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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

ТЕМА: РАБОТА С ИЗОБРАЖЕНИЯМИ

Студент гр. 3342 _____ Львов А.В.

Преподаватель _____ Глазунов С.А.

Санкт-Петербург

2024

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студент Львов А.В.

Группа 3342

Тема работы: Работа с изображениями

Исходные данные:

Вариант 5.3

Задание

Программа должна иметь следующую функции по обработке изображений:

- 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
- 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`)
- Поменять местами 4 куска 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`
- 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)

Содержание пояснительной записки:

Разделы пояснительный записки: «Содержание», «Введение», «Структуры», «Функции», «Тестирование», «Заключение», «Список использованных источников», «Приложение А. Примеры работы программы», «Приложение Б. Исходный код программы».

Дата выдачи задания: 18.03.2024

Дата сдачи реферата: 13.05.2024

Дата защиты реферата: 15.05.2024

Студент _____ Львов А.В.

Преподаватель _____ Глазунов С.А.

АННОТАЦИЯ

Курсовая работа представляет собой программу, реализующую CLI и обрабатывающую изображение формата BMP в соответствии с переданными пользователем в неё опциями (фильтр RGB-компонент, рисование квадрата, изменение 4 частей области, замена часто встречаемого цвета другим). Для выполнения этой задачи программа использует функции различных библиотек языка Си, в том числе “getopt.h” для реализации CLI. Выполнив необходимые преобразования создаётся файл с изменённым изображением.

SUMMARY

The course work is a program that implements the CLI and processes a BMP image in accordance with the options passed to it by the user (RGB component filter, drawing a square, changing 4 parts of the area, replacing a frequently encountered color with another). To accomplish this task, the program uses the functions of various C language libraries, including "getopt.h” for the CLI implementation. After completing the necessary transformations, a file with the modified image is created.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью данной работы является написание программы, осуществляющую обработку изображения в соответствии с опциями, введёнными пользователем. Для этого требуется:

- Создать функции, реализующие CLI.
- Создать функции, обрабатывающие изображение в соответствии с выбором пользователя.
- Написать Makefile, с помощью которого будет реализована сборка программы.

1. СТРУКТУРЫ

1.1. Структура BitmapFileHeader

Структура BitmapFileHeader состоит из таких полей как:

- unsigned short signature - поле заголовка, используемое для идентификации файла BMP и DIB, имеет шестнадцатеричное значение, равное BM в ASCII.
- unsigned int filesize - размер файла BMP в байтах.
- unsigned short reserved1 - зарезервировано; фактическое значение зависит от приложения, создающего изображение.
- unsigned short reserved2 - зарезервировано; фактическое значение зависит от приложения, создающего изображение.
- unsigned int pixelArrOffset - смещение, т. е. начальный адрес байта, в котором находятся данные изображения (массив пикселей).

1.2. Структура BitmapInfoHeader

Структура BitmapInfoHeader состоит из таких полей как:

- unsigned int headerSize – размер этого заголовка в байтах.
- unsigned int width – ширина изображения в пикселях.
- unsigned int height – длина изображения в пикселях.
- unsigned short planes – количество цветовых плоскостей.
- unsigned short bitsPerPixel – глубина цвета изображения.
- unsigned int compression – используемый метод сжатия.
- unsigned int imageSize – размер изображения.
- unsigned int xPixelsPerMeter – горизонтальное разрешение изображения.
- unsigned int yPixelsPerMeter – вертикальное разрешение изображения.

- unsigned int colorsInColorTable – количество цветов в цветовой палитре.
- unsigned int importantColorCount – количество используемых важных цветов.

1.3. Структура RGB

Структура RGB состоит из таких полей как:

- unsigned char b – синяя компонента цвета.
- unsigned char g – зелёная компонента цвета.
- unsigned char r – красная компонента цвета.

1.4. Структура RGBFitler

Структура RGBFilter состоит из таких полей как:

- enum RGB componentName – перечисление типа RGB, отвечающее за название компоненты.
- unsigned char componentValue – значение компоненты.

1.5. Структура Square

Структура Square состоит из таких полей как:

- int x, y – координаты левого верхнего угла.
- unsigned int sideSize, thickness – длина стороны, толщина линий соответсвенно.
- RGB color, fillColor – цвет, цвет заливки соответственно.
- unsigned char fill – поле, отвечающее за то, была ли введена опция заливки.

1.6. Структура Exchange

Структура Exchange состоит из таких полей как:

- enum ExchangeType extype – перечисление типа ExchangeType, отвечающее за тип изменения мест частей области.
- int lx, ly – координаты левого верхнего угла.
- int rx, ry – координаты правого нижнего угла.

1.7. Структура Tasks

Структура Tasks представляет собой проверки на введённые пользователем опции и состоит из таких полей как:

- unsigned char check_info.
- unsigned char check_rgbfilter.
- unsigned char check_component_name,
- unsigned char check_component_value,
- unsigned char check_input.
- unsigned char check_output.
- unsigned char check_square.
- unsigned char check_thickness.
- unsigned char check_side_size.
- unsigned char check_fill.
- unsigned char check_fill_color.
- unsigned char check_freq_color.
- unsigned char check_color.
- unsigned char check_exchange.
- unsigned char check_right_down.
- unsigned char check_exchange_type.
- unsigned char check_left_up.

2. ФУНКЦИИ

2.1. Функции, реализующие CLI

Функция handleInput (void handleInput(int argc, char * argv[])) принимает на вход количество и список аргументов, переданных в программу. Далее, с помощью getopt_long и switch осуществляется обработка опций командной строки и следующие действия с изображением: чтение файла – обработка изображения в соответствии с переданными аргументами – запись в новый файл.

Функция initTasks (void initTasks(Tasks * tasks)) инициализирует поля структуры Tasks.

Функция checkCoordinates (int checkCoordinates(char * coords)) проверяет, является ли переданная в функцию последовательность символов правильной записью координат.

Функция checkNum (int checkNum(char * string)) проверяет, является ли переданный массив символов числом.

Функция checkColor (int checkColor(char * color)) проверяет, является ли переданная в функцию последовательность символов правильной записью цвета.

2.2. Функции первого задания

Функция filter (RGB ** filter(RGB ** arr, unsigned int H, unsigned int W, enum RGB componentName, unsigned char componentValue)) принимает на вход массив пикселей, длину, высоту изображения, перечисление типа RGB, указывающее на то, какую компоненту цвета необходимо заменить и её новое значение. Пройдя циклом по массиву пикселей, меняет соответствующую компоненту каждого пикселя согласно переданным аргументам.

2.3 Функции второго задания

Функция initSquare (void initSquare(Square * sq)) инициализирует поля структуры Square.

Функция drawLine (RGB ** drawLine(RGB ** arr, unsigned int x1, unsigned int x2, unsigned int y1, unsigned int y2, RGB color)) принимает на вход массив пикселей, координаты начала и конца линии, цвет. Далее, так как требуется нарисовать квадрат, происходят проверки на то, является ли линия перпендикулярной относительно оси OУ или OX и в соответствии с переданными координатами рисуется.

Функция drawSquare (RGB ** drawSquare(RGB ** arr, unsigned int H, unsigned int W, Square sq)) принимает на вход массив пикселей, высоту, ширину изображения, структуру Square. В начале происходит проверка на возможность рисования квадрата. Затем, в цикле for происходят проверки на возможность отрисовки сторон квадрата и выход за границы изображения. Если линию нарисовать возможно, вызывается функция drawLine. Если был введён флаг fill, то после рисования квадрата происходит его заливка с помощью последовательного вызова функции drawLine.

