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

ОТЧЕТ

по курсовой работе

по дисциплине «Программирование»

ТЕМА: РАЗРАБОТКА ПРОГРАММЫ НА ЯЗЫКЕ С ДЛЯ ОБРАБОТКИ РАСТРОВОГО ИЗОБРАЖЕНИЯ BMP

Студент гр. 4344 _____ Бровин В.С.

Преподаватель _____ Иванов Д. В.

Санкт-Петербург

2025

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студент Бровин В.С.

Группа 4344

Тема работы: **Обработка изображений формата bmp**

Исходные данные:

Необходима программа на языке C, удовлетворяющая заданию по обработке изображения формата bmp.

Содержание пояснительной записки:

1. Аннотация
2. Оглавление
3. Введение
4. Заключение
5. Список использованных источников

Дата выдачи задания: 7.03.2025

Дата сдачи реферата:

Дата защиты реферата: 23.05.2025

Студент _____ Бровин В.С.

Преподаватель _____ Иванов Д.В.

СОДЕРЖАНИЕ

АННОТАЦИЯ

Программа предназначена для обработки изображений BMP (24 бита, без сжатия) через командную строку с возможностью реализации графического интерфейса. Функция --rgbfilter позволяет изменить одну из цветовых компонент (красную, зелёную или синюю) на значение от 0 до 255. Команда --square рисует квадрат по заданным координатам левого верхнего угла, длине стороны, толщине линии и цвету. При наличии флага --fill квадрат заливается указанным цветом. Функция --exchange разбивает прямоугольную область изображения на четыре части и переставляет их в порядке, заданном параметром: по часовой стрелке, против часовой или по диагонали. Команда --freq_color находит самый часто встречающийся цвет и заменяет его на пользовательский.

SUMMARY

The program is designed to process BMP image files (24-bit, uncompressed) via a command-line interface, with optional GUI support. It allows modifying a specific RGB component using --rgbfilter, where the component (red, green, or blue) is set to a value between 0 and 255. The --square function draws a square based on the top-left corner coordinates, side length, line thickness, and line color. If the --fill flag is present, the square is filled with the specified fill color. The --exchange option divides a rectangular area defined by two corner points into four regions and rearranges them according to the specified type: clockwise, counterclockwise, or diagonally. The --freq_color command finds the most frequently occurring color in the image and replaces it with a user-defined color.

ВВЕДЕНИЕ

Целью данной курсовой работы является изучение формата хранения растровых изображений BMP и разработка консольного приложения для их обработки. Для достижения поставленной цели необходимо решить следующие задачи:

1. Изучить структуру формата BMP, в частности 24-битных несжатых изображений, включая особенности хранения пикселей, заголовков и выравнивания строк.
2. Ознакомиться с методами проверки корректности входного BMP-файла и обработкой возможных ошибок формата.
3. Разработать программу, реализующую заданный набор функций по обработке изображений через интерфейс командной строки.
4. Провести тестирование программы на различных изображениях с разными параметрами входных данных.
5. Сделать выводы о пригодности выбранных методов обработки графической информации и об эффективности их программной реализации.

Задание

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 следующую 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 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`
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

СТРУКТУРА И ФУНКЦИИ ПРОЕКТА

Структуры:

Структура BMPHeader содержит информацию, которая описывает общий формат BMP-файла и располагается в его начале. Она включает следующие поля:

- bfType – идентификатор формата файла. Для BMP этот параметр содержит значение, соответствующее символам "BM" в ASCII (0x4D42). Это позволяет определить, что файл действительно является BMP-изображением.
- bfSize – общий размер BMP-файла в байтах, включая все заголовки и пиксельные данные.
- bfReserved1 и bfReserved2 – зарезервированные поля, которые должны быть равны нулю. Они предназначены для будущего использования и обычно не содержат полезной информации.
- bfOffBits – смещение от начала файла до начала блока с пиксельными данными. То есть, это количество байт, которое нужно пропустить, чтобы дойти до изображения.

Структура BMPInfoHeader представляет собой заголовок информации о изображении (официально называется BITMAPINFOHEADER) и содержит подробные сведения о характеристиках самого изображения в BMP-файле. Вот описание каждого поля:

- biSize – размер этой структуры в байтах. Для стандартного BITMAPINFOHEADER это значение должно быть 40.
- biWidth – ширина изображения в пикселях.
- biHeight – высота изображения в пикселях. Если значение положительное, то изображение хранится снизу вверх (стандарт BMP). Если отрицательное — сверху вниз.

