

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

КУРСОВАЯ РАБОТА
по дисциплине «Программирование»
Тема: Обработка BMP изображений на языке программирования Си

Студент гр. 3381

Иванов А.А.

Преподаватель

Глазунов С.А.

Санкт-Петербург

2024

ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

Студент Иванов А.А.

Группа 3381

Тема работы: Обработка BMP изображений на языке программирования Си

Исходные данные:

Вариант 5.3

Программа **обязательно должна иметь CLI** (опционально дополнительное использование GUI). Более подробно тут:

http://se.moevm.info/doku.php/courses:programming:rules_extra_kurs

Программа должна реализовывать весь следующий функционал по обработке bmp-файла

Общие сведения

- 24 бита на цвет
- без сжатия
- файл может не соответствовать формату BMP, т.е. необходимо проверка на BMP формат (дополнительно стоит помнить, что версий у формата несколько). Если файл не соответствует формату BMP или его версии, то программа должна завершиться с соответствующей ошибкой.
- обратите внимание на выравнивание; мусорные данные, если их необходимо дописать в файл для выравнивания, должны быть нулями.
- обратите внимание на порядок записи пикселей
- все поля стандартных BMP заголовков в выходном файле должны иметь те же значения что и во входном (разумеется кроме тех, которые должны быть изменены).

Программа должна иметь следующие функции по обработке изображений:

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

2. Рисование квадрата. Флаг для выполнения данной операции: `--square`.

Квадрат определяется:

- Координатами левого верхнего угла. Флаг `--left_up`, значение задаётся в формате `left.up`, где `left` – координата по x, `up` – координата по y
- Размером стороны. Флаг `--side_size`. На вход принимает число больше 0
- Толщиной линий. Флаг `--thickness`. На вход принимает число больше 0
- Цветом линий. Флаг `--color` (цвет задаётся строкой `rrr.ggg.bbb`, где `rrr/ggg/bbb` – числа, задающие цветовую компоненту. пример `--color 255.0.0` задаёт красный цвет)
- Может быть залит или нет. Флаг `--fill`. Работает как бинарное значение: флага нет – `false`, флаг есть – `true`.
- Цветом которым он залит, если пользователем выбран залитый. Флаг `--fill_color` (работает аналогично флагу `--color`)

3. Поменять местами 4 куса области. Флаг для выполнения данной операции: `--exchange`. Выбранная пользователем прямоугольная область делится на 4 части и эти части меняются местами. Функционал определяется:

- Координатами левого верхнего угла области. Флаг `--left_up`, значение задаётся в формате `left.up`, где `left` – координата по x, `up` – координата по y
- Координатами правого нижнего угла области. Флаг `--right_down`, значение задаётся в формате `right.down`, где `right` – координата по x, `down` – координата по y

- Способом обмена частей: “по кругу”, по диагонали. Флаг `--exchange_type`, возможные значения: `clockwise`, `counterclockwise`, `diagonals`

4. Находит самый часто встречаемый цвет и заменяет его на другой заданный цвет. Флаг для выполнения данной операции: `--freq_color`.

Функционал определяется:

- Цветом, в который надо перекрасить самый часто встречаемый цвет. Флаг `--color` (цвет задаётся строкой `rrr.ggg.bbb`, где `rrr/ggg/bbb` – числа, задающие цветовую компоненту. пример `--color 255.0.0` задаёт красный цвет)

Каждую подзадачу следует вынести в отдельную функцию, функции сгруппировать в несколько файлов (например, функции обработки текста в один, функции ввода/вывода в другой). Сборка должна осуществляться при помощи `make` и `Makefile` или другой системы сборки

Содержание пояснительной записки:

«Содержание», «Введение», «Выполнение работы», «Заключение», «Список использованных источников»

Предполагаемый объем пояснительной записки:

Не менее 30 страниц.

Дата выдачи задания: 24.04.2024

Дата сдачи реферата: 30.04.2024

Дата защиты реферата: 04.05.2024

Студент

Иванов А.А.

Преподаватель

Глазунов С.А.

АННОТАЦИЯ

Курсовая работа написана на языке программирования Си, она принимает на вход BMP изображение без сжатия и 24 битами на цвет и обрабатывает его в соответствии с выбранными опциями.

Изменения, которые нужно произвести с изображением, подаются с помощью CLI интерфейса, внутри программа обрабатывает полученные флаги и в соответствии с ними обрабатывает подаваемое на вход изображение. Ошибки во всех функциях возвращаются в главную функцию и обрабатываются. В случае если обнаруживается ошибка, вызывается функция очистки динамической памяти и происходит завершение работы программы с кодом ошибки из диапазона [40;49] (коды ошибок описаны в файле `exceptions.h`).

SUMMARY

The course work is written in the C programming language, it accepts a BMP image without compression and 24 bits per color as input and processes it according to the selected options.

The changes that need to be made to the image are submitted using the CLI interface, inside the program processes the received flags and processes the image submitted to the input in accordance with them. Errors in all functions are returned to the main function and processed. If an error is detected, the dynamic memory cleanup function is called and the program is terminated with an error code from the range [40;49] (error codes are described in the `exceptions.h` file).

СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....	7
1. ВЫПОЛНЕНИЕ РАБОТЫ.....	8
1.1. Исследование.....	8
1.2. Написание кода программы.....	8
ЗАКЛЮЧЕНИЕ.....	13
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ.....	14
РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ.....	15
ИСХОДНЫЙ КОД ПРОГРАММЫ.....	24

ВВЕДЕНИЕ

Целью курсовой работы является — изучить внутреннее устройство BMP файлов, научиться производить считывание и обработки этих файлов на языке программирования Си. Для этих целей **нужно** реализовать следующий функционал:

1. Определить структуры для хранения мета-информации BMP файлов и структуру, содержащую информацию о цвете пикселя
2. Написать функции чтения/записи данных BMP файла. При чтении необходимо сохранить метаданные и массив пикселей изображения, также необходима проверка соответствия файла формату. При записи, полученные данные должны быть записаны корректно, изображение должно нормально открываться
3. Реализовать интерфейс командной строки (CLI) для программы. С помощью него производить управление программой. Для этого внутри программы требуется обработать флаги командной строки с помощью функции `getopt_long`
4. Реализовать функции для выполнения операций с изображением.
5. Написать `Makefile` для сборки программы
6. Протестировать программу на утечки памяти и корректное выполнение всех опций.

1. ВЫПОЛНЕНИЕ РАБОТЫ

1.1. Исследование

Для начала необходимо понять, из чего состоит BMP файл. Он состоит из BMP Header, DIB Header, массива пикселей и некоторых дополнительных данных, причём между header'ами и массивом пикселей есть отступ, значение которого хранится внутри BMP Header [1]. В рамках данной работы мы работаем только с заголовками BMP файла и массивом пикселей.

По требованиям курсовой работы, необходимо работать с цветом, под который отведены 24 бита, по 1-му байту на компоненту цвета: красный, зелёный и жёлтый. Для хранения каждого пикселя необходима структура, содержащая поля, хранящие компоненты цвета. Так как значения компонент имеют значения от 0 до 255 включительно, то для хранения каждой хватит типа данных `uint8_t`. К тому же внутри структуры необходимо расположить компоненты в обратном порядке, так как при хранении пикселей используется little-endian порядок.

1.2. Написание кода программы

Структура проекта:

- `main.c` — основной файл программы, в котором подключаются все заголовочные файлы и выполняется вся работа
- `Makefile` — файл сборки проекта
- `lib/` — директория, содержащая все исходные файлы проекта:
 - `bmp.c` — файл, содержащий функции чтения и записи BMP файла
 - `etc.c` — содержит функцию смены двух `int64_t` чисел местами. Функция была вынесена в отдельный файл, так как она необходима в нескольких исходных файлах.

- `exchange.c` — содержит функции, которые реализовывают замену местами частей области (3-я подзадача)
- `freq_color.c` — содержит функцию, которая реализовывает поиск самого часто встречаемого цвета на изображении (4-я подзадача)
- `parse_funcs.c` — содержит функции обработки аргументов флагов командной строки
- `print_funcs.c` — содержит функции: вывод предупреждающего сообщения (при отсутствии аргументов программы) и печать помощи (если был введён флаг `--help` или `-h`)
- `rgbfilter.c` — содержит функцию замены определённой компоненты цвета на заданное значение во всём изображении (1-я подзадача)
- `draw.c` — содержит функции, необходимые для рисования заданного квадрата (2-я подзадача)
- `include/` — директория, содержащая все заголовочные файлы проекта, которые необходимы для подключения исходных файлов из `lib/` в `main.c`. Из них следует выделить только файл `exceptions.h`: в нём реализовано перечисление, содержащие коды ошибок программы и макросы обработки этих ошибок

Файл `main.c`:

Подключённые заголовочные файлы:

- `#include <stdio.h>` - использование функций ввода и вывода
- `#include <unistd.h>` - использование функций записи/чтения из файла, функции побайтового перемещения внутри файлы (`fseek`)
- `#include <stdint.h>` - использование целочисленных типов с заданным размером
- `#include <stdlib.h>` - функции выделения и очистки памяти

- `#include <getopt.h>` - использование функций и переменных для реализации CLI
- `#include <string.h>` - использование функций работы со строками

Структуры:

- `Config` — хранение данных о наличии флагов
- `Optargs` — хранение аргументов флагов.

