
}

int getOffset(size_t width)
{
    unsigned int offset = (width * sizeof(Rgb)) % 4;
    offset = (offset ? 4-offset : 0);
    return offset;
}

void swap(int *a, int *b)
{
    int t = *a;
    *a = *b;
    *b = t;
}

void color_pixel(BMP* bmp_file, int x, int y, Rgb *color)
{
    bmp_file->img[y][x].r = color->r;
    bmp_file->img[y][x].g = color->g;
    bmp_file->img[y][x].b = color->b;
}

void check_bmp_data(BMP* bmp_file)
{
    if(bmp_file->bmih.bitsPerPixel != 24 || bmp_file->bmih.compression !=
0 || bmp_file->bmfh.signature != 0x4d42 || bmp_file->bmih.headerSize !=
40)
    {
        printf(WRONG_BMP_DATA_PRINT);
        exit(WRONG_BMP_DATA_EXIT);
    }
}

Rgb get_color(char* char_color)
{
    if(char_color == NULL)
    {
        printf(ERROR_COLOR_PRINT);
        exit(ERROR_COLOR_EXIT);
    }

    Rgb color;
    int rgb_values[3];
    int counter = 0;
    int check_counter = 0;
    char sep[3] =".";

    if(char_color[strlen(char_color) - 1] == '.' || char_color[0] == '.')
    {
        printf(ERROR_COLOR_PRINT);
        exit(ERROR_COLOR_EXIT);
    }
    for(int index = 0; index < strlen(char_color); index++)
    {
        if(char_color[index] == '.')
        {
```

```
            if(char_color[index + 1] == '.')
            {
                printf(ERROR_COLOR_PRINT);
                exit(ERROR_COLOR_EXIT);
            }
            check_counter++;
        }
    }
    if(check_counter != 2)
    {
        printf(ERROR_COLOR_PRINT);
        exit(ERROR_COLOR_EXIT);
    }

    char *pointer = strtok (char_color, sep);
    while (pointer != NULL)
    {
        for(int i = 0; i < strlen(pointer); i++)
        {
            if(!isdigit(pointer[i]))
            {
                printf(ERROR_COLOR_PRINT);
                exit(ERROR_COLOR_EXIT);
            }
        }
        if(atoi(pointer) < 0 || atoi(pointer) > 255)
        {
            printf(ERROR_COLOR_PRINT);
            exit(ERROR_COLOR_EXIT);
        }

        rgb_values[counter] = atoi(pointer);
        counter++;
        pointer = strtok (NULL,sep);
   }

    color.r = rgb_values[0];
    color.g = rgb_values[1];
    color.b = rgb_values[2];
    return color;
}

BMP read_bmp(char* file_name){
    BMP BMP_file;
    FILE *f = fopen(file_name, "rb");

    if(f == NULL)
    {
        printf(ERROR_OPENING_FILE_PRINT);
        exit(ERROR_OPENING_FILE_EXIT);
    }

    fread(&BMP_file.bmfh, 1, sizeof(BitmapFileHeader), f);
    fread(&BMP_file.bmih, 1, sizeof(BitmapInfoHeader), f);

    check_bmp_data(&BMP_file);
```

```
    fseek(f, BMP_file.bmfh.pixelArrOffset, SEEK_SET);

    unsigned int H = BMP_file.bmih.height;
    unsigned int W = BMP_file.bmih.width;