- biPlanes – количество цветовых плоскостей. Для BMP-файлов это всегда 1.
- biBitCount – количество бит на пиксель (глубина цвета). Наиболее распространённые значения: 1, 4, 8, 24, 32. Например, 24 означает, что каждый пиксель кодируется тремя байтами (по 8 бит на красный, зелёный и синий каналы).
- biCompression – тип сжатия. Значение 0 означает отсутствие сжатия (BI_RGB). Другие значения используются реже и означают различные схемы сжатия.
- biSizeImage – размер данных изображения в байтах. Может быть нулевым для несжатых изображений, но часто всё же указывается реальный размер.
- biXPelsPerMeter и biYPelsPerMeter – горизонтальное и вертикальное разрешение изображения, выраженное в пикселях на метр. Эти значения используются, например, при печати изображения.
- biClrUsed – количество цветов в палитре (если используется). Для полноцветных изображений (24 или 32 бита) обычно равно 0.
- biClrImportant – количество "важных" цветов. Обычно также устанавливается в 0, что означает, что все цвета считаются важными.

Структура BMP содержит в себе:

- BMPHeader – указатель на структуру BMPHeader.
- BMPInfoHeader – указатель на структуру BMPInfoHeader.
- Pixels – указатель на uint_8, хранит в себе массив пикселей.

Струтрура FilterParams служит входным параметром функции rgbfilter, содержит в себе:

- component_name – массив символов для хранения названия компоненты.

- component_value – целое число для хранения значения компоненты.

Структура RGB служит для хранения 3 компонент пикселя: красной, синей, зеленой. Содержит в себе:

- red – целое число для хранения красной компоненты.
- blue – целое число для хранения синей компоненты.
- green – целое число для хранения зеленой компоненты.

Структура Point служит для хранения координат пикселя по вертикали(y) и по горизонтали(x). Содержит в себе:

- x – целое число для хранения значения по горизонтали.
- y– целое число для хранения значения по вертикали.

Структура SquareParams служит входным параметром для функции draw_square. Содержит в себе:

- left_up – структура Point для хранения левой верхней координаты.
- side_size – целое число для хранения длины стороны.
- thickness – целое число для хранения ширины линий.
- fill – тип bool, хранит информацию о том, что нужно ли закрашивать внутри или нет.
- color – структура RGB, хранит в себе цвет, которым будут нарисованы стороны квадрата.
- fill_color - структура RGB, хранит в себе цвет, которым квадрат будет закрашен внутри.

Структура ExchangeParams служит входным параметром для функции exchange. Содержит в себе:

- left_up – структура Point для хранения левой верхней координаты.

- right_down – структура Point для хранения правой нижней координаты.
- mode – целое число для хранения типа перестановки.

Функции:

int main(int agrc, char * agrs[]) – принимает аргументы командной строки: имя входного и выходного BMP-файлов и параметры обработки изображения. Внутри создаются структуры FilterParams, SquareParams, ExchangeParams для хранения параметров, а также массивы символов для имён файлов. Далее с помощью getopt_long происходит разбор аргументов — в зависимости от флагов задаются режим работы (фильтрация цвета, рисование квадрата, замена частого цвета, перестановка фрагментов, вывод информации), координаты, цвет, толщина, тип перестановки и другие значения. Выполняется проверка корректности значений, контроль на наличие только одного действия и несовпадение имён входного и выходного файлов. Вызывается нужная функция (filter, draw_square, freq_color, exchange, info). В случае ошибок (например, неправильные параметры, не удалось открыть файл, одинаковые имена файлов) выводится сообщение, и программа завершает выполнение с кодом ошибки. В конце освобождается выделенная память и сохраняется результат в выходной файл.

BMP *read_bmp(const char *filename) – функция считывания файла формата BMP. На вход принимает имя файла, после чего происходит открытие файла. Далее создаются и считываются две структуры: BMPHeader, BMPInfoHeader. Файл проверятся на формат BMP и цвет пикселя, после чего считывается массив пикселей. Если во время функции не было ошибок, то она вернет указатель заполненную структуру BMP, иначе NULL и выведет сообщение об ошибке.

bool save_bmp(const char *filename, BMP * bmp) – сохраняет изображение в формате BMP. Она принимает имя файла и указатель на структуру BMP, содержащую заголовки и пиксельные данные. Сначала функция пытается

открыть файл для записи в бинарном режиме. Если открыть не удаётся, выводится сообщение об ошибке, и функция возвращает false. При успешном открытии в файл записываются заголовки BMP: сначала основной (BMPHeader), затем информационный (BMPInfoHeader). После этого указатель файла сдвигается на позицию начала пиксельных данных, определённую полем bfOffBits, и происходит запись самих пикселей, объём которых задаётся полем biSizeImage. По завершении записи файл закрывается, и функция возвращает true, если всё прошло успешно. Таким образом, save_bmp обеспечивает корректное сохранение структуры BMP в файл.