Функции:

- `void free_memory(Config config, Optarg optargs, RGB*** arr, BitmapInfoHeader* bmih)` — функция для очистки всей выделенной динамической памяти.
- `int main(int argc, char** argv)` — главная функция программы.

Функция `main`:

Первой строкой выводится информация о создателе работы и выполненном варианте. Далее, проверяется наличие переданных аргументов, если, помимо имени программы, ничего не было передано, программа завершает работу с выводом информации об ошибке.

Иначе происходит:

1. Определение и инициализация экземпляров структур `Config` (`Config config`) и `Optarg` (`Optarg optargs`)
2. Переменные для хранения символа (`int32_t optchar`), возвращённого `getopt_long` и возвращаемого значения программы соответственно (`int32_t ret_val`)
3. Определяются структуры, представляющие BMP и DIB header'ы (`BitmapFileHeader bmfh` и `BitmapInfoHeader bmih`)
4. Определяется и инициализируется указатель на первый элемент массива пикселей (`RGB** arr`)

5. Определение и инициализация строки, содержащей информацию о коротких флагах и структуры, содержащей информацию о длинных флагах интерфейса

Далее, в цикле `while` происходит последовательное считывание флагов командной строки с помощью функции `getopt_long` [2], возвращаемое значение записывается в переменную `optchar`, а аргумент (если он требуется) записывается в глобальную переменную `optarg`.

Внутри цикла обрабатываются возвращаемые функцией `getopt_long` символы и заполняются переменные `config` и `optargs`. При обработке некоторых флагов используются функции из файла `lib/parse_funcs.c`. Если был получен флаг `--help`, программа завершается на этом этапе и выводит информацию по функционалу программы.

После цикла, если было переопределено имя входного или выходного файла, оно записывается в `optargs`. Если имя входного файла не было переопределено, то именем входного файла считается последний аргумент командной строки. Если имя входного и выходного файла совпало, программа завершается с ошибкой.

Далее считывается BMP файл и производятся действия в соответствии с выбранными флагами:

- `-i --info` — выводится сообщение с информацией о BMP файле
- `-r --rgbfilter:`
 - Если не был обнаружен флаг `---component_name` или флаг `--component_value` — программа завершается с ошибкой.
 - Иначе вызывается функция `rgbfilter` и, в зависимости от возвращённого результата, программа либо завершается с ошибкой, либо продолжает выполнение. Внутри функции `rgbfilter` происходит обработка полученной компоненты цвета. Если это существующая

компонента, то она заменяется на всём изображении последовательным проходом по каждому пикселю изображения

- `-s --square:`
 - Если не были обнаружены необходимые флаги — программа завершается с ошибкой
 - Если был присутствует лишь один из флагов для заливки — программа завершается с ошибкой
 - Иначе, вызывается функция `draw_square` и проверяется наличие ошибок. Функция `draw_square` сначала проверяет, нужно ли залить квадрат, если да — сразу его заливает. После этого вычисляются координаты углов квадрата и рисуются его границы с помощью функции `draw_line`. Функция `draw_line` реализовывает алгоритм Брезенхема для рисования линий, толщина достигается рисованием залитых кругов определённого цвета вдоль заданной линии. Все проверки нахождения линий на изображении происходят в функции `draw_line`.
- `-e --exchange:`
 - Если не были обнаружены необходимые флаги — программа завершается с сообщением об ошибке
 - Иначе выполняется функция `exchange` и проверяется наличие ошибок. Функция `exchange` сначала обрабатывает координаты, приводит их к доступному для обработки виду. После происходит выбор, каким способом необходимо поменять части области местами. Внутри каждой из функций смены частей области происходит проход по одной части области, при проходе вычисляются соответствующие им координаты в других областях. Далее пиксели меняются местами.
- `-f --freq_color:`
 - Если отсутствуют необходимые флаги — программа выводит сообщение об ошибке и завершает работу

- Иначе выполняется функция `freq_color`. Функция `freq_color` последовательно проходит по изображению и ищет самый часто встречаемый цвет. Каждый цвет заносится в специальный массив умножением компонент на соответствующие им степени 256. С помощью этого массива достигается линейная скорость обращения к количеству каждого из найденных на изображении цветов. После мы снова последовательно проходим по изображению и заменяем самый часто встречаемый цвет на цвет, переданный аргументом функции.

После проверки всех флагов, новое изображение записывается в файл, имя которого содержится в структуре `optargs` и программа завершает свою работу.

ЗАКЛЮЧЕНИЕ

По итогу была изучена структура BMP файла, написаны функции считывания и записи этого типа файлов с помощью программы. Реализованы функции изменения компонент изображения, рисования квадрата, манипулирования частями изображения и поиска самого часто встречающегося цвета. Аргументы для реализованных функций передаются с помощью CLI интерфейса, реализованного с помощью функции `getopt_long` стандартной библиотеки. Полученная программа выполняет все поставленные перед ней задачи, отсутствуют утечки памяти, все ошибки при выполнении обрабатываются.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. BMP file format // wikipedia.org. URL:
https://en.wikipedia.org/wiki/BMP_file_format (дата обращения:
21.04.2024).
2. getopt(3) — Linux manual page // man7.org. URL:
<https://man7.org/linux/man-pages/man3/getopt.3.html> (дата обращения:
20.04.2024).

ПРИЛОЖЕНИЕ А РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ

Исходное изображение (для help и info):



Вызов help:
./cw --help

```
artyom@artyom-laptop:~/Git_repos/2nd_sem/pr-2024-3381/Ivanov_Artem_cw/src$ valgrind ./cw --help
==26652== Memcheck, a memory error detector
==26652== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==26652== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==26652== Command: ./cw --help
==26652==
Course work for option 5.3, created by Artem Ivanov
Usage: ./cw [OPTIONS] FILE
-h, --help          print this help and exit
-I, --info          print info about bmp file
-i, --input [INP_NAME]
```

<описание флагов>


```

-E, --exchange_type [clockwise|counterclockwise|diagonals]
    flag to choose mode to change image parts
-f, --freq_color
    finds and replaces with the most frequent color to argument of flag --color
==26652==
==26652== HEAP SUMMARY:
==26652==   in use at exit: 0 bytes in 0 blocks
==26652==   total heap usage: 1 allocs, 1 frees, 1,024 bytes allocated
==26652==
==26652== All heap blocks were freed -- no leaks are possible
==26652==
==26652== For lists of detected and suppressed errors, rerun with: -s
==26652== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)

```

Вызов info:

```
./cw --info math.bmp
```

```

artyom@artyom-laptop:~/Git_repos/2nd_sem/pr-2024-3381/Ivanov_Artem_cw/src$ ./cw --
info math.bmp
Course work for option 5.3, created by Artem Ivanov
File was succesfully read!
BitmapFileHeader:
signature:      4d42 (19778)
filesize:      29e062 (2744418)
reserved1:      0 (0)
reserved2:      0 (0)
pixelArrOffset: 8a (138)

BitmapInfoHeader:
headerSize:     7c (124)
width: 438 (1080)
height: 34f (847)
planes: 1 (1)
bitsPerPixel:   18 (24)
compression:    0 (0)
imageSize:     29dfd8 (2744280)
xPixelsPerMeter: 0 (0)
yPixelsPerMeter: 0 (0)
colorsInColorTable: 0 (0)
importantColorCount: 0 (0)