    BMP_file.img = (Rgb**)malloc(H * sizeof(Rgb*));
    if(BMP_file.img == NULL)
    {
        printf(ERROR_REALOCATING_MEMORY_PRINT);
        exit(ERROR_REALOCATING_MEMORY_EXIT);
    }

    for(int i = 0; i < H; i++)
    {
        BMP_file.img[i] = (Rgb*)malloc(W * sizeof(Rgb));
        if(BMP_file.img[i] == NULL)
        {
            printf(ERROR_REALOCATING_MEMORY_PRINT);
            exit(ERROR_REALOCATING_MEMORY_EXIT);
        }

        fread(BMP_file.img[i], 1, W * sizeof(Rgb) + getOffset(W), f);
    }

    fclose(f);
    return BMP_file;
}

void write_bmp(char file_name[], BMP BMP_file){
    FILE *ff = fopen(file_name, "wb");
    if(ff == NULL)
    {
        printf(ERROR_OPENING_FILE_PRINT);
        exit(ERROR_OPENING_FILE_EXIT);
    }

    unsigned int H = BMP_file.bmih.height;
    unsigned int W = BMP_file.bmih.width;

    fwrite(&BMP_file.bmfh, 1, sizeof(BitmapFileHeader), ff);
    fwrite(&BMP_file.bmih, 1, sizeof(BitmapInfoHeader), ff);

    fseek(ff, BMP_file.bmfh.pixelArrOffset, SEEK_SET);

    for(int i = 0; i < H; i++){
        fwrite(BMP_file.img[i], 1, W * sizeof(Rgb) + getOffset(W),ff);
    }
    fclose(ff);
}

void color_area(BMP* bmp_file, int x_left, int y_down, int x_right, int
y_up, Rgb* color)
{

    unsigned int H = bmp_file->bmih.height;
    unsigned int W = bmp_file->bmih.width;
```

```
    if (x_left < 0 || y_down < 0 || x_right < 0 || y_up < 0 || x_left >=
W || x_right >= W || y_down >= H || y_up >= H)
    {
        return;
    }

    if (y_down > y_up)
    {
        swap(&y_down, &y_up);
    }
    if (x_left > x_right)
    {
        swap(&x_left, &x_right);
    }

    for(int x = x_left; x <= x_right; x++)
    {
        for(int y = y_down; y <= y_up; y++)
        {
            color_pixel(bmp_file, x, y, color);
        }
    }
}

void draw_rectangle(BMP* bmp_file, Rgb* color, int thickness, int x_left,
int y_down, int x_right, int y_up)
{
    color_area(bmp_file, x_left, y_down, x_left + thickness - 1, y_up,
color);
    color_area(bmp_file, x_left, y_down, x_right, y_down + thickness -1,
color);
    color_area(bmp_file, x_left, y_up - thickness + 1, x_right, y_up,
color);
    color_area(bmp_file, x_right - thickness + 1, y_down, x_right, y_up,
color);
}
//first function
void color_replace(BMP* bmp_file, Rgb old_color, Rgb new_color)
{
    unsigned int H = bmp_file->bmih.height;
    unsigned int W = bmp_file->bmih.width;

    for(int y = 0; y < H; y++)
    {
        for(int x = 0; x < W; x++)
        {
            if(bmp_file->img[y][x].r   ==   old_color.r   &&   bmp_file-
>img[y][x].g == old_color.g && bmp_file->img[y][x].b == old_color.b)
            {
                color_pixel(bmp_file, x, y, &new_color);
            }
        }
    }
}

//second function
int ornament_check(int count, int thickness, char* pattern)
```

```
{
    int pattern_case = 0;

    if(!strcmp(pattern, "rectangle") || !strcmp(pattern, "semicircles"))
    {
        if(!strcmp(pattern, "rectangle"))
        {
            pattern_case = 1;
        }
        else if(!strcmp(pattern, "semicircles"))
        {
            pattern_case = 3;
        }

        if(count <= 0)
        {
            printf(ERROR_INVALID_COUNT_VALUE_PRINT);
            exit(ERROR_INVALID_COUNT_VALUE_EXIT);
        }
        if(thickness <= 0)
        {
            printf(ERROR_INVALID_THICKNESS_VALUE_PRINT);
            exit(ERROR_INVALID_THICKNESS_VALUE_EXIT);
        }
    }
    else if(!strcmp(pattern, "circle"))
    {
        pattern_case = 2;
    }
    else
    {
        printf(ERROR_INVALID_PATTERN_VALUE_PRINT);
        exit(ERROR_INVALID_PATTERN_VALUE_EXIT);
    }