2.4 Функции третьего задания

Функция initExchange (void initExchange(Exchange * ex)) инициализирует поля структуры Exchange.

Функция exchange (RGB ** exchange(RGB ** arr, unsigned int H, unsigned int W, Exchange ex)) принимает на вход массив пикселей, высоту, ширину изображения, структуру Exchange. Сначала происходят проверки на корректность переданных аргументов. Далее, с помощью switch, происходит определение типа перестановки (возможные значения - diagonals, clockwise, counterclockwise) и в соответствии с ним происходит перестановка пикселей в области местами.

2.5 Функции четвертого задания

Функция processFreqColor (RGB ** processFreqColor(BitmapInfoHeader * bmih, RGB ** arr, RGB newc)) получает на вход заголовок изображения, массив пикселей, новый цвет. Изначально формируется одномерный массив пикселей pixels, затем, с помощью функции qsort он сортируется и вызываются функции поиска наиболее часто встречаемого цвета (findFreqColor) и его замена (changeFreqColor).

Функция findFreqColor (RGB findFreqColor(RGB * sortedPixels, int sizePixels)) принимает на вход отсортированный одномерный массив пикселей, его размер. В цикле for происходит вычисление наиболее часто встречаемого цвета.

Функция changeFreqColor (RGB ** changeFreqColor(RGB ** arr, RGB freqc, RGB newc, unsigned int H, unsigned int W)) находит с помощью функции colorsEqual пиксели, чей цвет совпадает с наиболее часто встречаемым и меняет их на новый цвет.

Функция colorsEqual (int colorsEqual(RGB a, RGB b)) сравнивает два переданных цвета и возвращает 1, если они равны и 0 в противном случае.

Функция cmp (int cmp(const void * a, const void * b)) нужна для qsort. Она сравнивает два пикселя покомпонентно. Возвращает 0 если пиксели равны, 1, если значения компонент первого пикселя больше таковых у второго и -1 в противном случае.

2.6 Функции работы с файлами

Функция readBMP (RGB ** readBMP(char * file_name, BitmapFileHeader * bmfh, BitmapInfoHeader * bmih)) считывает переданный файл с структуры BitmapFileHeader и BitmapInfoHeader, а также в двумерный массив пикселей.

Функция checkBMP (int checkBMP(BitmapFileHeader * bmfh, BitmapInfoHeader * bmih)) проверяет соответствие ожидаемому формату файла.

Функция writeBMP (void writeBMP(char * filename, RGB ** arr, unsigned int H, unsigned int W, BitmapFileHeader * bmfh, BitmapInfoHeader * bmih)) записывает в новый файл все данные, полученные после обработки изображения.

2.7 Прочие функции

Функция printFileHeader (void printFileHeader(BitmapFileHeader header)) печатает всю информацию о заголовке файла.

Функция printInfoHeader (void printInfoHeader(BitmapInfoHeader header)) печатает всю информацию о заголовке изображения.

Функция help (void help()) выводит справку.

2.8 Функция main

Функция int main() вызывает функцию, реализующую CLI – handleInput.

Примеры работы программы см. в приложении А.

Разработанный программный код см. в приложении В.

3. СБОРКА ПРОГРАММЫ

Для сборки программы использовался Makefile, в котором компилируются все исходные файлы и линкуются. Программа собирается в файл “cw”.

Также есть цель clean, которая удаляет все объектные файлы.

ЗАКЛЮЧЕНИЕ

Курсовая работа включала в себя систематизацию знаний об изображениях и способах их обработки, изучение и применение функций стандартной библиотеки языка C.

В процессе работы были использованы структуры для хранения заголовков файлов, различных опций. Были изучены особенности языка, связанные с обработкой изображений и реализацией CLI. С помощью стандартной библиотеки были реализованы основные функции чтения и обработки файлов, представляющих собой BMP изображение.

В итоге готовый программный код выполняет все поставленные перед ним задачи, а именно, обрабатывает изображение, выводит сообщения о возникающих ошибках, задействует функции стандартной библиотеки языка C, использует структуры для хранения информации.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Курс “Программирование на Си. Практические задания. Второй семестр”. URL https://e.moevm.info/course/view.php?id=8

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

ПРИЛОЖЕНИЕ А
ПРИМЕРЫ РАБОТЫ ПРОГРАММЫ

ПРИМЕР 1 – вывод справки.

```
alex@rumeftw:~/pr-2024-3342/Lvov_Aleksandr_cw/src$ ./cw --help
Course work for option 5.3, created by Alexandr Lvov.
Available options:
'--help' ['-h'] - вывод справки.

--input ['-i'] - имя входного файла.

--output ['-o'] - имя выходного файла. Если флаг не использован, имя выходного файла - 'out.bmp'.

--info - вывод информации о файле.

'--rgbfilter' ['-f'] - этот инструмент позволяет для всего изображения либо установить в диапазоне от 0 до 255 значение заданной компоненты.
├── '--component_name' - какую компоненту требуется изменить. Возможные значения: red, green, blue.
└── '--component_value' - новое значение изменяемой компоненты. Возможные значения в диапазоне [0, 255].

'--square' ['-s'] - рисование квадрата.
├── '--left_up' - координата левого верхнего угла, значение задаётся в формате `left.up`, где left – координата по x, up – координата по y.
├── '--size_size' - размер стороны. Целое число, больше 0.
├── '--thickness' ['-t'] - толщина линий. Целое число, больше 0.
├── '--color' ['-c'] - цвет линий. Задаётся строкой `rrr.ggg.bbb`, где rrr/ggg/bbb – числа, задающие цветовую компоненту.
├── '--fill' - заливка. Работает как бинарное значение: флаг есть - true, в ином случае - false.
└─|── '--fill_color'- цвет заливки, если был введён флаг '--fill'. Принимает значения, аналогичные флагу '--color'.

'--exchange' ['-x'] - поменять местами 4 куска области. Выбранная пользователем прямоугольная область делится на 4 части и эти части меняются местами
├── '--left_up' - координата левого верхнего угла, значение задаётся в формате `left.up`, где left – координата по x, up – координата по y.
├── '--right_down' - координата правого нижнего угла, значение задаётся в формате `right.down`, где right – координата по x, down – координата по y.
└── '--exchange_type' - способ обмена частей. Возможные значения: `clockwise`, `counterclockwise`, `diagonals`.

'--freq_color' - найти самый часто встречаемый цвет и заменить его другим.
└── '--color' - цвет, на который необходимо заменить старый. Задаётся строкой `rrr.ggg.bbb`, где rrr/ggg/bbb – числа, задающие цветовую компоненту.
```

ПРИМЕР 2 – вывод информации о файле.