```

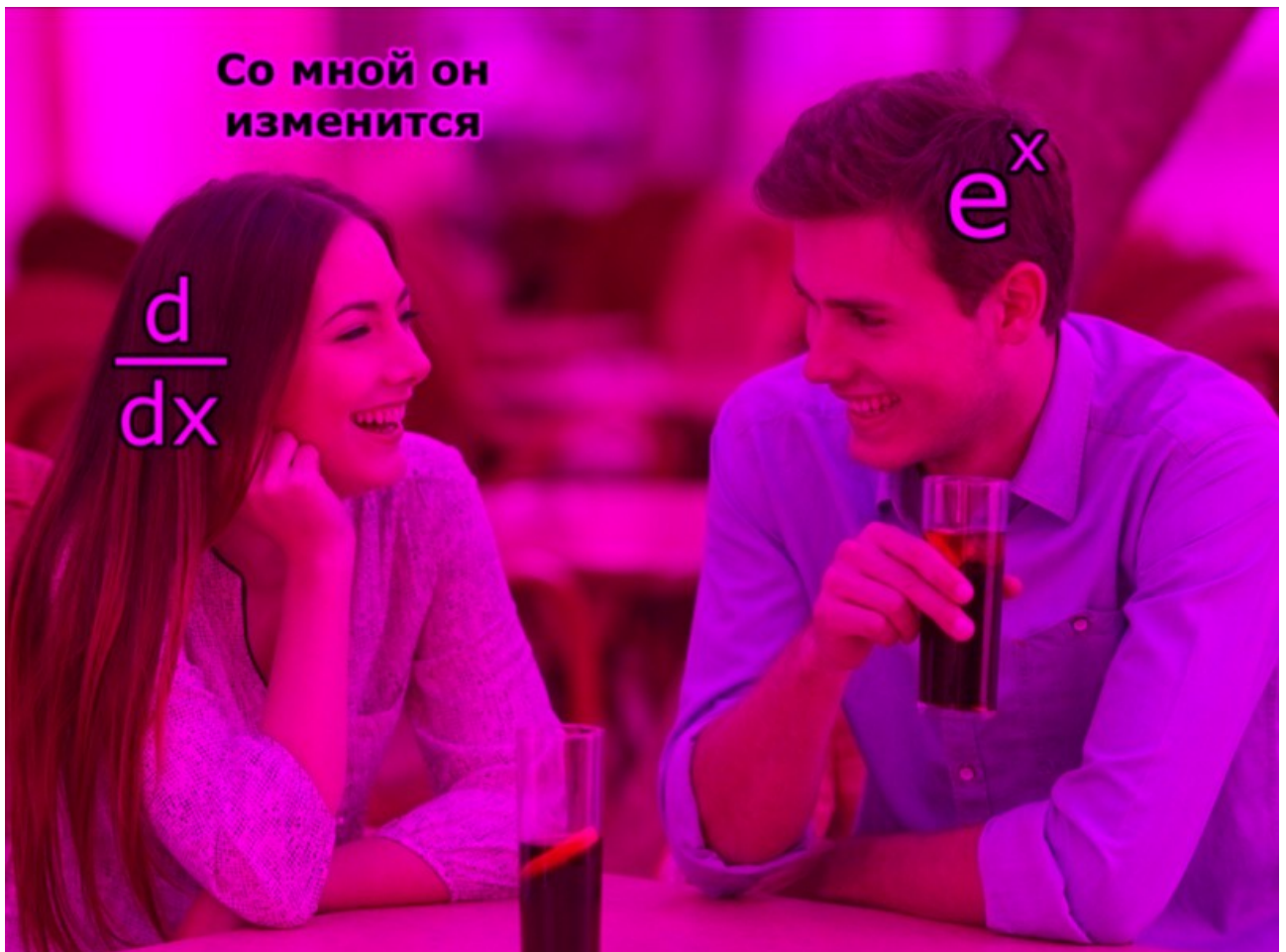
Вызов `rgbfilter` без входного файла:

```
./cw --rgbfilter --component_name green --component_value 0
```

```
artyom@artyom-laptop:~/Git_repos/2nd_sem/pr-2024-3381/Ivanov_Artem_cw/src$ ./cw --  
rgbfilter --component_name green --component_value 0  
Course work for option 5.3, created by Artem Ivanov  
Error: The 0 file was not open
```

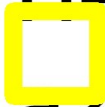
Вызов `rgbfilter`:

```
./cw --rgbfilter --component_name green --component_value 0  
error.bmp
```



Вызов square:


```
./cw --square --left_up 50.100 --side_size 100 --thickness 20  
--color 255.255.0 yozh.bmp
```

Не «отчислился из универа»,
 а «аль денте бакалавр»



Вызов square с заливкой:

```
./cw --square --left_up 50.100 --side_size 100 --thickness 20  
--color 255.255.0 --fill --fill_color 0.255.255 yozh.bmp
```

Не «отчислился из универа»,
 а «аль денте бакалавр»



Вызов square с неправильным значением флага:

```
./cw --square --left_up 50.100 --side_size 100 --thickness -2  
--color 255.255.0 yozh.bmp
```

```
artyom@artyom-laptop:~/Git_repos/2nd_sem/pr-2024-3381/Ivanov_Artem_cw/src$ ./cw --  
square --left_up 50.100 --side_size 100 --thickness -2 --color 255.255.0 yozh.bmp  
--fill --fill_color 0.255.255  
Course work for option 5.3, created by Artem Ivanov  
Error: Component value must be greater than 0
```

Вызов square с заливкой и выходящими за пределы изображения координатами:

```
./cw --square --left_up -50.100 --side_size 100 --thickness 20  
--color 176.55.33 yozh.bmp --fill --fill_color 0.2.37 wisdom.bmp
```



Вызов exchange:

```
./cw --exchange --left_up 100.150 --right_down 300.250  
--exchange_type clockwise fear.bmp
```



Вызов exchange с выходящими за предел изображения координатами:
./cw --exchange --left_up -100.150 --right_down 300.250
--exchange_type clockwise theboys.bmp



***5 заявления на академ,
пожалуйста.***

Вызов `freq_color`:

```
./cw --freq_color --color 0.255.255 pain.bmp
```



Вызов `freq_color` без необходимого флага:

```
./cw --freq_color pain.bmp
```

```
artyom@artyom-laptop:~/Git_repos/2nd_sem/pr-2024-3381/Ivanov_Artem_cw/src$ ./cw --  
freq_color pain.bmp  
Course work for option 5.3, created by Artem Ivanov  
File was succesfully read!  
Error: --freq_color need --color flag!
```


ПРИЛОЖЕНИЕ Б

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название главного исходного файла: `main.c`

Сборка программы осуществляется с помощью `make`, инструкции содержатся в файле: `Makefile`

main.c:

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

#include "include/bmp.h"
#include "include/exceptions.h"
#include "include/print_funcs.h"
#include "include/rgbfilter.h"
#include "include/draw.h"
#include "include/exchange.h"
#include "include/freq_color.h"
#include "include/parse_funcs.h"

// flags struct
typedef struct {
    int8_t image_readed;
    int8_t image_written;
    int8_t info;
    int8_t input;
    int8_t output;
    int8_t rgbfilter;
    int8_t component_name;
    int8_t component_value;
    int8_t square;
    int8_t left_up;
    int8_t side_size;
    int8_t thickness;
    int8_t color;
    int8_t fill;
    int8_t fill_color;
    int8_t freq_color;
    int8_t exchange;
    int8_t right_down;
    int8_t exchange_type;
} Config;

// flags args struct
typedef struct {
    char*    input;
    char*    output;
```

```

    char*      component_name;
    uint8_t    component_value;
    int64_t*   left_up;
    uint32_t   side_size;
    uint32_t   thickness;
    RGB        color;
    RGB        fill_color;
    int64_t*   right_down;
    char*      exchange_type;
} Optarg;

void free_memory(Config config, Optarg optargs, RGB*** arr,
BitmapInfoHeader* bmih);

int main(int argc, char** argv)
{
    printf("Course work for option 5.3, created by Artem
Ivanov\n");

    if (argc == 1) {
        print_warn_msg(argv[0]);
        return NO_ERROR;
    }

    Config config = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0};
    Optarg optargs = {NULL, NULL, NULL, 0, NULL, 0, 0, {0, 0, 0},
{0, 0, 0}, NULL, NULL};

    int32_t optchar;
    int32_t ret_val = 0;
    // to use custom error messages
    // opterr = 0;

    BitmapFileHeader bmfh;
    BitmapInfoHeader bmih;
    RGB** arr = NULL;

    char* short_options = "hIi:o:rn:v:sS:T:c:lC:eR:E:f";
    struct option long_options[] = {
        {"help", no_argument, NULL, 'h'},
        {"info", no_argument, NULL, 'I'},
        {"input", required_argument, NULL, 'i'},
        {"output", required_argument, NULL, 'o'},

        {"rgbfilter", no_argument, NULL, 'r'},
        {"component_name", required_argument, NULL, 'n'},
        {"component_value", required_argument, NULL, 'v'},

        {"square", no_argument, NULL, 's'},
        {"left_up", required_argument, NULL, 't'},
        {"side_size", required_argument, NULL, 'S'},
        {"thickness", required_argument, NULL, 'T'},

```

```

{"color", required_argument, NULL, 'c'},
{"fill", no_argument, NULL, 'l'},
{"fill_color", required_argument, NULL, 'C'},

{"exchange", no_argument, NULL, 'e'},
{"right_down", required_argument, NULL, 'R'},
{"exchange_type", required_argument, NULL, 'E'},
{"freq_color", no_argument, NULL, 'f'},
{NULL, no_argument, NULL, 0}
};