void help() – предназначена для вывода справочной информации о работе программы в консоль. Она не принимает аргументов и вызывается пользователем при необходимости ознакомиться с доступными функциями и их параметрами. Внутри функции с помощью последовательных вызовов printf выводится краткое описание программы, а также подробные инструкции по использованию всех поддерживаемых команд.

void info(BMP *bmp) – предназначена для вывода на экран содержимого структуры BMPInfoHeader, которая хранит основную информацию об изображении BMP. Она принимает указатель на структуру BMP, содержащую вложенный указатель на BMPInfoHeader, и с помощью функции printf поочерёдно отображает все поля этой структуры.

bool ckeck_correct_component_name(const char * component_name) – предназначена для проверки корректности введённого пользователем названия цветовой компоненты RGB. Она принимает строку component_name, содержащую имя компоненты, и сравнивает её с допустимыми значениями: "red", "green" и "blue". Если введённое имя совпадает с одним из этих значений, функция возвращает true, что означает корректный ввод. В противном случае она выводит сообщение об ошибке в поток стандартных ошибок (с помощью fprintf) и возвращает false. Эта функция используется для валидации пользовательского ввода в команде --rgbfilter и помогает предотвратить

выполнение программы с некорректными аргументами. Таким образом, ckeck_correct_component_name обеспечивает надёжную фильтрацию недопустимых значений компонент и повышает устойчивость программы.

bool check_correct_component_value(int component_value) - предназначена для проверки корректности введённого значения цветовой компоненты RGB. Она принимает целочисленное значение component_value и проверяет, находится ли оно в допустимом диапазоне от 0 до 255 включительно. Если значение попадает в указанные границы, функция возвращает true, что означает, что параметр введён правильно. В противном случае выводится сообщение об ошибке в поток стандартных ошибок с помощью fprintf, и функция возвращает false. Данная проверка необходима при использовании фильтра RGB-компоненты (--rgbfilter), где значения цветов должны строго соответствовать формату BMP.

bool is_integer(const char * str) – предназначена для проверки, является ли переданная строка корректным десятичным целым числом. Она принимает строку str и использует функцию strtol для преобразования строки в число типа long. При этом указатель endptr указывает на первую часть строки, не распознанную как число. Если вся строка успешно преобразована, *endptr будет равен '\0', и функция вернёт true, что означает, что строка действительно является целым числом. Если же в строке присутствуют недопустимые символы, *endptr укажет на них, и функция вернёт false.

bool to_coordinate(Point * point, char *params) – используется для преобразования строки в координаты и записи их в структуру Point. Она принимает два аргумента: указатель на структуру Point, в которую будут сохранены значения, и строку params, содержащую координаты в формате x.y, где x и y — неотрицательные целые числа. Сначала функция разбивает строку с помощью strtok, используя точку (".") в качестве разделителя. Первая часть сохраняется как x_str, вторая — как y_str. Далее с помощью функции is_integer проверяется, являются ли обе части допустимыми целыми числами. Также

проверяется, что значения x и y неотрицательны. Если все условия выполняются, строковые значения преобразуются в целые числа с помощью atoi, записываются в поля x и y структуры Point, и функция возвращает true, что означает успешный разбор координат. В противном случае выводится сообщение об ошибке в поток стандартных ошибок с помощью fprintf, и функция возвращает false.

bool to_side_size(int * size, const char * side_size) – предназначена для преобразования строки в целочисленное значение параметра side_size, который используется для задания размера стороны квадрата. Она принимает указатель на переменную типа int, в которую будет записан результат, и строку side_size, содержащую значение в текстовом виде. Сначала функция проверяет, является ли строка целым числом с помощью is_integer, а затем убеждается, что число неотрицательное. Если обе проверки проходят успешно, строка преобразуется в число с помощью atoi, и результат сохраняется по указанному указателю size. После этого функция возвращает true, сигнализируя об успешном разборе параметра. Если строка не соответствует формату или значение отрицательное, выводится сообщение об ошибке в поток стандартных ошибок с помощью fprintf, и функция возвращает false.

bool to_thickness(int *thickness, const char *thickness_str) – предназначена для преобразования строки в целочисленное значение параметра thickness, отвечающего за толщину линии. Она принимает указатель на переменную типа int, куда будет записано значение толщины, и строку thickness_str, содержащую число в текстовом формате. Функция сначала проверяет, что строка представляет собой целое число с помощью is_integer, а также что это число строго больше нуля. Если обе проверки успешны, строка преобразуется в целое число функцией atoi, и результат записывается по адресу, на который указывает thickness. После этого функция возвращает true, подтверждая успешный разбор. Если входная строка не является допустимым

положительным целым числом, выводится сообщение об ошибке в стандартный поток ошибок с помощью fprintf, и функция возвращает false.