    return pattern_case;
}

void rectangle_ornament(BMP* bmp_file, Rgb* color, int thickness, int
count)
{
    int x_left = 0;
    int y_down = 0;
    int x_right = bmp_file->bmih.width - 1;
    int y_up = bmp_file->bmih.height - 1;

    while(count !=0)
    {
        draw_rectangle(bmp_file,  color,  thickness,  x_left,  y_down,
x_right, y_up);
        count--;
        x_left = x_left + thickness*2;
        y_down = y_down + thickness*2;
        x_right = x_right - thickness*2;
        y_up = y_up - thickness*2;
    }
```

```
}

void circle_ornament(BMP* bmp_file, Rgb* color)
{
    int x_center = bmp_file->bmih.width / 2;
    int y_center = bmp_file->bmih.height / 2;

    int radius = 0;
    if(x_center > y_center){
        radius = y_center;
    }
    else{
        radius = x_center;
    }

    for (int x = 0; x < bmp_file->bmih.width; x++) {
        for (int y = 0; y < bmp_file->bmih.height; y++) {
            if (sqrt(pow(x - x_center, 2) + pow(y - y_center, 2)) >
radius) {
                color_pixel(bmp_file, x, y, color);
            }
        }
    }
}

void semicircle_ornament(BMP* bmp_file, Rgb* color, int thickness, int
count)
{
    unsigned int H = bmp_file->bmih.height;
    unsigned int W = bmp_file->bmih.width;

    int horizontal_circles_radius = 0;
    int vertical_circles_radius = 0;

    for(int r = 0; r < W; r++)
    {
        if(r*2*(count) + (count)*thickness > W)
        {
            horizontal_circles_radius = r;
            break;
        }
    }

    for(int r = 0; r < H; r++)
    {
        if(r*2*(count) + (count)*thickness > H)
        {
            vertical_circles_radius = r;
            break;
        }
    }

    //horizontal_ornament
    for(int above_down = 0; above_down < 2; above_down++)
    {
        int x_center = horizontal_circles_radius + ceil(thickness/2);
        int y_center;
```

```
        if(above_down == 0)
        {
            y_center = H - 1;
        }
        else
        {
            y_center = 0;
        }

        for(int i = 0; i < count; i++)
        {
            for(int x = 0; x < W; x++)
            {
                for(int y = 0; y < H; y++)
                {
                    double distance = sqrt(pow(abs(x - x_center), 2) +
pow(abs(y - y_center), 2));
                    if (distance >= horizontal_circles_radius && distance
<= horizontal_circles_radius + thickness)
                    {
                        color_pixel(bmp_file, x, y, color);
                    }
                }
            }
            x_center += thickness + 2*horizontal_circles_radius;
        }
    }
    //vertical_ornament
    for(int left_right = 0; left_right < 2; left_right++)
    {
        int x_center;
        int   y_center   =   H   -   1   -   (vertical_circles_radius   +
ceil(thickness/2));
        if(left_right == 0)
        {
            x_center = 0;
        }
        else
        {
            x_center = W - 1;
        }

        for(int i = 0; i < count; i++)
        {
            for(int x = 0; x < W; x++)
            {
                for(int y = 0; y < H; y++)
                {
                    double distance = sqrt(pow(abs(x - x_center), 2) +
pow(abs(y - y_center), 2));
                    if (distance >= vertical_circles_radius && distance
<= vertical_circles_radius + thickness)
                    {
                        color_pixel(bmp_file, x, y, color);
                    }
                }
            }
```

```
            y_center -= (thickness + 2*vertical_circles_radius);
        }
    }