// all configuration logic and parse optargs arguments
// must be in this while loop
while ((optchar = getopt_long(argc, argv, short_options,
long_options,
                                &optind)) != -1) {
    switch (optchar) {
        case 'h': // -h --help
            print_help(argv[0]);
            free_memory(config, optargs, &arr, &bmi);
            return NO_ERROR;
        case 'I': // -I --info
            config.info = 1;
            break;
        case 'i': // -i --input
            // if (optarg != NULL)
            // this check useless if opterr != 0
            // otherwise, a check is needed along with the
error output
            // else { "error throw logic" }

            config.input = 1;
            optargs.input = strdup(optarg);
            break;
        case 'o': // -o --output
            config.output = 1;
            optargs.output = strdup(optarg);
            break;
        case 'r': // --rgbfilter
            config.rgbfilter = 1;
            break;
        case 'n': // --component_name
            config.component_name = 1;
            optargs.component_name = strdup(optarg);
            break;
        case 'v': // --component_value
            config.component_value = 1;
            ret_val = parse_unsigned_char(optarg,
&(optargs.component_value));

            // check if parse was succesful
            if (ret_val != PARSE_ERROR)
                break;
    }
}

```

```

        free_memory(config, optarg, &arr, &bmi);
        return ARG_ERROR;
    case 's': // --square
        config.square = 1;
        break;
    case 't': // --left_up
        config.left_up = 1;
        ret_val = parse_coords(optarg,
&(optargs.left_up));

        if (ret_val != PARSE_ERROR)
            break;

        free_memory(config, optarg, &arr, &bmi);
        return ARG_ERROR;
    case 'S': // --size_size
        config.side_size = 1;
        ret_val = parse_posit_number(optarg,
&(optargs.side_size));

        if (ret_val != PARSE_ERROR)
            break;

        free_memory(config, optarg, &arr, &bmi);
        return ARG_ERROR;
    case 'T': // --thickness
        config.thickness = 1;
        // parse_side_size contains the necessary logic
        ret_val = parse_posit_number(optarg,
&(optargs.thickness));

        if (ret_val != PARSE_ERROR)
            break;

        free_memory(config, optarg, &arr, &bmi);
        return ARG_ERROR;
    case 'c': // --color
        config.color = 1;
        ret_val = parse_comps(optarg, &(optargs.color));

        if (ret_val != PARSE_ERROR)
            break;

        free_memory(config, optarg, &arr, &bmi);
        return ARG_ERROR;
    case 'l': // --fill
        config.fill = 1;
        break;
    case 'C': // --fill_color
        if (config.fill) {
            config.fill_color = 1;

```

```

        ret_val = parse_comps(optarg,
&(optargs.fill_color));
        if (ret_val != PARSE_ERROR)
            break;
    }

    free_memory(config, optargs, &arr, &bmi);
    return ARG_ERROR;
case 'e': // --exchange
    config.exchange = 1;
    break;
case 'R': // --right_down
    config.right_down = 1;
    ret_val = parse_coords(optarg,
&(optargs.right_down));

    if (ret_val != PARSE_ERROR)
        break;

    free_memory(config, optargs, &arr, &bmi);
    return ARG_ERROR;
case 'E': // --exchange_type
    config.exchange_type = 1;
    optargs.exchange_type = strdup(optarg);
    if (!strcmp(optarg, "clockwise") ||
        !strcmp(optarg, "counterclockwise") ||
        !strcmp(optarg, "diagonals")) {
        break;
    }
    free_memory(config, optargs, &arr, &bmi);
    return ARG_ERROR;
case 'f': // --freq_color
    config.freq_color = 1;
    break;
case '?': // unknown flag
    free_memory(config, optargs, &arr, &bmi);
    return ARG_ERROR;
}
}

if (!config.input) {
    config.input = 1;
    optargs.input = strdup(argv[argc - 1]);
}

if (!config.output) {
    config.output = 1;
    optargs.output = strdup("out.bmp");
}
// check if input and output filename is match
// but if --info flag is chosen, this not important
if (!strcmp(optargs.input, optargs.output) && !config.info) {

```

```

        free_memory(config, optarg, &arr, &bmi);
        error_return("Input and output file names musn't be the
same!\n", ARG_ERROR);
    }

    // input bmp file information
    ret_val = read_bmp(optarg, &arr, &bmf, &bmi);
    if (ret_val) {
        free_memory(config, optarg, &arr, &bmi);
        return ret_val;
    } else
        config.image_readed = 1;

    // print file information
    if (config.info) {
        printf("BitmapFileHeader:\n");
        print_file_header(bmf);
        printf("\nBitmapInfoHeader:\n");
        print_info_header(bmi);

        free_memory(config, optarg, &arr, &bmi);
        return NO_ERROR;
    }

    // 1. rgbfilter subtask
    if (config.rgbfilter) {
        if (!config.component_name || !config.component_value) {
            free_memory(config, optarg, &arr, &bmi);
            error_return("Missing flags to --rgbfilter, type
--help to more information\n",
                        ARG_ERROR);
        } else {
            ret_val = rgbfilter(&arr, &bmi,
                                config.component_name,
                                config.component_value);
            if (ret_val) {
                free_memory(config, optarg, &arr, &bmi);
                return ret_val;
            }
        }
    }

    // 2. square subtask
    if (config.square) {
        // check for necessary flags
        if (!config.left_up || !config.side_size ||
            !config.thickness || !config.color) {
            free_memory(config, optarg, &arr, &bmi);
            error_return("Missing flags to --square, type --help
for more information\n",
                        ARG_ERROR);
        }
        // check for the necessary flags

```

```

        } else if (!config.fill && config.fill_color ||
                    config.fill && !config.fill_color) {
            free_memory(config, optarg, &arr, &bmih);
            error_return("Flag --fill and --fill_color need to be
used together\n", ARG_ERROR);
        } else {
            ret_val = draw_square(&arr, &bmih, optarg.left_up,
                                optarg.side_size,
                                optarg.thickness,
                                optarg.color,
                                config.fill,
                                optarg.fill_color);
            if (ret_val) {
                free_memory(config, optarg, &arr, &bmih);
                return ret_val;
            }
        }

// 3. exchange subtask
if (config.exchange) {
    if (!config.left_up || !config.right_down || !
config.exchange_type) {
        free_memory(config, optarg, &arr, &bmih);
        error_return("--exchange need flags!\n", ARG_ERROR);
    } else {
        ret_val = exchange(&arr, &bmih, optarg.left_up,
                           optarg.right_down,
                           optarg.exchange_type);
        if (ret_val) {
            free_memory(config, optarg, &arr, &bmih);
            return ret_val;
        }
    }
}

// 4. freq_color subtask
if (config.freq_color) {
    if (!config.color) {
        free_memory(config, optarg, &arr, &bmih);
        error_return("--freq_color need --color flag!\n",
ARG_ERROR);
    }

    ret_val = freq_color(&arr, &bmih, optarg.color);
    if (ret_val) {
        free_memory(config, optarg, &arr, &bmih);
        return ret_val;
    }
}

// write changes in file
ret_val = write_bmp(optarg.output, &arr, &bmfh, &bmih);

```

```

    if (ret_val) {
        free_memory(config, optargs, &arr, &bmih);
        return ret_val;
    } else
        config.image_written = 1;

    // for (; optind < argc; optind++) {
    //     printf("extra arg is %s\n", argv[optind]);
    // }

    free_memory(config, optargs, &arr, &bmih);
    return NO_ERROR;
}

void free_memory(Config config, Optarg optargs, RGB*** arr,
BitmapInfoHeader* bmih)
{
    // free pixels array
    if (config.image_readed && (*arr) != NULL) {
        for (size_t i = 0; i < bmih->height; i++)
            free((*arr)[i]);
        free(*arr);
    } // free optargs
    if (config.input)
        free(optargs.input);
    if (config.output)
        free(optargs.output);
    if (config.component_name)
        free(optargs.component_name);
    if (config.left_up)
        free(optargs.left_up);
    if (config.right_down)
        free(optargs.right_down);
    if (config.exchange_type)
        free(optargs.exchange_type);
}

```

Makefile:

```

PROJ_NAME = cw
CC = gcc
FLAGS =-g -c

OBJ = main.o \
      bmp.o \
      print_funcs.o \
      rgbfilter.o \
      parse_funcs.o \
      draw.o \
      exchange.o \
      freq_color.o \

```



```

        dictionary.o \
        etc.o

main: $(OBJ)
    $(CC) -g -Wall -Werror -o $(PROJ_NAME) $(OBJ) -lm
    rm *.o

main.o: main.c
    $(CC) $(FLAGS) main.c

bmp.o: lib/bmp.c include/bmp.h
    $(CC) $(FLAGS) lib/bmp.c

print_funcs.o: lib/print_funcs.c include/print_funcs.h
    $(CC) $(FLAGS) lib/print_funcs.c

rgbfilter.o: lib/rgbfilter.c include/rgbfilter.h
    $(CC) $(FLAGS) lib/rgbfilter.c

parse_funcs.o: lib/parse_funcs.c include/parse_funcs.h
    $(CC) $(FLAGS) lib/parse_funcs.c

draw.o: lib/draw.c include/draw.h
    $(CC) $(FLAGS) lib/draw.c

exchange.o: lib/exchange.c include/exchange.h
    $(CC) $(FLAGS) lib/exchange.c

freq_color.o: lib/freq_color.c include/freq_color.h
    $(CC) $(FLAGS) lib/freq_color.c

dictionary.o: lib/dictionary.c include/dictionary.h
    $(CC) $(FLAGS) lib/dictionary.c

etc.o: lib/etc.c include/etc.h
    $(CC) $(FLAGS) lib/etc.c

```

Файлы директории lib:

bmp.c:

```

#include "../include/bmp.h"

int32_t read_bmp(const char* file_name, RGB*** arr,
                 BitmapFileHeader* bmfh, BitmapInfoHeader* bmif)
{
    // to do special logic
    FILE* fd = fopen(file_name, "rb");
    if (fd == NULL) {
        error_return_warg("The %s file was not open\n", IO_ERROR,
            file_name);
    }
}

```

```

// simplified verification form
if (!fread(bmfh, sizeof(BitmapFileHeader), 1, fd)) {
    error_return_wfd("BitmapFileHeader was no read\n",
                    BMP_FORMAT_ERROR, fd);
}
if (bmfh->signature != 0x4D42) {
    // exit from program with error
    error_return_wfd("BMP file signature doesn't match BM\n",
                    BMP_FORMAT_ERROR, fd);
}

// simplified verification form
if (!fread(bmif, sizeof(BitmapInfoHeader), 1, fd)) {
    error_return_wfd("BitmapInfoHeader was no read\n",
                    IO_ERROR, fd);
}

// turn on after find image without compression and with 24-
bit color
if (bmif->bitsPerPixel != 24 || bmif->compression != 0) {
    error_return_wfd("This program can't process BMP image
like this\n",
                    BMP_FORMAT_ERROR, fd);
}
fseek(fd, bmfh->pixelArrOffset, SEEK_SET);

uint32_t H = bmif->height;
uint32_t W = bmif->width;

uint64_t padded_width = W * sizeof(RGB) + (4 - W * sizeof(RGB)
% 4) % 4;

(*arr) = (RGB**)calloc(H, sizeof(RGB*));
if ((*arr) == NULL) {
    error_return_wfd("Allocating memory to pixels array lines
return NULL",
                    ALLOC_ERROR, fd);
}
for (int64_t i = 0; i < H; i++) {
    (*arr)[i] = (RGB*)calloc(padded_width, 1);
    if ((*arr)[i] == NULL) {
        error_return_wfd("Allocating memory to pixels array
line return NULL\n",
                        ALLOC_ERROR, fd);
    }

    fread((*arr)[i], padded_width, 1, fd);
    /*if (!fread((*arr)[i], padded_width, 1, fd)) {
        error_return_wfd("Pixels array line was not read\n",
IO_ERROR, fd);
    }*/
}

```

```

        fclose(fd);
        printf("File was succesfully read!\n");

        return NO_ERROR;
    }

int32_t write_bmp(const char* file_name, RGB*** arr,
                  const BitmapFileHeader* bmfh, const
                  BitmapInfoHeader* bmih)
{
    int32_t H = bmih->height;
    int32_t W = bmih->width;
    uint64_t padded_width = W * sizeof(RGB) + (4 - W * sizeof(RGB)
% 4) % 4;

    FILE* fd = fopen(file_name, "wb");
    if (fd == NULL) {
        error_return_warg("This %s file was not open\n",
                          IO_ERROR, file_name);
    }

    if (fwrite(bmfh, sizeof(BitmapFileHeader), 1, fd) != 1) {
        error_return_wfd("BitmapFileHeader was not written\n",
        IO_ERROR, fd);
    }

    if (fwrite(bmih, sizeof(BitmapInfoHeader), 1, fd) != 1) {
        error_return_wfd("BitmapInfoHeader was not written\n",
        IO_ERROR, fd);
    }

    fseek(fd, bmfh->pixelArrOffset, SEEK_SET);

    for (int64_t i = 0; i < H; i++) {
        fwrite((*arr)[i], padded_width, 1, fd);
        /*if (fwrite((*arr)[i], padded_width, 1, fd) <
padded_width) {
            error_return_wfd("Pixels array line no written\n",
        IO_ERROR, fd);
        }*/
    }

    fclose(fd);
    printf("File was succesfully written!\n");

    return NO_ERROR;
}

void print_file_header(BitmapFileHeader header)
{
    printf("signature:\t%x (%hu)\n", header.signature,
    header.signature);
}

```

```

        printf("filesize:\t%x (%u)\n", header.fileSize,
header.fileSize);
        printf("reserved1:\t%x (%hu)\n", header.reserved1,
header.reserved1);
        printf("reserved2:\t%x (%hu)\n", header.reserved2,
header.reserved2);
        printf("pixelArrOffset:\t%x (%u)\n", header.pixelArrOffset,
header.pixelArrOffset);
    }

void print_info_header(BitmapInfoHeader header)
{
    printf("headerSize:\t%x (%u)\n", header.headerSize,
header.headerSize);
    printf("width:\t%x (%u)\n", header.width, header.width);
    printf("height:\t%x (%u)\n", header.height, header.height);
    printf("planes:\t%x (%hu)\n", header.planes, header.planes);
    printf("bitsPerPixel:\t%x (%hu)\n", header.bitsPerPixel,
header.bitsPerPixel);
    printf("compression:\t%x (%u)\n", header.compression,
header.compression);
    printf("imageSize:\t%x (%u)\n", header.imageSize,
header.imageSize);
    printf("xPixelsPerMeter:\t%x (%u)\n", header.xPixelsPerMeter,
header.xPixelsPerMeter);
    printf("yPixelsPerMeter:\t%x (%u)\n", header.yPixelsPerMeter,
header.yPixelsPerMeter);
    printf("colorsInColorTable:\t%x (%u)\n",
header.colorsInColorTable, header.colorsInColorTable);
    printf("importantColorCount:\t%x (%u)\n",
header.importantColorCount, header.importantColorCount);
}

```

draw.c:

```

#include "../include/draw.h"

void draw_8pixels(RGB*** arr, int64_t x0, int64_t y0,
                 int64_t x, int64_t y, int32_t thickness,
                 uint32_t H, uint32_t W, RGB color)
{
    fill_circle(arr, x + x0, y + y0, thickness, H, W, color);
    fill_circle(arr, x + x0, -y + y0, thickness, H, W, color);
    fill_circle(arr, -x + x0, -y + y0, thickness, H, W, color);
    fill_circle(arr, -x + x0, y + y0, thickness, H, W, color);
    fill_circle(arr, y + x0, x + y0, thickness, H, W, color);
    fill_circle(arr, y + x0, -x + y0, thickness, H, W, color);
    fill_circle(arr, -y + x0, -x + y0, thickness, H, W, color);
    fill_circle(arr, -y + x0, x + y0, thickness, H, W, color);
}

// preparation to cw protection

```

```

void draw_circle(RGB*** arr, int64_t x0, int64_t y0, int32_t
radius,
                int32_t thickness, uint32_t H, uint32_t W, RGB
color,
                uint8_t is_fill, RGB fill_color)
{
    int64_t hor_dist;
    int64_t diag_dist;
    int64_t dist;

    int64_t x = 0;
    int64_t y = radius;

    if (is_fill) {
        fill_circle(arr, x0, y0, radius, H, W, fill_color);
    }

    dist = 3 - 2 * y;
    while (x <= y) {
        draw_8pixels(arr, x0, y0, x, y, thickness, H, W, color);
        if (dist < 0)
            dist = dist + 4 * x + 6;
        else {
            dist = dist + 4 * (x - y) + 10;
            y--;
        }
        x++;
    }
}

void fill_circle(RGB*** arr, int64_t x0, int64_t y0, int32_t
radius,
                uint32_t H, uint32_t W, RGB fill_color)
{
    for (int32_t i = -radius; i <= radius; i++) {
        for (int32_t j = -radius; j <= radius; j++) {
            // real coord values
            int64_t x_r = x0 + j;
            int64_t y_r = y0 - i;
            if (i * i + j * j <= radius * radius &&
                x_r >= 0 && x_r < W && y_r >= 0 && y_r < H)
                (*arr)[y_r][x_r] = fill_color;
        }
    }
}

// brezenham line drawing algorithm
int32_t draw_line(RGB*** arr,
                int64_t x0, int64_t y0, int64_t x1, int64_t y1,
                uint32_t H, uint32_t W, int32_t thickness, RGB
color)
{
    if (x0 != x1) {

```