bool to_color(RGB * color, char *color_str) – предназначена для преобразования строки с цветом в структуру RGB, содержащую три компоненты цвета: красную, зелёную и синюю. Она принимает указатель на структуру RGB, в которую будут записаны значения, и строку color_str, содержащую цвет в формате rrr.ggg.bbb, где rrr, ggg и bbb — целые числа от 0 до 255. Внутри функция разбивает строку на три части с помощью strtok, используя точку в качестве разделителя, выделяя компоненты red_str, green_str и blue_str. Затем проверяется, что каждая часть является корректным целым числом с помощью is_integer. Если эта проверка проходит, строки преобразуются в числа и записываются в соответствующие поля структуры color. Далее функция дополнительно проверяет, что значения каждой компоненты находятся в допустимом диапазоне от 0 до 255, вызывая check_correct_component_value. Если все условия выполнены, функция возвращает true, сигнализируя об успешном разборе цвета. Если какая-либо проверка не пройдена, выводится сообщение об ошибке с помощью fprintf, и функция возвращает false.

bool to_exchange_type(int * exchange_type, const char *exchange_type_str) - предназначена для преобразования строки, задающей тип обмена (exchange_type_str), в целочисленное значение, которое записывается по адресу, указанному указателем exchange_type. Входными параметрами служат указатель на переменную типа int и строка с названием типа обмена. Функция сравнивает входную строку с тремя заранее определёнными константами, обозначающими допустимые варианты обмена: CLOCKWISE_EXCHANGE_TYPE, DIAGONALS_EXCHANGE_TYPE и COUNTERCLOCKWISE_EXCHANCE_TYPE. Если строка совпадает с первой константой, переменной exchange_type присваивается значение 1; если со второй — 2; если с третьей — 3. В каждом из этих случаев функция возвращает

true, подтверждая успешный разбор параметра. Если входная строка не совпадает ни с одной из допустимых констант, функция выводит сообщение об ошибке в поток стандартных ошибок с помощью fprintf, указывающее на некорректный тип обмена, и возвращает false.

void rgbfilter(FilterParams * filter_params, BMP * bmp) – предназначена для изменения заданной цветовой компоненты (красной, зелёной или синей) у каждого пикселя изображения BMP на указанное значение. На вход она принимает указатель на структуру FilterParams. Определяется индекс нужной цветовой компоненты (0 — синий, 1 — зелёный, 2 — красный) с помощью вспомогательной функции get_component_index. Далее происходит двойной цикл по всем пикселям изображения: внешний цикл перебирает строки по вертикали, а внутренний — столбцы по горизонтали. Выбранной компоненте текущего пикселя присваивается новое значение из параметров фильтра.

void draw_square(SquareParams * square_params, BMP * bmp) – рисует квадрат на BMP-изображении, принимает два параметра: указатель на структуру SquareParams и указатель на структуру BMP. Корректирует координаты верхнего левого угла квадрата так, чтобы линия не выходила за границы изображения. Далее происходит обход пикселей в заданной области с помощью вложенных циклов по вертикали и горизонтали. Функция проверяет, находится ли текущий пиксель в области границы или внутри квадрата, и в зависимости от этого либо пропускает пиксель, либо окрашивает его цветом линии или цветом заливки, если включена заливка.

void freq_color(RGB * color, BMP * bmp) – принимает указатель на структуру RGB, задающую новый цвет, и указатель на структуру BMP. Она находит самый часто встречающийся цвет в изображении и заменяет его на заданный цвет. Для этого сначала выделяется трёхмерный массив colors размером $256 \times 256 \times 256$, где каждый индекс соответствует значению компонентов синего, зелёного и красного цвета, инициализируется нулями. Затем функция проходит по всем пикселям изображения, подсчитывая

количество каждого цвета, увеличивая соответствующий элемент массива colors. После подсчёта определяется цвет с максимальной частотой max_color. Далее происходит второй проход по изображению, в котором все пиксели с этим цветом заменяются на указанный новый цвет. В конце выделенная память под массив colors освобождается.

void swap_fragment(FragmentParams * first_fragment, FragmentParams * second_fragment, BMP * bmp) – принимает два указателя на структуры FragmentParams, которые задают координаты двух областей изображения, и указатель на структуру BMP. Она меняет местами содержимое этих двух областей. Функция проходит по пикселям обеих областей одновременно, используя вложенные циклы, и для каждого пикселя обеих областей выполняет обмен значениями цвета. Для временного хранения цвета первого пикселя выделяется память под массив из трёх байт, куда копируются его значения. После копирования цвета второго пикселя в первый, значения из временного буфера записываются во второй пиксель. В конце временная память освобождается.