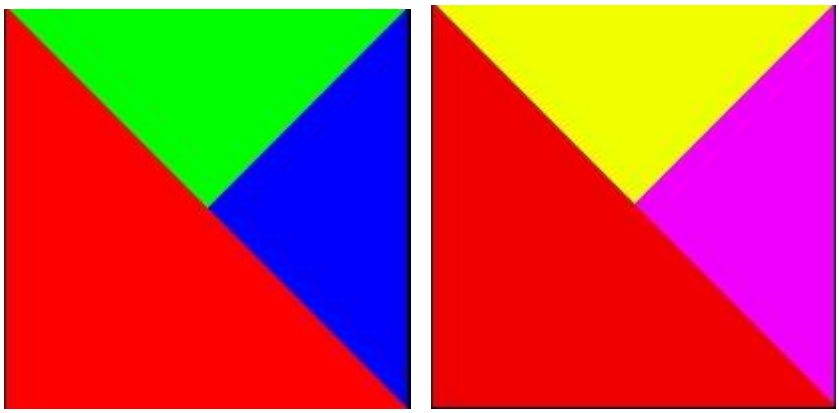
```
alex@rumeftw:~/pr-2024-3342/Lvov_Aleksandr_cw/src$ ./cw --info --input ./img/blackbuck.bmp
signature:      4d42 (19778)
filesize:       c0036 (786486)
reserved1:      0 (0)
reserved2:      0 (0)
pixelArrOffset: 36 (54)
headerSize:     28 (40)
width:  200 (512)
height:         200 (512)
planes:         1 (1)
bitsPerPixel:   18 (24)
compression:    0 (0)
imageSize:      c0000 (786432)
xPixelsPerMeter:        0 (0)
yPixelsPerMeter:        0 (0)
colorsInColorTable:     0 (0)
importantColorCount:    0 (0)
```

ПРИМЕР 3 – обработка случая несуществующего файла.

```
alex@rumeftw:~/pr-2024-3342/Lvov_Aleksandr_cw/src$ ./cw --info --input ./img/n.bmp
Error: file doesn't exist!
```

ПРИМЕР 4 – фильтр RGB компонент.

Параметры запуска: ./cw —rgbfilter —input ./img/bmp_24.bmp —component_value 240 —component_name red

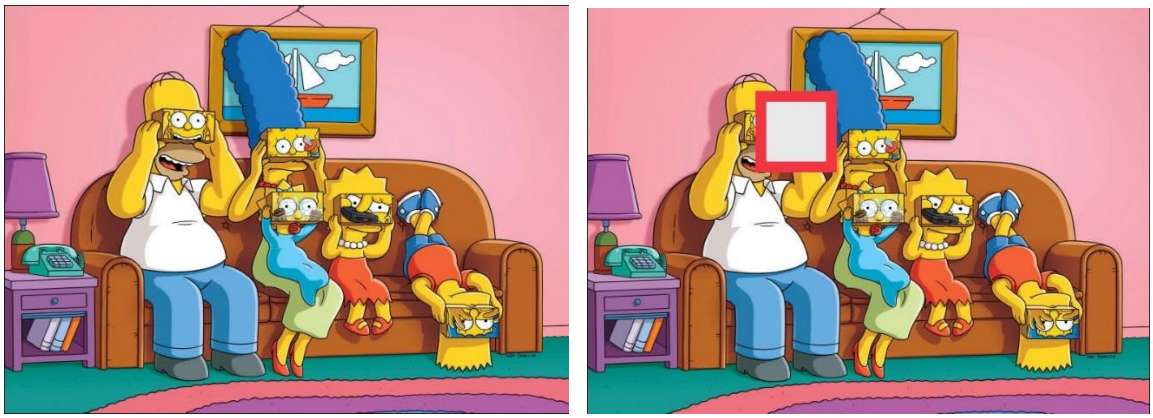


ПРИМЕР 5 – Обработка случая недостаточного количества опций для выполнения задания.

```
alex@rumeftw:~/pr-2024-3342/Lvov_Aleksandr_cw/src$ ./cw --rgbfilter --input ./img/bmp_24.bmp
Error: not enough options to use rgb-filter!
```

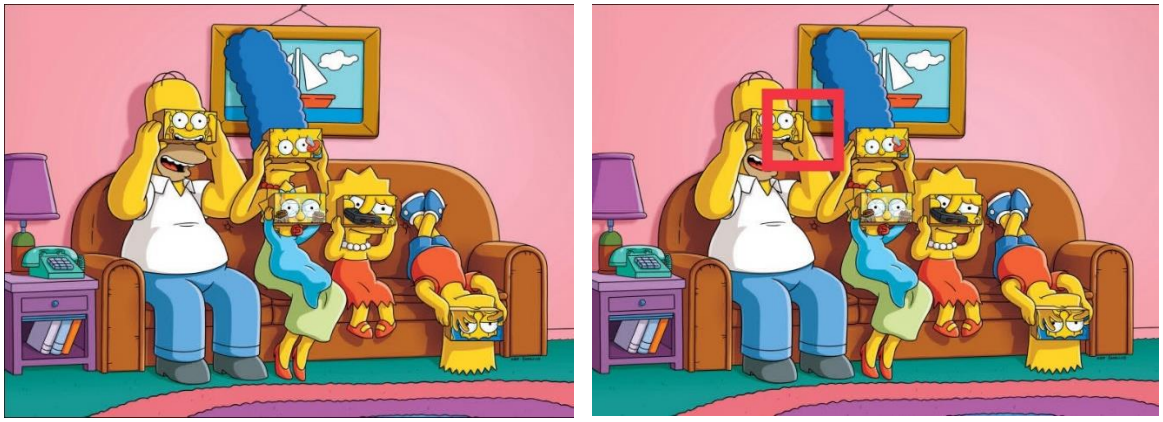
ПРИМЕР 6 – Рисование квадрата с заливкой.

Параметры запуска: ./cw —square —input ./img/simpsonsvr.bmp —left_up 240.123 —side_size 100 —thickness 15 —color 244.53.68 —fill —fill_color 233.233.233



ПРИМЕР 7 – Рисование квадрата без заливки.

Параметры запуска: ./cw —square —input ./img/simpsonsvr.bmp —left_up 240.123 —side_size 100 —thickness 15 —color 244.53.68



ПРИМЕР 8 – Обработка случая некорректных координат.

```
alex@rumeftw:~/pr-2024-3342/Lvov_Aleksandr_cw/src$ ./cw --square --input ./img/simpsonsvr.bmp --left_up 123123.123 --side_size 100 --thickness 15 --color 244.53.68
Error: cannot draw square - invalid x coordinate!
```

ПРИМЕР 9 – Перестановка 4 частей области местами.

Параметры запуска: ./cw —exchange —exchange_type diagonals —left_up 234.345 —right_down 123.121 —input ./img/b.bmp

ПРИМЕР 10 – Обработка случая некорректного типа перестановки.

```
alex@rumeftw:~/pr-2024-3342/Lvov_Aleksandr_cw/src$ ./cw --exchange --exchange_type diadonals --left_up 234.345 --right_down 123.121 --input ./img/b.bmp
Error: incorrect type of exchange!
```

ПРИМЕР 11 – Изменение наиболее часто встречаемого цвета.