```

        if (x0 > x1) {
            swap_int64(&x0, &x1);
            swap_int64(&y0, &y1);
        }

        double k = (double)(y1 - y0) / (x1 - x0);
        double b = (double)y0 - (double)k * x0;
        for (int32_t x = x0; x < x1 + 1; x++) {
            int32_t y = k * x + b;
            fill_circle(arr, x, y, thickness / 2, H, W, color);
        }
    } else {
        if (y0 > y1)
            swap_int64(&y0, &y1);

        for (int32_t y = y0; y < y1; y++) {
            fill_circle(arr, x0, y, thickness / 2, H, W, color);
        }
    }

    return NO_ERROR;
}

int32_t draw_square(RGB*** arr, const BitmapInfoHeader* bmih,
                    int64_t* left_up, uint32_t side_size, uint32_t
thickness,
                    RGB color, uint8_t is_fill, RGB fill_color)
{
    int64_t H = bmih->height;
    int64_t W = bmih->width;

    // x and y swaped to be nice for test system -
    // need to fix later
    int64_t x0 = left_up[0];
    int64_t y0 = H - left_up[1] - 1;

    int64_t delta_left = thickness / 2;
    int64_t delta_right = (thickness % 2) ? (thickness / 2)
                                           : (thickness / 2 - 1);

    if (is_fill && side_size > 2) {
        for (int64_t y = y0; y > y0 - side_size; y--) {
            for (int64_t x = x0; x < x0 + side_size - 1; x++) {
                if (x >= 0 && x < W && y >= 0 && y < H)
                    (*arr)[y][x] = fill_color;
            }
        }
    }

    // left up corner
    int64_t x_lu = x0;
    int64_t y_lu = y0;

```

```

    // right up corner
    int64_t x_ru = x_lu + side_size - 1;
    int64_t y_ru = y_lu;

    // right down corner
    int64_t x_rd = x_ru;
    int64_t y_rd = y_ru - (side_size - 1);

    // left down corner
    int64_t x_ld = x_lu;
    int64_t y_ld = y_rd;

    draw_line(arr, x_lu, y_lu, x_ru, y_ru, H, W, thickness,
color);
    draw_line(arr, x_ru, y_ru, x_rd, y_rd, H, W, thickness,
color);
    draw_line(arr, x_rd, y_rd, x_ld, y_ld, H, W, thickness,
color);
    draw_line(arr, x_ld, y_ld, x_lu, y_lu, H, W, thickness,
color);

    return NO_ERROR;
}

```

etc.c:

```

#include "../include/etc.h"

void swap_int64(int64_t *a, int64_t *b)
{
    int64_t tmp = *a;
    *a = *b;
    *b = tmp;
}

```

exchange.c:

```

#include "../include/exchange.h"

// delta x and delta y
int64_t dx;
int64_t dy;

// coords after mid of area
int64_t aft_mid_x;
int64_t aft_mid_y;

void swap_rgb(RGB* a, RGB* b)
{
    RGB tmp = *a;
    *a = *b;
    *b = tmp;
}

```

```

}

void rotate_clockwise(RGB*** arr, int64_t x0,  int64_t y0,
                    int64_t x1, int64_t y1, uint32_t W, uint32_t
H)
{
    for (int64_t x_lu = x0; x_lu < aft_mid_x; x_lu++) {
        for (int64_t y_lu = y0; y_lu > aft_mid_y; y_lu--) {

            // x and y for right down area
            int64_t x_rd = aft_mid_x + labs(x_lu - x0);
            int64_t y_rd = aft_mid_y - labs(y_lu - y0);

            // x and y for left down area
            int64_t x_ld = x_lu;
            int64_t y_ld = y_rd;

            // x and y for right up area
            int64_t x_ru = x_rd;
            int64_t y_ru = y_lu;

            // left up with left down
            swap_rgb(&(*arr)[y_lu][x_lu], &(*arr)[y_ld][x_ld]);
            // right up with right down
            swap_rgb(&(*arr)[y_ru][x_ru], &(*arr)[y_rd][x_rd]);
            // curr right up (prev right down) with curr left down
            (prev left_up)
            swap_rgb(&(*arr)[y_ru][x_ru], &(*arr)[y_ld][x_ld]);
        }
    }
}

void rotate_cclockwise(RGB*** arr, int64_t x0,  int64_t y0,
int64_t x1, int64_t y1, uint32_t W, uint32_t H)
{
    for (int64_t x_lu = x0; x_lu < aft_mid_x; x_lu++) {
        for (int64_t y_lu = y0; y_lu > aft_mid_y; y_lu--) {

            // x and y for right down area
            int64_t x_rd = aft_mid_x + labs(x_lu - x0);
            int64_t y_rd = aft_mid_y - labs(y_lu - y0);

            // x and y for left down area
            int64_t x_ld = x_lu;
            int64_t y_ld = y_rd;

            // x and y for right up area
            int64_t x_ru = x_rd;
            int64_t y_ru = y_lu;

            // right down with right up

```



```

        swap_rgb(&(*arr)[y_rd][x_rd], &(*arr)[y_ru][x_ru]);
        // left up with left down
        swap_rgb(&(*arr)[y_lu][x_lu], &(*arr)[y_ld][x_ld]);
        // curr left up (prev left down) with right down (prev
right up)
        swap_rgb(&(*arr)[y_lu][x_lu], &(*arr)[y_rd][x_rd]);
    }
}

void rotate_diagonal(RGB*** arr, int64_t x0, int64_t y0, int64_t
x1, int64_t y1, uint32_t W, uint32_t H)
{
    // iterating throw left up area
    for (int64_t x_lu = x0; x_lu < aft_mid_x; x_lu++) {
        for (int64_t y_lu = y0; y_lu > aft_mid_y; y_lu--) {

            // x and y for right down area
            int64_t x_rd = aft_mid_x + labs(x_lu - x0);
            int64_t y_rd = aft_mid_y - labs(y_lu - y0);

            // x and y for left down area
            int64_t x_ld = x_lu;
            int64_t y_ld = y_rd;

            // x and y for right up area
            int64_t x_ru = x_rd;
            int64_t y_ru = y_lu;

            // left up and right down
            swap_rgb(&(*arr)[y_lu][x_lu], &(*arr)[y_rd][x_rd]);
            // left down and right up
            swap_rgb(&(*arr)[y_ld][x_ld], &(*arr)[y_ru][x_ru]);
        }
    }
}

// processing only good sized image
int32_t exchange(RGB ***arr, const BitmapInfoHeader *bmih,
int64_t *left_up, int64_t *right_down,
char *exchange_type)
{
    uint32_t W = bmih->width;
    uint32_t H = bmih->height;

    int64_t x0 = left_up[0];
    int64_t y0 = H - left_up[1] - 1;
    int64_t x1 = right_down[0];
    int64_t y1 = H - right_down[1] - 1;

    // standartize x coords
    if (x0 < 0)
        x0 = 0;

```

```

    if (x0 >= W)
        x0 = W - 1;
    if (x1 < 0)
        x0 = 0;
    if (x1 >= W)
        x0 = W - 1;

    // standartize y coords
    if (y0 < 0)
        y0 = 0;
    if (y0 >= H)
        y0 = H - 1;
    if (y1 < 0)
        y1 = 0;
    if (y1 >= H)
        y1 = H - 1;

    // standarize points
    if (x0 > x1)
        swap_int64(&x0, &x1);
    if (y0 < y1)
        swap_int64(&y0, &y1);

    // change coords
    if ((x1 - x0 + 1) % 2 != 0)
        x1++;
    if ((y0 - y1 + 1) % 2 != 0)
        y1--;

    dx = labs(x1 - x0);
    dy = labs(y1 - y0);

    aft_mid_x = x0 + dx / 2/* + 1*/;
    aft_mid_y = y0 - dy / 2/* - 1*/;

    if (!strcmp(exchange_type, "clockwise")) {
        rotate_clockwise(arr, x0, y0, x1, y1, W, H);
    } else if (!strcmp(exchange_type, "counterclockwise")) {
        rotate_cclockwise(arr, x0, y0, x1, y1, W, H);
    } else if (!strcmp(exchange_type, "diagonals")) {
        rotate_diagonal(arr, x0, y0, x1, y1, W, H);
    } else
        error_return("Unknown argument for --exchange_type
flag\n", ARG_ERROR);

    return NO_ERROR;
}

```

freq_color.c:

```
#include "../include/freq_color.h"
```

```

int32_t freq_color(RGB*** arr, const BitmapInfoHeader *bmih, RGB
color)
{
    int32_t ret_val;

    uint32_t W = bmih->width;
    uint32_t H = bmih->height;

    // this realization is fast but use a lot of memory
    RGB max_color = {0, 0, 0};
    uint32_t max_count = 0;

    // def and init colors array
    const uint64_t colors_len = 256*256*256 + 256*256 + 256;
    uint32_t *colors = (uint32_t*)calloc(colors_len,
sizeof(uint32_t));