}

void colored_pixel_check(int* flag, int x, int y, int colored_count,
int** start_colored_pixels, int** end_colored_pixels)
{
    if(colored_count == 0)
    {
        *flag = 0;
        return;
    }
    else
    {
        for(int t = 0; t <  colored_count; t++)
        {
            if(x     >=     start_colored_pixels[t][0]     &&     x     <=
end_colored_pixels[t][0]  &&  y  >=  start_colored_pixels[t][1]  &&  y  <=
end_colored_pixels[t][1])
            {
                *flag = 1;
                break;
            }
            else
            {
                *flag = 0;
            }
        }
    }
}

void filled_rects(BMP* bmp_file, Rgb* color, Rgb* border_color, int
thickness)
{
    unsigned int H = bmp_file->bmih.height;
    unsigned int W = bmp_file->bmih.width;

    int x_left = 0;
    int y_down = 0;
    int x_right = 0 ;
    int y_up = 0;

    int colored_count = 0;
    int flag = 0;
    int** start_colored_pixels = malloc(sizeof(int*)*2);
    if(start_colored_pixels == NULL)
    {
        printf(ERROR_REALOCATING_MEMORY_PRINT);
        exit(ERROR_REALOCATING_MEMORY_EXIT);
    }
    start_colored_pixels[colored_count] = malloc(sizeof(int) * 3);
    if(start_colored_pixels[colored_count] == NULL)
    {
        printf(ERROR_REALOCATING_MEMORY_PRINT);
        exit(ERROR_REALOCATING_MEMORY_EXIT);
```

```
    }

    int** end_colored_pixels = malloc(sizeof(int*)*2);
    if(end_colored_pixels == NULL)
    {
        printf(ERROR_REALOCATING_MEMORY_PRINT);
        exit(ERROR_REALOCATING_MEMORY_EXIT);
    }
    end_colored_pixels[colored_count] = malloc(sizeof(int)*3);
    if(end_colored_pixels[colored_count] == NULL)
    {
        printf(ERROR_REALOCATING_MEMORY_PRINT);
        exit(ERROR_REALOCATING_MEMORY_EXIT);
    }

    for(int x = 0; x < W; x++)
    {
        for(int y = 0; y < H; y++)
        {
            if(bmp_file->img[y][x].r == color->r && bmp_file->img[y][x].g
== color->g && bmp_file->img[y][x].b == color->b)
            {
                colored_pixel_check(&flag,    x,    y,    colored_count,
start_colored_pixels, end_colored_pixels);
                if(flag)
                {
                    flag = 0;
                    break;
                }

                x_left = x;
                x_right = x;
                y_down = y;
                y_up = y;
                for(int t = x; t < W; t++)
                {
                    if(bmp_file->img[y][t].r  ==  color->r  &&  bmp_file-
>img[y][t].g == color->g && bmp_file->img[y][t].b == color->b)
                    {
                        colored_pixel_check(&flag,  x,  t,  colored_count,
start_colored_pixels, end_colored_pixels);
                        if(flag)
                        {
                            flag = 0;
                            break;
                        }

                        x_right = t;
                    }
                    else
                    {
                        break;
                    }
                }
                for(int z = y; z < H; z++)
                {
                    for(int w = x_left; w <= x_right; w++)
```

```
                    {
                        colored_pixel_check(&flag, w, z, colored_count,
start_colored_pixels, end_colored_pixels);
                        if(flag)
                        {
                            flag = 0;
                            break;
                        }

                        if(!(bmp_file->img[z][w].r    ==    color->r    &&
bmp_file->img[z][w].g == color->g && bmp_file->img[z][w].b == color->b))
                        {
                            flag = 1;
                            break;
                        }
                        y_up = z;
                    }
                    if(flag)
                    {
                        flag = 0;
                        break;
                    }
                }

                start_colored_pixels[colored_count][0] = x_left;
                start_colored_pixels[colored_count][1] = y_down;
                end_colored_pixels[colored_count][0] = x_right;
                end_colored_pixels[colored_count][1] = y_up;
                colored_count++;