Параметры запуска: ./cw —freq_color —color 255.0.0 —input ./img/blackbuck.bmp

ПРИЛОЖЕНИЕ В
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: bmp-handler.h

```
#ifndef CW_BMP_HANDLER_H
#define CW_BMP_HANDLER_H

#include "libs.h"
#include "structures.h"
#include "errors.h"

RGB **readBMP(char * file_name, BitmapFileHeader * bmfh, BitmapInfoHeader * bmih); // чтение BMP файла

int checkBMP(BitmapFileHeader * bmfh, BitmapInfoHeader * bmih); // проверка на корректный формат BMP

void printFileHeader(BitmapFileHeader header); // вывод информации о заголовке файла

void printInfoHeader(BitmapInfoHeader header); // вывод информации о заголовке изображения

void writeBMP(char * filename, RGB ** arr, unsigned int H, unsigned int W, BitmapFileHeader * bmfh, BitmapInfoHeader * bmih); // запись в BMP файл

void help(); // вывод справки

#endif //CW_BMP_HANDLER_H
```

Название файла: errors.h

```
#ifndef CW_ERRORS_H
#define CW_ERRORS_H

#define INVALID_OPTION_ERROR 40

#define NO_REQUIRED_OPTIONS_ERROR 41

#define INVALID_OPTION_ARG_ERROR 42

#define INCORRECT_BMP_FORMAT_ERROR 43

#define FILE_ERROR 44

#define MEMORY_ALLOCATION_ERROR 45

#endif //CW_ERRORS_H
```

Название файла: exchange.h

```
#ifndef CW_EXCHANGE_H
#define CW_EXCHANGE_H
```

```
#include "libs.h"
#include "structures.h"
#include "errors.h"

void initExchange(Exchange * ex); // инициализация полей структуры Exchange

RGB ** exchange(RGB ** arr, unsigned int H, unsigned int W, Exchange ex);

#endif //CW_EXCHANGE_H
```

Название файла: freq-color.h

```
#ifndef CW_FREQ_COLOR_H
#define CW_FREQ_COLOR_H

#include "structures.h"
#include "libs.h"
#include "errors.h"

RGB ** processFreqColor(BitmapInfoHeader * bmih, RGB ** arr, RGB newc); // выполнение действий с часто встречаемым цветом

RGB ** changeFreqColor(RGB ** arr, RGB freqc, RGB newc, unsigned int H, unsigned int W); // изменение цвета пикселей, если он совпадает с часто встречаемым

RGB findFreqColor(RGB * sortedPixels, int sizePixels); // нахождение самого часто встречаемого цвета

int colorsEqual(RGB a, RGB b); // проверка на то, являются ли цвета одинаковыми

int cmp(const void * a, const void * b); // функция-компаратор цветов для qsort

#endif //CW_FREQ_COLOR_H
```

Название файла: libs.h

```
#ifndef CW_LIBS_H
#define CW_LIBS_H

#include <stdlib.h>
#include <stdio.h>
#include <getopt.h>
#include <ctype.h>
#include <string.h>

#endif //CW_LIBS_H
```

Название файла: options-list.h

```
#ifndef CW_PROB_OPTS_H
#define CW_PROB_OPTS_H
```

```
const struct option longOptions[] = {
        { "help", no_argument, NULL, 'h' },
        { "output", required_argument, NULL, 'o' },
        { "info", no_argument, NULL, 400 },
        { "input", required_argument, NULL, 'i' },
        { "rgbfilter", no_argument, NULL, 'f' },
        { "component_name", required_argument, NULL, 100 },
        { "component_value", required_argument, NULL, 101 },
        { "square", no_argument, NULL, 's' },
        { "left_up", required_argument, NULL, 200 },
        { "side_size", required_argument, NULL, 201 },
        { "thickness", required_argument, NULL, 't' },
        { "color", required_argument, NULL, 'c' },
        { "fill", no_argument, NULL, 203 },
        { "fill_color", required_argument, NULL, 204 },
        { "exchange", no_argument, NULL, 'x' },
        { "right_down", required_argument, NULL, 300 },
        { "exchange_type", required_argument, NULL, 301 },
        { "freq_color", no_argument, NULL, 500 },
        { NULL, 0, NULL, 0}
};

char * shortOptions = "ho:i:fst:c:x";

#endif //CW_PROB_OPTS_H
```

Название файла: options.h

```
#ifndef CW_OPTIONS_H
#define CW_OPTIONS_H

#include "libs.h"
#include "bmp-handler.h"
#include "errors.h"
#include "exchange.h"
#include "freq-color.h"
#include "rgb-filter.h"
#include "square.h"

typedef struct {
    // info
    unsigned char check_info;

    // rgb-filter
    unsigned char check_rgbfilter;
    unsigned char check_component_name;
    unsigned char check_component_value;

    // input file
    unsigned char check_input;

    // output file
    unsigned char check_output;

    // square
    unsigned char check_square;
    unsigned char check_thickness;
```

```
        unsigned char check_side_size;
        unsigned char check_fill;
        unsigned char check_fill_color;

        // freq color
        unsigned char check_freq_color;

        // color
        unsigned char check_color;

        // exchange
        unsigned char check_exchange;
        unsigned char check_right_down;
        unsigned char check_exchange_type;

        // left_up for exchange or square
        unsigned char check_left_up;
    } Tasks;

    void initTasks(Tasks * tasks); // инициализация списка заданий

    int checkCoordinates(char * coords); // проверка на валидность
координат

    int checkNum(char * string); // проверка на то, является ли строка
числом

    int checkColor(char * color); // проверка на валидность цвета

    void handleInput(int argc, char * argv[]); // обработка опций (cli)

    #endif //CW_OPTIONS_H
```

Название файла: rgb-filter.h

```
    #ifndef CW_RGB_FILTER_H
    #define CW_RGB_FILTER_H

    #include "structures.h"

    RGB ** filter(RGB ** arr, unsigned int H, unsigned int W, enum RGB
componentName, unsigned char componentValue);

    #endif //CW_RGB_FILTER_H


    Название файла: square.h
    #ifndef CW_SQUARE_H
    #define CW_SQUARE_H

    #include "structures.h"
    #include "libs.h"
    #include "errors.h"

    void initSquare(Square * sq); // инициализация полей структуры
Square
```

```
    RGB ** drawLine(RGB ** arr, unsigned int x1, unsigned int x2,
unsigned int y1, unsigned int y2, RGB color); // рисование линии

    RGB ** drawSquare(RGB ** arr, unsigned int H, unsigned int W,
Square sq); // рисование квадрата

    #endif //CW_SQUARE_H
```

Название файла: structures.h

```
    #ifndef CW_STRUCTURES_H
    #define CW_STRUCTURES_H

    #pragma pack(push, 1)

    typedef struct {
        unsigned short signature;
        unsigned int filesize;
        unsigned short reserved1;
        unsigned short reserved2;
        unsigned int pixelArrOffset;
    } BitmapFileHeader;

    typedef struct {
        unsigned int headerSize;
        unsigned int width;
        unsigned int height;
        unsigned short planes;
        unsigned short bitsPerPixel;
        unsigned int compression;
        unsigned int imageSize;
        unsigned int xPixelsPerMeter;
        unsigned int yPixelsPerMeter;
        unsigned int colorsInColorTable;
        unsigned int importantColorCount;
    } BitmapInfoHeader;

    #pragma pack(pop)

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

    enum RGB {
        red,
        green,
        blue
    };

    typedef struct {
        enum RGB componentName;
        unsigned char componentValue;
    } RGBFilter;

    typedef struct {
```

```
    int x, y;
    unsigned int sideSize, thickness;
    RGB color, fillColor;
    unsigned char fill; // 1 if true else 0
} Square;

enum ExchangeType {
    clockwise,
    counterclockwise,
    diagonals,
    undefined
};

typedef struct {
    enum ExchangeType extype;
    int lx, ly; // left-up coordinates
    int rx, ry; // right-down coordinates
} Exchange;

#endif //CW_STRUCTURES_H
```