    // find most freq color
    for (uint32_t i = 0; i < H; i++) {
        for (uint32_t j = 0; j < W; j++) {
            RGB curr_color = (*arr)[i][j];
            uint8_t r = curr_color.r;
            uint8_t g = curr_color.g;
            uint8_t b = curr_color.b;
            uint64_t ind = 256*256*r + 256*g + b;

            colors[ind]++;
            if (colors[ind] > max_count) {
                max_count = colors[ind];
                max_color = curr_color;
            }
        }
    }

    for (uint32_t i = 0; i < H; i++) {
        for (uint32_t j = 0; j < W; j++) {
            RGB curr_color = (*arr)[i][j];
            if (curr_color.r == max_color.r &&
                curr_color.g == max_color.g &&
                curr_color.b == max_color.b) {
                (*arr)[i][j] = color;
            }
        }
    }

    printf("The most frequent color was succesfully replaced!\n");
    printf("Max freq color is (%u,%u,%u)\n",
        max_color.r, max_color.g, max_color.b);
    printf("Max color count is [%u]\n", max_count);

    free(colors);

    return NO_ERROR;
}

```

```
}
```

parse_funcs.c:

```
#include "../include/parse_funcs.h"

// -1 if parse failed else parsed number
int32_t parse_unsigned_char(const char* arg, uint8_t* val)
{
    regex_t regex;
    int32_t reti = regcomp(&regex, "^([01]?[0-9]?[0-9]|2[0-4][0-9]|25[0-5])$", REG_EXTENDED);
    if (reti) {
        error_return("Could not compile regex\n", PARSE_ERROR);
    }

    reti = regexec(&regex, arg, 0, NULL, 0);
    if (!reti) {
        *val = atoi(arg);
        regfree(&regex);
        return NO_ERROR;
    } else if (reti == REG_NOMATCH) {
        regfree(&regex);
        error_return("Component value must be in range [0..255]\n",
                    PARSE_ERROR);
    } else {
        char err_buf[100];
        regerror(reti, &regex, err_buf, sizeof(err_buf));
        regfree(&regex);
        error_return_warg("Regex match failed %s\n", PARSE_ERROR,
err_buf);
    }
}

int32_t parse_coords(const char* arg, int64_t** val_arr)
{
    const uint32_t max_groups = 3;
    regex_t regex;
    regmatch_t groups[max_groups];

    const char* reg_str = "^(-?[0-9]+)\\.\\.(-?[0-9]+)$";
    int32_t reti = regcomp(&regex, reg_str, REG_EXTENDED);
    if (reti) {
        error_return("Could not compile regex\n", PARSE_ERROR);
    }

    reti = regexec(&regex, arg, max_groups, groups, 0);
    uint8_t buf_ind = 0;
    char buffer[100];

    // alloc memory to array with x and y component
```

```

        *val_arr = (int64_t*)malloc((max_groups - 1) *
sizeof(int64_t));
        if (!reti) {
            /*val_arr =
            for (size_t i = 1; i < max_groups; i++) {
                if (groups[i].rm_so == -1)
                    break;

                for (size_t j = groups[i].rm_so; j <
groups[i].rm_eo; j++)
                    //printf("%c", arg[j]);
                    buffer[buf_ind++] = arg[j];

                buffer[buf_ind] = '\0';
                (*val_arr)[i - 1] = atoi(buffer);
                buf_ind = 0;
            }

            regfree(&regex);

            return NO_ERROR;
        } else if (reti == REG_NOMATCH) {
            regfree(&regex);
            error_return("Components values must be greater or
equal to 0\n",
                        PARSE_ERROR);
        } else {
            char err_buf[100];
            regerror(reti, &regex, err_buf, sizeof(err_buf));
            regfree(&regex);
            error_return_warg("Regex match failed %s\n", PARSE_ERROR,
err_buf);
        }
    }

int32_t parse_posit_number(const char* arg, uint32_t* val)
{
    regex_t regex;
    int32_t reti = regcomp(&regex, "[1-9][0-9]*$", REG_EXTENDED);
    if (reti)
        error_return("Could not compile regex\n", PARSE_ERROR);

    reti = regexec(&regex, arg, 0, NULL, 0);
    if (!reti) {
        *val = atoi(arg);
        regfree(&regex);
        return NO_ERROR;
    } else if (reti == REG_NOMATCH) {
        regfree(&regex);
        error_return("Component value must be greater than 0\n",
                    PARSE_ERROR);
    } else {
        char err_buf[100];

```

```

        regerror(reti, &regex, err_buf, sizeof(err_buf));
        regfree(&regex);
        error_return_warg("Regex match failed %s\n", PARSE_ERROR,
err_buf);
    }
}

int32_t parse_comps(const char* arg, RGB *color)
{
    const uint32_t max_groups = 4;
    regex_t regex;
    regmatch_t groups[max_groups];

    const char* reg_str = "^[01]?[0-9]?[0-9]|2[0-4][0-9]|25[0-
5])"
                        "\\."
                        "([01]?[0-9]?[0-9]|2[0-4][0-9]|25[0-5])"
                        "\\."
                        "([01]?[0-9]?[0-9]|2[0-4][0-9]|25[0-5])
$";
    int32_t reti = regcomp(&regex, reg_str, REG_EXTENDED);
    if (reti) {
        error_return("Could not compile regex\n", PARSE_ERROR);
    }

    reti = regexec(&regex, arg, max_groups, groups, 0);
    uint8_t buf_ind = 0;
    char buffer[100] = "";

    // alloc memory to array with r, g and b components
    if (!reti) {
        for (size_t i = 1; i < max_groups; i++) {
            if (groups[i].rm_so == -1)
                break;

            for (size_t j = groups[i].rm_so; j <
groups[i].rm_eo; j++)
                buffer[buf_ind++] = arg[j];

            buffer[buf_ind] = '\\0';
            //(*val_arr)[i - 1] = atoi(buffer);
            if (i - 1 == 0)
                color->r = atoi(buffer);
            else if (i - 2 == 0)
                color->g = atoi(buffer);
            else if (i - 3 == 0)
                color->b = atoi(buffer);
            buf_ind = 0;
        }

        regfree(&regex);
        return NO_ERROR;
    } else if (reti == REG_NOMATCH) {

```

```

        regfree(&regex);
        error_return("Component values must be in range
[0..255]\n",
                    PARSE_ERROR);
    } else {
        char err_buf[100];
        regerror(reti, &regex, err_buf, sizeof(err_buf));
        regfree(&regex);
        error_return_warg("Regex match failed %s\n", PARSE_ERROR,
err_buf);
    }
}

```

print_funcs.c:

```

#include "../include/print_funcs.h"

void print_warn_msg(char* exe_name)
{
    printf("%s: missing arguments\n", exe_name);
    printf("Try \'%s --help\' for more information\n", exe_name);
}

void print_help(char* exe_name)
{
    printf("Usage: %s [OPTIONS] FILE\n", exe_name);
    printf("%4s %-40s\n\t%-30s\n", "-h,", "--help", "print this
help and exit");
    printf("%4s %-40s\n\t%-30s\n", "-I,", "--info", "print info
about bmp file");
    printf("%4s %-40s\n\t%-30s\n", "-i,", "--input [INP_NAME]",
"set up file to processing");
    printf("%4s %-40s\n\t%-30s\n", "-o,", "--output [OUT_NAME]",
"change standard output file name");

    printf("%4s %-40s\n\t%-30s\n", "-r,", "--rgbfilter", "changes
chosen RGB component of image, need flags --component_name and
--component_value to work");
    printf("%4s %-40s\n\t%-30s\n", "-n,", "--component_name [red|
green|blue]", "flag to choose component to be replaced");
    printf("%4s %-40s\n\t%-30s\n", "-v,", "--component_value
[0..255]", "flag to enter new value of component");

    printf("%4s %-40s\n\t%-30s\n", "-s,", "--square", "draws
square, for work need flag --left_up, --side_size, --thickness,
--color and optionally argument --fill and --fill_color");
    printf("%4s %-40s\n\t%-30s\n", " ", "--left_up [LEFT.UP]",
"entering coordinates of the upper-left corner of the square");
    printf("%4s %-40s\n\t%-30s\n", "-S,", "--side_size
[POSITIVE_NUMBER]", "entering side size of square");
    printf("%4s %-40s\n\t%-30s\n", "-T,", "--thickness
[POSITIVE_NUMBER]", "entering thickness of square side");
}

```

```

    printf("%4s %-40s\n\t%-30s\n", "-c,", "--color [R.G.B]",
"entering color components");
    printf("%4s %-40s\n\t%-30s\n", "-l,", "--fill", "it works as
a flag and indicates whether a square should be filled in");
    printf("%4s %-40s\n\t%-30s\n", "-C,", "--fill_color [R.G.B]",
"selecting the color to fill the square with if the --fill flag
was entered");

    printf("%4s %-40s\n\t%-30s\n", "-e,", "--exchange", "flag to
change image parts, need flags --left_up, --right_down and
--exchange_type");
    printf("%4s %-40s\n\t%-30s\n", "-R,", "--right_down
[RIGHT.DOWN]", "flag to choose right down corner of segment to be
changed");
    printf("%4s %-40s\n\t%-30s\n", "-E,", "--exchange_type
[clockwise|counterclockwise|diagonals]", "flag to choose mode to
change image parts");
    printf("%4s %-40s\n\t%-30s\n", "-f,", "--freq_color", "finds
and replaces with the most frequent color to argument of flag
--color");
}

```

rgbfilter.c:

```

#include "../include/rgbfilter.h"

// checks if compt_name match "red", "green" or "blue"
uint8_t identify(const char* compt_name)
{
    if (!strcmp("red", compt_name))
        return 'r';
    else if (!strcmp("green", compt_name))
        return 'g';
    else if (!strcmp("blue", compt_name))
        return 'b';
    else
        return '\0';
}

int32_t rgbfilter(RGB*** arr, const BitmapInfoHeader* bmih,
                  const char* compt_name, const uint8_t compt_val)
{
    const char indicator = identify(compt_name);

    switch (indicator) {
        case 'r':
            for (size_t i = 0; i < bmih->height; i++) {
                for (size_t j = 0; j < bmih->width; j++)
                    (*arr)[i][j].r = compt_val;
            }
            printf("Red component was changed to %u\n",
compt_val);

```



```

        return NO_ERROR;
    case 'g':
        for (size_t i = 0; i < bmih->height; i++) {
            for (size_t j = 0; j < bmih->width; j++)
                (*arr)[i][j].g = compt_val;
        }
        printf("Green component was changed to %u\n",
compt_val);
        return NO_ERROR;
    case 'b':
        for (size_t i = 0; i < bmih->height; i++) {
            for (size_t j = 0; j < bmih->width; j++)
                (*arr)[i][j].b = compt_val;
        }
        printf("Blue component was changed to %u\n",
compt_val);
        return NO_ERROR;
    default:
        error_return("The --component_name flag takes one of
the arguments: red, blue, green\n", ARG_ERROR);
    }
}

```

Файлы директории include:

bmp.h:

```

#ifndef __BMP__
#define __BMP__

#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include <unistd.h>

#include "exceptions.h"

#pragma pack(push, 1)
typedef struct {
    uint16_t signature;           // BMP file identifier
    uint32_t filesize;           // BMP filesize in bytes
    uint16_t reserved1;          // reserved space
    uint16_t reserved2;          //
    uint32_t pixelArrOffset;     // offset (pixels array bytes
offset)
} BitmapFileHeader;

typedef struct {
    uint32_t headerSize;         // header size in bytes
    uint32_t width;              // image width
    uint32_t height;             // image height
    uint16_t planes;             // count of color planes (m/b 1)

```

```

        uint16_t bitsPerPixel;           // color depth (1, 4, 8, 16, 24,
32)
        uint32_t compression;           // compression method
        uint32_t imageSize;             // image size
        uint32_t xPixelsPerMeter;       // horizontal resolution of
image
        uint32_t yPixelsPerMeter;       // vertical resolution of image
        uint32_t colorsInColorTable;    // number of colors in the color
palette
        uint32_t importantColorCount;    // number of important colors, 0
if all colors is important
    } BitmapInfoHeader;

// thats little-endian.
typedef struct {
    uint8_t b;
    uint8_t g;
    uint8_t r;
} RGB;
#pragma pack(pop)

int32_t read_bmp(const char* file_name, RGB*** arr,
                BitmapFileHeader* bmfh, BitmapInfoHeader* bmif);
int32_t write_bmp(const char* file_name, RGB*** arr,
                const BitmapFileHeader* bmfh, const
BitmapInfoHeader* bmif);

void print_file_header(BitmapFileHeader header);
void print_info_header(BitmapInfoHeader header);

#endif

```

draw.h:

```

#ifndef __DRAW__
#define __DRAW__

#include <stdint.h>
#include <math.h>

#include "bmp.h"
#include "exceptions.h"
#include "etc.h"

typedef struct {
    uint8_t up_hor;
    uint8_t down_hor;
    uint8_t left_vert;
    uint8_t right_vert;
} DrawConf;

void draw_8pixels(RGB*** arr, int64_t x0, int64_t y0,

```

```

        int64_t x, int64_t y, int32_t thickness,
        uint32_t H, uint32_t W, RGB color);

void draw_circle(RGB*** arr, int64_t x0, int64_t y0, int32_t
radius,
                int32_t thickness, uint32_t H, uint32_t W, RGB
color,
                uint8_t is_fill, RGB fill_color);

void fill_circle(RGB*** arr, int64_t x0, int64_t y0, int32_t
radius,
                uint32_t H, uint32_t W, RGB color);

int32_t draw_line(RGB*** arr,
                int64_t x0, int64_t y0, int64_t x1, int64_t y1,
                uint32_t H, uint32_t W, int32_t thickness, RGB
color);

int32_t draw_square(RGB*** arr, const BitmapInfoHeader* bmih,
                int64_t* left_up, uint32_t side_size, uint32_t
thickness,
                RGB color, uint8_t is_fill, RGB fill_color);

#endif

```

etc.h:

```

#ifndef __ETC__
#define __ETC__

#include <stdint.h>

void swap_int64(int64_t *a, int64_t *b);

#endif

```

exceptions.h:

```

#ifndef __EXCEPTIONS__
#define __EXCEPTIONS__

#include <stdio.h>

extern enum {
    NO_ERROR = 0,
    IO_ERROR = 40,
    ALLOC_ERROR = 41,
    BMP_FORMAT_ERROR = 42,
    ARG_ERROR = 43,
    PARSE_ERROR = 44,
    POINTER_ERROR = 45,

```

```

    DICT_ERROR = 46
} external_exception_t;

#define error_return(msg, error_code) \
    ({ fprintf(stderr, "Error: "); \
      fprintf(stderr, msg); \
      return error_code; })

// error_return with file descriptor
#define error_return_wfd(msg, error_code, fd) \
    ({ fprintf(stderr, "Error: "); \
      fprintf(stderr, msg); \
      fclose(fd); \
      return error_code; })

// error_return with argument
#define error_return_warg(msg, error_code, arg) \
    ({ fprintf(stderr, "Error: "); \
      fprintf(stderr, msg, arg); \
      return error_code; })

#endif

```

exchange.h:

```

#ifndef __EXCHANGE__
#define __EXCHANGE__

#include <stdint.h>
#include <string.h>

#include "bmp.h"
#include "exceptions.h"
#include "etc.h"

void swap_rgb(RGB* a, RGB* b);

void rotate_clockwise(RGB*** arr, int64_t x0, int64_t y0,
                     int64_t x1, int64_t y1, uint32_t W, uint32_t
H);
// counterclockwise
void rotate_cclockwise(RGB*** arr, int64_t x0, int64_t y0,
                      int64_t x1, int64_t y1, uint32_t W, uint32_t
H);

void rotate_diagonals(RGB*** arr, int64_t x0, int64_t y0,
                     int64_t x1, int64_t y1, uint32_t W, uint32_t
H);

int32_t exchange(RGB ***arr, const BitmapInfoHeader *bmih,
                 int64_t *left_up, int64_t *right_down,
                 char *exchange_type);

```

```
#endif
```

freq_color.h:

```
#ifndef __FREQ_COLOR__  
#define __FREQ_COLOR__
```

```
#include <stdint.h>
```

```
#include "bmp.h"  
#include "dictionary.h"  
#include "exceptions.h"
```

```
int32_t freq_color(RGB*** arr, const BitmapInfoHeader *bmih, RGB  
color);
```

```
#endif
```

parse_funcs.h:

```
#ifndef __PARSE_FUNCS__  
#define __PARSE_FUNCS__
```

```
#include <stdio.h>  
#include <stdint.h>  
#include <stdlib.h>  
#include <regex.h>
```

```
#include "bmp.h"  
#include "exceptions.h"
```

```
int32_t parse_unsigned_char(const char*, uint8_t*);  
int32_t parse_coords(const char*, int64_t**);  
int32_t parse_posit_number(const char*, uint32_t*);  
int32_t parse_comps(const char*, RGB* color);
```

```
#endif
```

print_funcs.h:

```
#ifndef __PRINT_FUNCS__  
#define __PRINT_FUNCS__
```

```
#include <stdio.h>
```

```
void print_warn_msg(char* exe_name);  
void print_help(char* exe_name);
```

```
#endif
```

rgbfilter.h:

```
#ifndef __RGBFILTER__
#define __RGBFILTER__

#include <stdint.h>
#include <string.h>
#include <regex.h>

#include "bmp.h"
#include "exceptions.h"
#include "parse_funcs.h"

uint8_t identify(const char*);
int32_t rgbfilter(RGB**, const BitmapInfoHeader*, const char*,
const uint8_t);

#endif
```