void exchange(ExchangeParams *exchange_params, BMP *bmp) – принимает указатель на структуру ExchangeParams, а также указатель на структуру BMP. Она корректирует координаты правого нижнего угла области, чтобы они не выходили за пределы изображения, и при необходимости уменьшает ширину и высоту области на единицу, если они нечётные, чтобы обеспечить равное деление на четыре части. Затем вычисляется ширина и высота области, после чего создаются четыре структуры FragmentParams. В зависимости от значения параметра mode эти части переставляются между собой с помощью функции swap_fragment. При mode равном 1 происходит перестановка по часовой стрелке. При mode равном 2 происходит обмен по диагонали. При mode равном 3 выполняется перестановка против часовой стрелки. Если mode не соответствует этим значениям, перестановок не происходит.

ТЕСТИРОВАНИЕ

Тестирование проводилось на одном и том же файле (рисунок 1). Результаты тестирования представлены в таблице 1.

Рисунок 1 - Изображение для теста

№	Входные данные	Выходные данные	Комментарии
1	--info	biSize: 40 biWidth: 2895 biHeight: 1628 biPlanes: 1 biBitCount: 24 biCompression: 0 biSizeImage: 14144064 biXPelsPerMeter: 2834 biYPelsPerMeter: 2834 biClrUsed: 0 biClrImportant: 0	Выводит информацию о соответствующем изображении
2	--rgbfilter --component_name red --component_value 255		Задает красной компоненте значение 255
3	--rgbfilter --component_name rad --component_value 255	Unknown component_name: rad Possible options: red, blue, green.	Если неверно введено название компоненты, завершается с ошибкой.

4	--rgbfilter --component_name red --component_value 256	Uncorrect component value: 256 Possible values are from 0 to 255.	Если неверно введено значение компоненты, завершается с ошибкой.
5	--square --left_up 500.500 --side_size 600 --color 255.255.255 --thickness 100		Рисует квадрат, начиная с заданной координаты, с заданной длиной сторон и шириной линий.
6	--square --left_up 500.500 --side_size 600 --color 255.255.255 --thickness 100 --fill --fill_color 255.0.0		То же самое, что и в прошлом тесте, только дополнительно закрашивает внутри.
7	--square --left_up 500.500 --side_size 2000 --color 255.255.255 --thickness --fill --fill_color 255.0.0		Если квадрат выходит за границы, то программа завершается без ошибок.
8	--freq_color --color 0.255.0		Заменяет самый часто встречаемый цвет на зеленый.
9	--exchange --left_up 100.100 --right_down 1500.1500 --exchange_type diagonals		Разбил заданную область на 4 равные части и обменял их по диагонали.
10	--exchange --left_up 100.100 --right_down 3000.3000 --exchange_type diagonals		Если область выходит за границы, то программа завершается без ошибок.
11	--exchange --left_up 100.100 --right_down	Uncorrect exchange_type: hello	Если неверно введен параметр exchange_type,

	3000.3000 --exchange_type hello	Possible options: clockwise, diagonals.	программа завершает с ошибкой.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы была разработана программа для обработки изображений в формате BMP с использованием интерфейса командной строки. Программа успешно реализует все поставленные задачи: проверку корректности BMP-файлов, рисование квадрата, нахождение и замена часто встречаемого цвета, изменение компоненты цвета и перестановка областей.

Разработка велась с соблюдением принципов модульности: функции разделены по логическим модулям, что облегчает сопровождение и дальнейшее расширение проекта.

Сборка программы осуществляется с помощью Makefile, что обеспечивает удобство компиляции и сборки проекта. В процессе выполнения работы были изучены особенности формата BMP, принципы работы с двоичными файлами, алгоритмы обработки изображений и построения простых графических фигур. Работа с консольным интерфейсом позволила закрепить навыки разработки CLI-приложений и обработки аргументов командной строки.

С разработанным программным кодом можно ознакомиться в приложении А.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Б. В. Керниган, Д. М. Ритчи. Язык программирования Си. М.: Диалектика, 2020. 288 с.

2. Справочник по библиотекам языка Си. URL: https://cplusplus.com/reference/

3. Базовые сведения к выполнению курсовой работы по дисциплине «Программирование». Второй семестр. (Авторы: М. М. Заславский, А. А. Лисс, А. В. Гаврилов, С. А. Глазунов, Я. С. Государкин, С. А. Тиняков, В. П. Голубева, К. В. Чайка, В. Е. Допира.)