Название файла: options.c

```
#include "../../headers/options.h"
#include "../../headers/options-list.h"

#define MAX_SIZE 4096

void handleInput(int argc, char * argv[]) {
    char * input = (char *)calloc(MAX_SIZE, sizeof(char));
    if (input == NULL) {
        printf("Error: cannot allocate memory!\n");
        exit(MEMORY_ALLOCATION_ERROR);
    }

    char * output = (char *)calloc(MAX_SIZE, sizeof(char));
    if (output == NULL) {
        printf("Error: cannot allocate memory!\n");
        exit(MEMORY_ALLOCATION_ERROR);
    }

    opterr = 0;
    int option_index;
     int res = getopt_long(argc, argv, shortOptions, longOptions,
&option_index);

    int lx, ly;

    Tasks tasks;
    initTasks(&tasks);

    RGB color;
    RGBFilter rgbf;
    Square sq;
    Exchange ex;

    initSquare(&sq);
```

```
            initExchange(&ex);

            while (res != -1) {
                switch (res) {
                    case 'h': // --help
                        help();
                        exit(0);

                    case 'o': // --output
                        strcpy(output, optarg);
                        tasks.check_output++;
                        break;

                    case 'i': // --input
                        input = optarg;
                        tasks.check_input++;
                        break;

                    case 400: // --info
                        tasks.check_info++;
                        break;

                    // --rgb-filter
                    case 'f': // --rgbfilter
                        tasks.check_rgbfilter++;
                        break;

                    case 100: // --component_name
                        if (strcmp(optarg, "red") == 0) {
                            rgbf.componentName = red;
                        } else if (strcmp(optarg, "green") == 0) {
                            rgbf.componentName = green;
                        } else if (strcmp(optarg, "blue") == 0) {
                            rgbf.componentName = blue;
                        } else {
                            printf("Error: incorrect component name!\n");
                            exit(INVALID_OPTION_ARG_ERROR);
                        }
                        tasks.check_component_name++;
                        break;

                    case 101: // --component_value
                        if (checkNum(optarg)) {
                            int arg;
                            sscanf(optarg, "%d", &arg);
                            if (arg < 0 || arg > 255) {
                                printf("Error: component_value must be in
[0; 255]!\n");
                                exit(INVALID_OPTION_ARG_ERROR);
                            }
                            rgbf.componentValue = arg;
                        } else {
                                printf("Error: component value must be
integer!\n");
                            exit(INVALID_OPTION_ARG_ERROR);
                        }
                        tasks.check_component_value++;
```

```
                    break;

                case 'c': // --color
                    if (checkColor(optarg)) {
                           sscanf(optarg, "%hhu.%hhu.%hhu", &color.r,
&color.g, &color.b);
                    } else {
                        printf("Error: invalid color!\n");
                        exit(INVALID_OPTION_ARG_ERROR);
                    }

                    tasks.check_color++;

                    break;

                // square
                case 's': // --square
                    tasks.check_square++;
                    break;

                case 200: // --left_up
                    if (checkCoordinates(optarg)) {
                        sscanf(optarg, "%d.%d", &lx, &ly);
                    } else {
                           printf("Error: coordinates must match the
format <x.y> where x and y are integers!\n");
                        exit(INVALID_OPTION_ARG_ERROR);
                    }

                    tasks.check_left_up++;
                    break;

                case 201: // --side_size
                    if (checkNum(optarg)) {
                        sscanf(optarg, "%u", &sq.sideSize);
                    } else {
                           printf("Error: side size must be positive
integer!\n");
                        exit(INVALID_OPTION_ARG_ERROR);
                    }

                    tasks.check_side_size++;
                    break;

                case 't': // --thickness
                    if (checkNum(optarg)) {
                        sscanf(optarg, "%u", &sq.thickness);
                    }
                    else {
                           printf("Error: thickness must be positive
integer!\n");
                        exit(INVALID_OPTION_ARG_ERROR);
                    }

                    tasks.check_thickness++;
                    break;
```

```
                case 203: // --fill
                    tasks.check_fill++;
                    sq.fill = 1;
                    break;

                case 204: // --fill_color
                    if (checkColor(optarg)) {
                                    sscanf(optarg,  "%hhu.%hhu.%hhu",
&sq.fillColor.r, &sq.fillColor.g, &sq.fillColor.b);
                    } else {
                        printf("Error: invalid fill color!\n");
                        exit(INVALID_OPTION_ARG_ERROR);
                    }
                    break;

                // exchange
                case 'x': // --exchange
                    tasks.check_exchange++;
                    break;

                case 300: // --right_down
                    if (checkCoordinates(optarg)) {
                        sscanf(optarg, "%d.%d", &ex.rx, &ex.ry);
                    } else {
                            printf("Error: coordinates must match the
format <x.y> where x and y are integers!\n");
                        exit(INVALID_OPTION_ARG_ERROR);
                    }
                    break;

                case 301: // --exchange_type
                    if (strcmp(optarg, "clockwise") == 0) {
                        ex.extype = clockwise;
                     } else if (strcmp(optarg, "counterclockwise") ==
0) {
                        ex.extype = counterclockwise;
                    } else if (strcmp(optarg, "diagonals") == 0) {
                        ex.extype = diagonals;
                    } else {
                       printf("Error: incorrect type of exchange!\n");
                        exit(INVALID_OPTION_ARG_ERROR);
                    }
                    break;

                // freq-color
                case 500: // freq_color
                    tasks.check_freq_color++;
                    break;

                case '?':
                     printf("Error: invalid option!\nTo see the help,
use the option '--help ['-h']'\n");
                    exit(INVALID_OPTION_ERROR);

                default:
                    break;
            }
```

```
            res = getopt_long(argc, argv, shortOptions, longOptions,
&option_index);
        }

        if (tasks.check_input == 0) {
            input = argv[argc - 1];
        }

        if (tasks.check_output == 0) {
            output = "out.bmp";
        }

        if (strcmp(input, output) == 0) {
             printf("Error: names of input and output files must be
different!\n");
            exit(INVALID_OPTION_ARG_ERROR);
        }

        BitmapFileHeader bmfh;
        BitmapInfoHeader bmih;
        RGB ** rgb = readBMP(input, &bmfh, &bmih);

        if (!(checkBMP(&bmfh, &bmih))) {
            printf("Error: incorrect BMP format!\n");
            exit(INCORRECT_BMP_FORMAT_ERROR);
        }

        if (tasks.check_exchange) {
            ex.lx = lx;
            ex.ly = ly;
            if (ex.extype == undefined) {
                  printf("Error: you didn't enter '--exchange_type'
option!\n");
                exit(NO_REQUIRED_OPTIONS_ERROR);
            }
            rgb = exchange(rgb, bmih.height, bmih.width, ex);
        }

        if (tasks.check_square) {
            sq.x = lx;
            sq.y = ly;
            sq.color = color;
                 if  (tasks.check_color  &&  tasks.check_left_up  &&
tasks.check_thickness && tasks.check_side_size) {
                rgb = drawSquare(rgb, bmih.height, bmih.width, sq);
            } else {
                printf("Error: not enough options to draw square!\n");
                exit(NO_REQUIRED_OPTIONS_ERROR);
            }
        }

        if (tasks.check_rgbfilter) {
                            if     (tasks.check_component_name     &&
tasks.check_component_value) {
                       rgb  =  filter(rgb,  bmih.height,  bmih.width,
rgbf.componentName, rgbf.componentValue);
            }
```

```
            else {
                    printf("Error: not enough options to use rgb-
filter!\n");
                exit(NO_REQUIRED_OPTIONS_ERROR);
            }
        }

        if (tasks.check_freq_color) {
            if (tasks.check_color) {
                rgb = processFreqColor(&bmih, rgb, color);
            }
            else {
                 printf("Error: not enough options to replace freq.
color!\n");
                exit(NO_REQUIRED_OPTIONS_ERROR);
            }
        }

        writeBMP(output, rgb, bmih.height, bmih.width, &bmfh, &bmih);

        if (tasks.check_info) {
            printFileHeader(bmfh);
            printInfoHeader(bmih);
        }


        for (unsigned int i = 0; i < bmih.height; i++) {
            free(rgb[i]);
        }
        free(rgb);

        free(output);
    }

    void initTasks(Tasks * tasks) {
        tasks -> check_info = 0;

        tasks -> check_rgbfilter = 0;
        tasks -> check_component_name = 0;
        tasks -> check_component_value = 0;

        tasks -> check_input = 0;

        tasks -> check_output = 0;

        tasks -> check_square = 0;
        tasks -> check_thickness = 0;
        tasks -> check_side_size = 0;
        tasks -> check_fill = 0;
        tasks -> check_fill_color = 0;

        tasks -> check_freq_color = 0;

        tasks -> check_color = 0;

        tasks -> check_exchange = 0;
        tasks -> check_right_down = 0;
```

```
    tasks -> check_exchange_type = 0;

    tasks -> check_left_up = 0;
}

int checkCoordinates(char * coords) {
    char * cpy = malloc(strlen(coords) + 1);
    if (cpy == NULL) {
        printf("Error: cannot allocate memory!\n");
        exit(MEMORY_ALLOCATION_ERROR);
    }

    strcpy(cpy, coords);
    int c = 0; // количество координат

    char * tmp = strtok(cpy, ".");
    while (tmp != NULL) {
        if (!(checkNum(tmp))) {
            free(cpy);
            return 0;
        }
        c++;
        tmp = strtok(NULL, ".");
    }

    free(cpy);
    return c == 2;
}

int checkNum(char * string) {
    int start = (string[0] == '-');
    for (size_t i = start; i < strlen(string); i++) {
        if (!(isdigit(string[i]))) {
            return 0;
        }
    }
    return 1;
}

int checkColor(char * color) {
    char * cpy = malloc(strlen(color) + 1);
    if (cpy == NULL) {
        printf("Error: cannot allocate memory!\n");
        exit(MEMORY_ALLOCATION_ERROR);
    }

    strcpy(cpy, color);
    int c = 0; // количество компонент (r g b)

    char * tmp = strtok(cpy, ".");
    while (tmp != NULL) {
        if (!(checkNum(tmp))) {
            free(cpy);
            return 0;
        } else {
            int component = atoi(tmp);
            if (component < 0 || component > 255) {
```

```
                free(cpy);
                return 0;
            }
        }
        tmp = strtok(NULL, ".");
        c++;
    }
    free(cpy);
    return c == 3;
}
```