                start_colored_pixels    =    realloc(start_colored_pixels,
sizeof(int*)*(colored_count + 5));
                if(start_colored_pixels == NULL)
                {
                    printf(ERROR_REALOCATING_MEMORY_PRINT);
                    exit(ERROR_REALOCATING_MEMORY_EXIT);
                }

                start_colored_pixels[colored_count]                      =
malloc(sizeof(int)*3);
                if(start_colored_pixels[colored_count] == NULL)
                {
                    printf(ERROR_REALOCATING_MEMORY_PRINT);
                    exit(ERROR_REALOCATING_MEMORY_EXIT);
                }

                end_colored_pixels     =     realloc(end_colored_pixels,
sizeof(int*)*(colored_count + 5));
                if(end_colored_pixels == NULL)
                {
                    printf(ERROR_REALOCATING_MEMORY_PRINT);
                    exit(ERROR_REALOCATING_MEMORY_EXIT);
                }

                end_colored_pixels[colored_count]                        =
malloc(sizeof(int)*3);
                if(end_colored_pixels[colored_count] == NULL)
```

```
                {
                    printf(ERROR_REALOCATING_MEMORY_PRINT);
                    exit(ERROR_REALOCATING_MEMORY_EXIT);
                }

                //border
                if((y_down + thickness - 1 >= y_up) || x_left + thickness
- 1 >= x_right)
                {
                    color_area(bmp_file, x_left, y_down, x_right, y_up,
border_color);
                }
                else
                {
                    draw_rectangle(bmp_file,  border_color,  thickness,
x_left, y_down, x_right, y_up);
                }
            }
        }
    }

    for(int counter = 0; counter <= colored_count; counter++)
    {
        free(start_colored_pixels[counter]);
        free(end_colored_pixels[counter]);
    }
    free(start_colored_pixels);
    free(end_colored_pixels);

}

void gamma_func(BMP* bmp_file, double value)
{
    if(value <= 0)
    {
        printf(ERROR_INVALID_VALUE_PRINT);
        exit(ERROR_INVALID_VALUE_EXIT);
    }

    unsigned int H = bmp_file->bmih.height;
    unsigned int W = bmp_file->bmih.width;

    for(int y = 0; y < H; y++)
    {
        for(int x = 0; x < W; x++)
        {
            int old_r = bmp_file->img[y][x].r;
            int old_g = bmp_file->img[y][x].g;
            int old_b = bmp_file->img[y][x].b;

            double number = 255;

            double new_r = old_r / number;
            double new_g = old_g / number;
            double new_b = old_b / number;
```

```
            if(floor(powf( new_r, value) * 255) >= 0 && floor(powf(new_r,
value) * 255) <= 255)
            {
                bmp_file->img[y][x].r = (int)floor(powf( new_r, value) *
255);
            }
            if(floor(powf(new_g, value)*255) >= 0 && floor(powf(new_g,
value)*255) <= 255)
            {
                bmp_file->img[y][x].g      =      (int)floor(powf(new_g,
value)*255);
            }
            if(floor(powf(new_b, value)*255) >= 0 && floor(powf(new_b,
value)*255) <= 255)
            {
                bmp_file->img[y][x].b      =      (int)floor(powf(new_b,
value)*255);
            }
        }
    }
}

int main(int argc, char *argv[])
{
    if(argc < 2)
    {
        printf(ERROR_FEW_ARGS_PRINT);
        exit(ERROR_FEW_ARGS_EXIT);
    }

    const char *short_options = "hiI:o:CO:n:rp:c:t:u:fb:gv:";

    const struct option long_options[] =
        {
            {"help", no_argument, 0, 'h'},
            {"info", no_argument, 0, 'i'},
            {"input", required_argument, 0, 'I'},
            {"output", required_argument, 0, 'o'},
            {"color_replace", no_argument, 0, 'C'},
            {"old_color", required_argument, 0, 'O'},
            {"new_color", required_argument, 0, 'n'},
            {"ornament", no_argument, 0, 'r'},
            {"pattern", required_argument, 0, 'p'},
            {"color", required_argument, 0, 'c'},
            {"thickness", required_argument, 0, 't'},
            {"count", required_argument, 0, 'u'},
            {"filled_rects", no_argument, 0, 'f'},
            {"border_color", required_argument, 0, 'b'},
            {"gamma", no_argument, 0, 'g'},
            {"value", required_argument, 0, 'v'},
            {0, 0, 0, 0}
        };