ПРИЛОЖЕНИЕ А.
ИСХОДНЫЙ КОД ПРОГРАММЫ

bmp.c

Исходный код bmp.c

```
#include "bmp.h"

BMP *read_bmp(const char *filename)
{
    FILE * file = fopen(filename, "rb");

    if(!file)
    {
        fprintf(stderr, ERROR_OPEN_FILE, filename);
        return NULL;
    }

    BMPHeader * bmpHeader = malloc(sizeof(BMPHeader));
    BMPInfoHeader * bmpInfoHeader = malloc(sizeof(BMPInfoHeader));

    fread(bmpHeader, 1,sizeof(BMPHeader),file);
    fread(bmpInfoHeader, 1,sizeof(BMPInfoHeader), file);

    if(bmpHeader->bfType != BMP_TYPE)
    {
        fprintf(stderr, ERROR_IS_NOT_BMP, filename);
        free(bmpHeader);
        free(bmpInfoHeader);
        return NULL;
    }
    else if(bmpInfoHeader->biBitCount != 24)
    {
        fprintf(stderr, ERROR_BI_BIT_COUNT);
        free(bmpHeader);
        free(bmpInfoHeader);
        return NULL;
    }

    BMP * bmp = malloc(sizeof(BMP));
    bmp->BMPHeader = bmpHeader;
    bmp->BMPInfoHeader = bmpInfoHeader;

    if(bmp->BMPInfoHeader->biSizeImage == 0)
    {
        int width = bmp->BMPInfoHeader->biWidth;
        int height = abs(bmp->BMPInfoHeader->biHeight);
        int rowSize = ((width * 3 + 3) / 4) * 4;
        bmp->BMPInfoHeader->biSizeImage = rowSize * height;
    }
```

```
    bmp->pixels = malloc(bmp->BMPInfoHeader->biSizeImage);
    fseek(file, bmp->BMPHeader->bfOffBits, SEEK_SET);
    fread(bmp->pixels,1, bmp->BMPInfoHeader->biSizeImage, file);

    fclose(file);

    return bmp;
}

bool save_bmp(const char *filename, BMP * bmp)
{
    FILE * file = fopen(filename, "wb");
    if(!file)
    {
        fprintf(stderr, ERROR_OPEN_FILE, filename);
        return false;
    }

    fwrite(bmp->BMPHeader, sizeof(BMPHeader), 1, file);
    fwrite(bmp->BMPInfoHeader, sizeof(BMPInfoHeader), 1, file);
    fseek(file, bmp->BMPHeader->bfOffBits, SEEK_SET);
    fwrite(bmp->pixels, 1, bmp->BMPInfoHeader->biSizeImage, file);

    fclose(file);

    return true;
}

int get_component_index(const char * component_name)
{
    if(!strcmp(component_name, "blue"))
    {
        return 0;
    }
    else if(!strcmp(component_name, "green"))
    {
        return 1;
    }
    return 2;
}

void filter(FilterParams * filter_params, BMP * bmp)
{
    int rowSize = ((bmp->BMPInfoHeader->biWidth * 3 + 3 ) /4) * 4;
    int component_index = get_component_index(filter_params-
>component_name);

    for(int y = 0; y < bmp->BMPInfoHeader->biHeight; y++)
    {
        for(int x = 0; x < bmp->BMPInfoHeader->biWidth; x++)
```

```
        {
            uint8_t * pixel = &bmp->pixels[(bmp->BMPInfoHeader-
>biHeight - y -1) * rowSize + x * 3];
            pixel[component_index] = filter_params->component_value;
        }
    }
}

void draw_square(SquareParams * square_params, BMP * bmp)
{
    int rowSize = ((bmp->BMPInfoHeader->biWidth * 3 + 3) / 4) * 4;

    if(square_params->left_up.x - square_params->thickness / 2 < 0)
    {
        square_params->left_up.x = 0;
    }
    else
    {
        square_params->left_up.x -= square_params->thickness / 2;
    }

    if(square_params->left_up.y - square_params->thickness / 2 < 0)
    {
        square_params->left_up.y = 0;
    }
    else
    {
        square_params->left_up.y -= square_params->thickness / 2;
    }

    for(int y = square_params->left_up.y; (y <= square_params-
>left_up.y + square_params->side_size + square_params->thickness / 2)
&& y < bmp->BMPInfoHeader->biHeight;y++)
    {
        for(int x = square_params->left_up.x; (x <= square_params-
>left_up.x + square_params->side_size + square_params->thickness / 2)
&& x < bmp->BMPInfoHeader->biWidth; x++)
        {
            uint8_t * pixel = &bmp->pixels[(bmp->BMPInfoHeader-
>biHeight - y -1) * rowSize + x * 3];

            if(square_params->fill == false && !(x < square_params-
>left_up.x + square_params->thickness) && !(x > square_params-
>left_up.x + square_params->side_size - square_params->thickness)
                && !(y < square_params->left_up.y + square_params-
>thickness) && !(y > square_params->left_up.y + square_params-
>side_size - square_params->thickness))
                continue;
```

```
            else if(square_params->fill == true && !(x <
square_params->left_up.x + square_params->thickness) && !(x >