Название файла: exchange.c

```
#include "../../headers/exchange.h"

void initExchange(Exchange * ex) {
    ex -> extype = undefined;
    ex -> lx = 0;
    ex -> ly = 0;
    ex -> rx = 0;
    ex -> ry = 0;
}

RGB ** exchange(RGB ** arr, unsigned int H, unsigned int W, Exchange
ex) {
     if ((ex.lx < 0 && ex.rx < 0) || (ex.lx > (int)W && ex.rx >
(int)W) || (ex.ly > (int)H && ex.ry > (int)H) || (ex.ry < 0 && ex.ly <
0)) {
        printf("Error: invalid coordinates!\n");
        exit(INVALID_OPTION_ARG_ERROR);
    }

    // перевод координат y в индексы массива пикселей
    ex.ly = ((int)H) - ex.ly;
    ex.ry = ((int)H) - ex.ry;

    // выделение левого верхнего и правого нижнего углов
    if (ex.ly < ex.ry) {
        int tmp = ex.ly;
        ex.ly = ex.ry;
        ex.ry = tmp;
    }
    if (ex.lx > ex.rx) {
        int tmp = ex.lx;
        ex.lx = ex.rx;
        ex.rx = tmp;
    }

    // проверки на выход за границы изображения
    if (ex.lx < 0){
        ex.lx = 0;
    }
    if (ex.rx > (int)W) {
```

```
            ex.rx = W;
        }
        if (ex.ly > (int)H) {
            ex.ly = H;
        }
        if (ex.ry < 0) {
            ex.ry = 0;
        }

        if ((ex.rx - ex.lx) % 2) {
            ex.rx--;
        }
        if ((ex.ly - ex.ry) % 2) {
            ex.ly--;
        }

        switch (ex.extype) {
            case diagonals:
                for (int i = ex.ly; i > (ex.ly + ex.ry) / 2; i--) {
                    for (int j = ex.lx; j < (ex.rx + ex.lx) / 2; j++)
{
                        RGB tmp = arr[i][j];
                        arr[i][j] = arr[i - ((ex.ly - ex.ry) / 2)][j +
(ex.rx - ex.lx) / 2];
                         arr[i - ((ex.ly - ex.ry) / 2)][j + (ex.rx -
ex.lx) / 2] = tmp;
                    }
                }
                for (int i = (ex.ly + ex.ry) / 2; i > ex.ry; i--) {
                    for (int j = ex.lx; j < (ex.rx + ex.lx) / 2; j++)
{
                        RGB tmp = arr[i][j];
                        arr[i][j] = arr[i + ((ex.ly - ex.ry) / 2)][j +
(ex.rx - ex.lx) / 2];
                         arr[i + ((ex.ly - ex.ry) / 2)][j + (ex.rx -
ex.lx) / 2] = tmp;
                    }
                }
                break;
            case clockwise:
                for (int i = ex.ly; i > (ex.ly + ex.ry) / 2; i--) {
                    for (int j = ex.lx; j < (ex.rx + ex.lx) / 2; j++)
{
                        RGB tmp = arr[i][j];
                        arr[i][j] = arr[i][j + (ex.rx - ex.lx) / 2];
                        arr[i][j + (ex.rx - ex.lx) / 2] = tmp;
                    }
                }
                for (int i = (ex.ly + ex.ry) / 2; i > ex.ry; i--) {
                    for (int j = ex.lx; j < (ex.rx + ex.lx) / 2; j++)
{
                        RGB tmp = arr[i][j];
                        arr[i][j] = arr[i][j + (ex.rx - ex.lx) / 2];
                        arr[i][j + (ex.rx - ex.lx) / 2] = tmp;
                    }
                }
                for (int i = ex.ly; i > (ex.ly + ex.ry) / 2; i--) {
```

```
                    for (int j = ex.lx; j < (ex.rx + ex.lx) / 2; j++)
{
                        RGB tmp = arr[i][j];
                        arr[i][j] = arr[i - ((ex.ly - ex.ry) / 2)][j +
(ex.rx - ex.lx) / 2];
                         arr[i - ((ex.ly - ex.ry) / 2)][j + (ex.rx -
ex.lx) / 2] = tmp;
                    }
                }
                break;
            case counterclockwise:
                for (int i = ex.ly; i > (ex.ly + ex.ry) / 2; i--) {
                    for (int j = ex.lx; j < (ex.rx + ex.lx) / 2; j++)
{
                        RGB tmp = arr[i][j];
                        arr[i][j] = arr[i][j + (ex.rx - ex.lx) / 2];
                        arr[i][j + (ex.rx - ex.lx) / 2] = tmp;
                    }
                }
                for (int i = (ex.ly + ex.ry) / 2; i > ex.ry; i--) {
                    for (int j = ex.lx; j < (ex.rx + ex.lx) / 2; j++)
{
                        RGB tmp = arr[i][j];
                        arr[i][j] = arr[i][j + (ex.rx - ex.lx) / 2];
                        arr[i][j + (ex.rx - ex.lx) / 2] = tmp;
                    }
                }
                for (int i = (ex.ly + ex.ry) / 2; i > ex.ry; i--) {
                    for (int j = ex.lx; j < (ex.rx + ex.lx) / 2; j++)
{
                        RGB tmp = arr[i][j];
                        arr[i][j] = arr[i + ((ex.ly - ex.ry) / 2)][j +
(ex.rx - ex.lx) / 2];
                         arr[i + ((ex.ly - ex.ry) / 2)][j + (ex.rx -
ex.lx) / 2] = tmp;
                    }
                }
                break;
            case undefined:
                printf("Error: undefined type of exchange!\n");
                exit(INVALID_OPTION_ARG_ERROR);
        }

        return arr;
    }
```