    char *input_file = argv[argc - 1];
    char *output_file = "out.bmp";
```

```
    int opt = 0;
    int option_index = 0;
    int option = 0;

    char* char_old_color = NULL;
    char* char_new_color = NULL;
    char* pattern = NULL;
    char* char_color = NULL;
    int thickness = -1;
    int count = -1;
    char* char_border_color = NULL;
    double value = 0;

        while ((opt = getopt_long(argc, argv, short_options,
long_options, &option_index)) != -1)
    {
        switch (opt)
        {
            case 'h':
            {
                printHelp();
                exit(0);
                break;
            };
            case 'i':
            {
                option = 4;
                break;
            };
            case 'o':
            {
                output_file = optarg;
                break;
            };
            case 'I':
            {
                input_file = optarg;
                break;
            };
            case 'C':
            {
                option = 1;
                break;
            };
            case 'O':
            {
                char_old_color = optarg;
                break;
            };
            case 'n':
            {
                char_new_color = optarg;
                break;
            };
            case 'r':
            {
                option = 2;
```

```
            break;
        };
        case 'p':
        {
            pattern = optarg;
            break;
        };
        case 'c':
        {
            char_color = optarg;
            break;
        };
        case 't':
        {
            thickness = atoi(optarg);
            break;
        };
        case 'u':
        {
            count = atoi(optarg);
            break;
        };
        case 'f':
        {
            option = 3;
            break;
        };
        case 'b':
        {
            char_border_color = optarg;
            break;
        };
        case 'g':
        {
            option = 5;
            break;
        };
        case 'v':
        {
            value = atof(optarg);
            break;
        };
        case '?':
        {
            printf(WRONG_OPTION_PRINT);
            exit(WRONG_OPTION_EXIT);
            break;
        }
    }
}

BMP bmp_file = read_bmp(input_file);

unsigned int H = bmp_file.bmih.height;
unsigned int W = bmp_file.bmih.width;

switch(option)
```

```
    {
        case 1:
        {
            Rgb old_color = get_color(char_old_color);
            Rgb new_color = get_color(char_new_color);

            color_replace(&bmp_file, old_color, new_color);

            break;
        }

        case 2:
        {

            Rgb color = get_color(char_color);

            int pattern_case = ornament_check(count, thickness, pattern);

            switch(pattern_case)
            {
                case 1:
                {
                    rectangle_ornament(&bmp_file,   &color,   thickness,
count);
                    break;
                }
                case 2:
                {
                    circle_ornament(&bmp_file, &color);
                    break;
                }
                case 3:
                {

                    semicircle_ornament(&bmp_file,   &color,   thickness,
count);
                    break;
                }
            }


            break;
        }

        case 3:
        {
            Rgb color = get_color(char_color);
            Rgb border_color = get_color(char_border_color);

            if(thickness <= 0)
            {
               printf(ERROR_INVALID_THICKNESS_VALUE_PRINT);
               exit(ERROR_INVALID_THICKNESS_VALUE_EXIT);
            }

            filled_rects(&bmp_file, &color, &border_color, thickness);
            break;
```

```
        }

        case 4:
        {
            printFileHeader(bmp_file.bmfh);
            printInfoHeader(bmp_file.bmih);
            exit(0);
            break;
        }

        case 5:
        {
            gamma_func(&bmp_file, value);
            break;
        }
        default:
        {
            printf(WRONG_OPTION_PRINT);
            exit(WRONG_OPTION_EXIT);
            break;
        }
    }

    write_bmp(output_file, bmp_file);

    image_free(&bmp_file);

    return 0;
}
```