square_params->left_up.x + square_params->side_size - square_params-
>thickness)
                && !(y < square_params->left_up.y + square_params-
>thickness) && !(y > square_params->left_up.y + square_params-
>side_size - square_params->thickness))
                {
                    pixel[0] = square_params->fill_color.blue;
                    pixel[1] = square_params->fill_color.green;
                    pixel[2] = square_params->fill_color.red;
                }
            else{
                pixel[0] = square_params->color.blue;
                pixel[1] = square_params->color.green;
                pixel[2] = square_params->color.red;
            }
        }
    }
}

void freq_color(RGB * color, BMP * bmp)
{
    int *** colors = calloc(256, sizeof(int **));
    for(int i = 0; i < 256;i++)
    {
        colors[i] = calloc(256, sizeof(int *));
        for(int j = 0;j < 256;j++)
        {
            colors[i][j] = calloc(256, sizeof(int));
        }
    }

    int rowSize = ((bmp->BMPInfoHeader->biWidth * 3 + 3) / 4) * 4;

    for(int y = 0; y < bmp->BMPInfoHeader->biHeight; y++)
    {
        for(int x = 0; x < bmp->BMPInfoHeader->biWidth;x++)
        {
            uint8_t  * pixel = &bmp->pixels[y * rowSize + x * 3];
            colors[pixel[0]][pixel[1]][pixel[2]]++;
        }
    }

    RGB max_color = {0, 0,0};
    int max_count = 0;

    for(int i = 0; i < 256; i++)
    {
        for(int j = 0; j < 256; j++)
```

```
        {
            for(int k = 0; k < 256; k++)
            {
                if(colors[i][j][k] > max_count)
                {
                    max_color.red = k;
                    max_color.green = j;
                    max_color.blue = i;
                    max_count = colors[i][j][k];
                }
            }
        }
    }

    for(int y = 0; y < bmp->BMPInfoHeader->biHeight; y++)
    {
        for(int x = 0; x < bmp->BMPInfoHeader->biWidth; x++)
        {
            uint8_t * pixel = &bmp->pixels[y * rowSize + x * 3];
            if(pixel[0] == max_color.blue & pixel[1] ==
max_color.green & pixel[2] == max_color.red)
            {
                pixel[0] = color->blue;
                pixel[1] = color->green;
                pixel[2] = color->red;
            }
        }
    }

    for (int i = 0; i < 256; i++) {
        for (int j = 0; j < 256; j++) {
            free(colors[i][j]);
        }
        free(colors[i]);
    }
    free(colors);
}

void swap_fragment(FragmentParams * first_fragment, FragmentParams *
second_fragment, BMP * bmp)
{
    int rowSizeBmp = ((bmp->BMPInfoHeader->biWidth * 3 + 3) / 4) * 4;

    for(int y = first_fragment->left_up.y, j = second_fragment-
>left_up.y; y <= first_fragment->right_down.y && j <= second_fragment-
>right_down.y; y++, j++)
    {
        for(int x = first_fragment->left_up.x, i = second_fragment-
>left_up.x; x <= first_fragment->right_down.x && i <= second_fragment-
>right_down.x; x++, i++)
```

```
        {
            uint8_t * pixel_first = &bmp->pixels[rowSizeBmp * (bmp-
>BMPInfoHeader->biHeight - y -1) + x * 3];
            uint8_t * pixel_second = &bmp->pixels[rowSizeBmp * (bmp-
>BMPInfoHeader->biHeight - j - 1) + i * 3];
            uint8_t * tmp = malloc(sizeof(3));
            tmp[0] = pixel_first[0];
            tmp[1] = pixel_first[1];
            tmp[2] = pixel_first[2];

            pixel_first[0] = pixel_second[0];
            pixel_first[1] = pixel_second[1];
            pixel_first[2] = pixel_second[2];

            pixel_second[0] = tmp[0];
            pixel_second[1] = tmp[1];
            pixel_second[2] = tmp[2];

            free(tmp);
        }
    }
}

bool exchange(ExchangeParams *exchange_params, BMP *bmp)
{

    if(exchange_params->right_down.y > bmp->BMPInfoHeader->biHeight)
    {
        exchange_params->right_down.y = bmp->BMPInfoHeader->biHeight;
    }

    if(exchange_params->right_down.x > bmp->BMPInfoHeader->biWidth)
    {
        exchange_params->right_down.x = bmp->BMPInfoHeader->biWidth;
    }

    if((exchange_params->right_down.x - exchange_params->left_up.x) %
2 != 0)
        exchange_params->right_down.x--;
    if((exchange_params->right_down.y - exchange_params->left_up.y) %