Название файла: freq-color.c

```
    #include "../../headers/freq-color.h"

    RGB ** processFreqColor(BitmapInfoHeader * bmih, RGB ** arr, RGB
newc) {
        RGB * pixels =(RGB *)malloc(bmih -> height * bmih -> width *
sizeof(RGB)); // одномерный массив пикселей
        if (pixels == NULL) {
            printf("Error: cannot allocate memory!\n");
            exit(MEMORY_ALLOCATION_ERROR);
```

```
        }

        int sizep = 0; // размер одномерного массива пикселей

        for (unsigned int i = 0; i < bmih -> height; i++) {
            for (unsigned int j = 0; j < bmih -> width; j++) {
                pixels[sizep++] = arr[i][j];
            }
        }

         qsort(pixels, bmih -> height * bmih -> width, sizeof(RGB),
cmp);

        RGB freq = findFreqColor(pixels, sizep);
        RGB ** rgb = changeFreqColor(arr, freq, newc, bmih -> height,
bmih -> width);
        return rgb;
    }

    RGB ** changeFreqColor(RGB ** arr, RGB freqc, RGB newc, unsigned
int H, unsigned int W) {
        for (unsigned int i = 0; i < H; i++) {
            for (unsigned int j = 0; j < W; j++) {
                if (colorsEqual(arr[i][j], freqc)) {
                    arr[i][j] = newc;
                }
            }
        }
        free(pixels);
        return arr;
    }

    int colorsEqual(RGB a, RGB b) {
        return a.r == b.r && a.g == b.g && a.b == b.b;
    }

    RGB findFreqColor(RGB * sortedPixels, int sizePixels) {
        int max = 1; // максимальное количество вхождений одного цвета
        int curr = 1; // текущее количество вхождений цвета
        RGB maxc; // цвет с максимальным количеством вхождений

        for (int i = 1; i < sizePixels; i++) {
            if (colorsEqual(sortedPixels[i], sortedPixels[i - 1])) {
                curr++;
            }
            else {
                curr = 1;
            }
            if (curr > max) {
                max = curr;
                maxc = sortedPixels[i];
            }
        }

        return maxc;
    }
```

```
int cmp(const void * a, const void * b) {
    RGB f = *(RGB *)a;
    RGB s = *(RGB *)b;
    if (f.r > s.r) {
        return 1;
    }
    else if (f.r == s.r) {
        if (f.g > s.g) {
            return 1;
        }
        else if (f.g == s.g) {
            if (f.b > s.b) {
                return 1;
            }
            else if (f.b == s.b) {
                return 0;
            }
            else {
                return -1;
            }
        } else {
            return -1;
        }
    }
    else {
        return -1;
    }
}
```

Название файла: rgb-filter.c

```
#include "../../headers/rgb-filter.h"

RGB ** filter(RGB ** arr, unsigned int H, unsigned int W, enum RGB
componentName, unsigned char componentValue) {
    for (unsigned int i = 0; i < H; i++) {
        for (unsigned int j = 0; j < W; j++) {
            switch (componentName) {
                case red:
                    arr[i][j].r = componentValue;
                    break;
                case green:
                    arr[i][j].g = componentValue;
                    break;
                case blue:
                    arr[i][j].b = componentValue;
                    break;
            }
        }
    }
    return arr;
}
```

Название файла: square.c

```
#include "../../headers/square.h"
```

```
    void initSquare(Square * sq) {
        sq -> x = 0;
        sq -> y = 0;
        sq -> sideSize = 0;
        sq -> thickness = 0;
        RGB color = {0, 0, 0};
        sq -> color = color;
        RGB fillColor = {0, 0, 0};
        sq -> fillColor = fillColor;
        sq -> fill = 0;
    }

    RGB ** drawLine(RGB ** arr, unsigned int x1, unsigned int x2,
unsigned int y1, unsigned int y2, RGB color) {
        if (x1 == x2) {
            for (unsigned int col = y1; col < y2; col++) {
                arr[col][x1] = color;
            }
        } else if (y1 == y2) {
            for (unsigned int row = x1; row < x2; row++) {
                arr[y1][row] = color;
            }
        }
        return arr;
    }

    RGB ** drawSquare(RGB ** arr, unsigned int H, unsigned int W,
Square sq) {
        sq.y = H - sq.y;

        if (sq.x > (int)W || (int)(sq.x + sq.sideSize + sq.thickness /
2) <= 0) {
                printf("Error: cannot draw square - invalid x
coordinate!\n");
            exit(INVALID_OPTION_ARG_ERROR);
        } else if ((int)(sq.y + sq.thickness / 2) < 0) {
                printf("Error: cannot draw square - invalid y
coordinate!\n");
            exit(INVALID_OPTION_ARG_ERROR);
        }

        unsigned int x1, x2, y1, y2;
        sq.x += sq.thickness / 2;

        sq.y -= sq.thickness / 2;
        sq.sideSize -= sq.thickness + 1;

        for (unsigned int t = 0; t < sq.thickness; t++) {

            if ((int)(sq.y + t) < (int)H && (int)(sq.y + t) >= 0) {
                y1 = sq.y + t;
                if ((int)(sq.x - t) < 0) {
                    x1 = 0;
                } else {
                    x1 = sq.x - t;
                }
                if ((int)(sq.x + sq.sideSize + t) > (int)W) {
```

```
                    x2 = W;
                } else {
                    if ((int)(sq.x + sq.sideSize + t) < 0) {
                        continue;
                    }
                    x2 = sq.x + sq.sideSize + t;
                }
                drawLine(arr, x1, x2, y1, y1, sq.color);
            }

            if ((int)(sq.y - sq.sideSize - t) >= 0 && sq.y - sq.sideSize
- t < H) {
                y1 = sq.y - sq.sideSize - t;
                if ((int)(sq.x - t) < 0) {
                    x1 = 0;
                } else {
                    x1 = sq.x - t;
                }
                if (sq.x + sq.sideSize + t > W) {
                    x2 = W;
                } else {
                    x2 = sq.x + sq.sideSize + t;
                }
                drawLine(arr, x1, x2, y1, y1, sq.color);
            }

            if ((int)(sq.x - t) >= 0) {
                x1 = sq.x - t;
                if (sq.y + t > H) {
                    y2 = H;
                } else {
                    y2 = sq.y + t;
                }
                if ((int)(sq.y - sq.sideSize - t) < 0) {
                    y1 = 0;
                } else {
                    y1 = sq.y - sq.sideSize - t;
                }
                drawLine(arr, x1, x1, y1, y2, sq.color);

            }

            if (sq.x + sq.sideSize + t <= W) {
                x1 = sq.x + sq.sideSize + t;
                if ((int)(sq.y + t + 1) > (int)H) {
                    y2 = H;
                } else if ((int)(sq.y + t + 1) >= 0){
                    y2 = sq.y + t + 1;
                }
                else {
                    continue;
                }
                if ((int)(sq.y - sq.sideSize - t) < 0) {
                    y1 = 0;
                } else {
                    y1 = sq.y - sq.sideSize - t;
                }
```

```
                drawLine(arr, x1, x1, y1, y2, sq.color);

            }
        }

        if (sq.fill) {
            if ((int)(sq.x + sq.sideSize) < 0) {
                printf("Square wasn't filled!\n");
                return arr;
            }
            sq.sideSize += sq.thickness - 1;
             for (y1 = sq.y - 1; (int)y1 > (int)(sq.y - sq.sideSize +
sq.thickness - 1); y1--) {
                if (y1 > (H - 1)) {
                    continue;
                }
                if (sq.x < 0) {
                    x1 = 0;
                } else {
                    x1 = sq.x + 1;
                }
                if ((int)(sq.x + sq.sideSize - 1) > (int)W) {
                    x2 = W;
                } else {
                    x2 = sq.x + sq.sideSize - sq.thickness + 1;
                }
                drawLine(arr, x1, x2, y1,y1, sq.fillColor);
            }
        }

        return arr;
    }
```

Название файла: bmp-handler.c

```
    #include "../headers/bmp-handler.h"

    RGB ** readBMP(char * file_name, BitmapFileHeader * bmfh,
BitmapInfoHeader * bmih){
        FILE *f = fopen(file_name, "rb");
        if (f == NULL) {
            printf("Error: file doesn't exist!\n");
            exit(FILE_ERROR);
        }
        fread(bmfh, 1, sizeof(BitmapFileHeader), f);
        fread(bmih, 1, sizeof(BitmapInfoHeader), f);

        fseek(f, bmfh -> pixelArrOffset, SEEK_SET);
        unsigned int H = bmih -> height;
        unsigned int W = bmih -> width;

        RGB **arr = malloc(H * sizeof(RGB*));
        if (arr == NULL) {
            printf("Error: cannot allocate memory!\n");
            exit(MEMORY_ALLOCATION_ERROR);
        }
```

```
        for(unsigned int i = 0; i < H; i++){
            arr[i] = malloc(W * sizeof(RGB) + (4 - W * sizeof(RGB) %
4) % 4);
            if (arr[i] == NULL) {
                printf("Error: cannot allocate memory!\n");
                exit(MEMORY_ALLOCATION_ERROR);
            }
            fread(arr[i], 1, W * sizeof(RGB) + (4 - W * sizeof(RGB) %
4) % 4,f);
        }
        fclose(f);
        return arr;
    }

    int checkBMP(BitmapFileHeader * bmfh, BitmapInfoHeader * bmih) {
        return bmfh -> signature == 0x4D42 && bmih -> compression == 0
&& bmih -> bitsPerPixel == 24;
    }

    void writeBMP(char * filename, RGB ** arr, unsigned int H, unsigned
int W, BitmapFileHeader * bmfh, BitmapInfoHeader * bmih) {
        FILE * file = fopen(filename, "wb");
        if (file == NULL) {
            printf("Error: cannot open the file [%s]!\n", filename);
            exit(FILE_ERROR);
        }
        fwrite(bmfh, 1, sizeof(BitmapFileHeader), file);
        fwrite(bmih, 1, sizeof(BitmapInfoHeader), file);
        fseek(file, bmfh->pixelArrOffset, SEEK_SET);
        for (unsigned int i = 0; i < H; i++) {
           fwrite(arr[i], 1,   W * sizeof(RGB) + (4 - W * sizeof(RGB) %
4) % 4, file);
        }
        fclose(file);
    }

    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 help() {
          printf("Course work for option 5.3, created by Alexandr
Lvov.\n");
         printf("Available options:\n");
         printf("'--help' ['-h'] - вывод справки.\n\n");
         printf("--input ['-i'] - имя входного файла.\n\n");
          printf("--output ['-o'] - имя выходного файла. Если флаг не
использован, имя выходного файла - 'out.bmp'.\n\n");
         printf("--info - вывод информации о файле.\n\n");
          printf("'--rgbfilter' ['-f'] - этот инструмент позволяет для
всего изображения либо установить в диапазоне от 0 до 255 значение
заданной компоненты.\n");
          printf("├── '--component_name' - какую компоненту требуется
изменить. Возможные значения: red, green, blue.\n");
          printf("└── '--component_value' - новое значение изменяемой
компоненты. Возможные значения в диапазоне [0, 255].\n\n");
         printf("'--square' ['-s'] - рисование квадрата.\n");
          printf("├── '--left_up' - координата левого верхнего угла,
значение задаётся в формате `left.up`, где left – координата по x, up –
координата по y.\n");
         printf("├── '--size_size' - размер стороны. Целое число, больше
0.\n");
         printf("├── '--thickness' ['-t'] - толщина линий. Целое число,
больше 0.\n");
          printf("├── '--color' ['-c'] - цвет линий. Задаётся строкой
`rrr.ggg.bbb`,   где   rrr/ggg/bbb   –   числа,   задающие   цветовую
компоненту.\n");
         printf("├── '--fill' - заливка. Работает как бинарное значение:
флаг есть - true, в ином случае - false.\n");
          printf("└──|── '--fill_color'- цвет заливки, если был введён
флаг '--fill'. Принимает значения, аналогичные флагу '--color'.\n\n");
         printf("'--exchange' ['-x'] - поменять местами 4 куска области.
Выбранная пользователем прямоугольная область делится на 4 части и эти
части меняются местами\n");
          printf("├── '--left_up' - координата левого верхнего угла,
значение задаётся в формате `left.up`, где left – координата по x, up –
координата по y.\n");
         printf("├── '--right_down' - координата правого нижнего угла,
значение задаётся в формате `right.down`, где right – координата по x,
down – координата по y.\n");
```

```
        printf("└── '--exchange_type' - способ обмена частей. Возможные
значения: `clockwise`, `counterclockwise`, `diagonals`.\n\n");
        printf("'--freq_color' - найти самый часто встречаемый цвет и
заменить его другим.\n");
        printf("└── '--color' - цвет, на который необходимо заменить
старый. Задаётся строкой `rrr.ggg.bbb`, где rrr/ggg/bbb – числа,
задающие цветовую компоненту.\n");
    }
```

Название файла: main.c

```
#include "../headers/options.h"

int main(int argc, char * argv[]) {
    handleInput(argc, argv);
}
```

Название файла: Makefile

```
CC=gcc
CFLAGS=-I$(HEADERS) -std=c99 -Wall -Wextra

BIN=./bin/
HEADERS=../headers

SRC = $(wildcard ./*.c)
SRC_CLI = $(wildcard ./cli/*.c)
SRC_TASKS = $(wildcard ./tasks/*.c)

OBJ =  $(patsubst ./%.c, $(BIN)%.o, $(SRC))
OBJ_CLI =  $(patsubst ./cli/%.c, $(BIN)%.o, $(SRC_CLI))
OBJ_TASKS =  $(patsubst ./tasks/%.c, $(BIN)%.o, $(SRC_TASKS))

OBJECTS = $(OBJ) $(OBJ_CLI) $(OBJ_TASKS)

.PHONY: all clean

all: $(BIN) $(OBJECTS)
    $(CC) $(CFLAGS) $(OBJECTS) -o cw

$(BIN):
    @ mkdir $(BIN)

${BIN}%.o: %.c
    @ $(CC) $(CFLAGS) -c $< -o $@

${BIN}%.o: ./cli/%.c
    @ $(CC) $(CFLAGS) -c $< -o $@

${BIN}%.o: ./tasks/%.c
    @ $(CC) $(CFLAGS) -c $< -o $@

clean:
    rm -rf $(BIN) cw *.o
```