2 != 0)
        exchange_params->right_down.y--;

    int width = (exchange_params->right_down.x - exchange_params-
>left_up.x);
    int height = (exchange_params->right_down.y - exchange_params-
>left_up.y);
```

```
    FragmentParams first = {{exchange_params->left_up.x,
exchange_params->left_up.y}, {exchange_params->left_up.x + width / 2 -
1,  exchange_params->left_up.y + height / 2 - 1}};
    FragmentParams second = {{exchange_params->left_up.x + width / 2,
exchange_params->left_up.y}, {exchange_params->right_down.x,
exchange_params->left_up.y + height / 2 - 1}};
    FragmentParams third = {{exchange_params->left_up.x,
exchange_params->left_up.y + height / 2}, {exchange_params->left_up.x
+ width / 2 - 1, exchange_params->right_down.y - 1}};
    FragmentParams fourth = {{exchange_params->left_up.x + width / 2,
exchange_params->left_up.y + height / 2}, {exchange_params-
>right_down.x, exchange_params->right_down.y - 1}};


    switch (exchange_params->mode)
    {
    case 1:
        swap_fragment(&first, &second, bmp);
        swap_fragment(&third, &fourth, bmp);
        swap_fragment(&first, &fourth, bmp);
        break;
    case 2:
        swap_fragment(&first, &fourth, bmp);
        swap_fragment(&second, &third, bmp);
        break;
    case 3:
        swap_fragment(&first, &third, bmp);
        swap_fragment(&first, &second, bmp);
        swap_fragment(&second, &fourth, bmp);
        break;

    default:
        break;
    }

}

bool ckeck_correct_component_name(const char *component_name)
{
    if(strcmp(component_name, "red") == 0 || strcmp(component_name,
"blue") == 0 || strcmp(component_name, "green") == 0)
        return true;
    fprintf(stderr, ERROR_UNKNOWN_COMPONENT_NAME, component_name);
    return false;
}

bool check_correct_component_value(int component_value)
{
    if(component_value >= 0 && component_value <= 255)
        return true;
```

```
    fprintf(stderr, ERROR_UNCORRECT_COMPONENT_VALUE, component_value);
    return false;
}

bool is_integer(const char * str)
{
    char * endptr;
    strtol(str, &endptr, 10);
    return *endptr == '\0';
}

bool to_coordinate(Point * point, char *params)
{
    char * x_str = strtok(params, ".");
    char * y_str = strtok(NULL, ".");
    if(is_integer(x_str) & is_integer(y_str) & atoi(x_str) >= 0 &
atoi(y_str) >= 0)
    {
        point->x = atoi(x_str);
        point->y = atoi(y_str);
        return true;
    }
    else
    {
        fprintf(stderr, ERROR_UNCORRECT_COORDINATES, x_str, y_str);
        return false;
    }
}

bool to_side_size(int * size, const char * side_size)
{
    if(is_integer(side_size) & atoi(side_size) >= 0)
    {
        *size = atoi(side_size);
        return true;
    }
    fprintf(stderr, ERROR_UNCORRECT_SIDE_SIZE, side_size);
    return false;
}

bool to_thickness(int *thickness, const char *thickness_str)
{
    if(is_integer(thickness_str) & atoi(thickness_str) > 0)
    {
        *thickness = atoi(thickness_str);
        return true;
    }
    fprintf(stderr, ERROR_UNCORRECT_THICKNESS, thickness_str);
    return false;
}
```

```
bool to_color(RGB * color, char *color_str)
{
    char * red_str = strtok(color_str, ".");
    char * green_str = strtok(NULL, ".");
    char * blue_str = strtok(NULL, ".");
    if(is_integer(red_str) & is_integer(green_str) &
is_integer(blue_str))
    {
        color->red = atoi(red_str);
        color->green = atoi(green_str);
        color->blue = atoi(blue_str);
        if(check_correct_component_value(color->red) &
check_correct_component_value(color->green) &
check_correct_component_value(color->blue))
        {
            return true;
        }
        return false;
    }
    fprintf(stderr, ERROR_UNCORRECT_COLOR, color_str);
    return false;
}

bool to_exchange_type(int * exchange_type, const char
*exchange_type_str)
{
    if(strcmp(exchange_type_str, CLOCKWISE_EXCHANGE_TYPE) == 0)
    {
        *exchange_type = 1;
        return true;
    }
    else if(strcmp(exchange_type_str, DIAGONALS_EXCHANGE_TYPE) == 0)
    {
        *exchange_type = 2;
        return true;
    }
    else if(strcmp(exchange_type_str, COUNTERCLOCKWISE_EXCHANCE_TYPE)
== 0)
    {
        *exchange_type = 3;
        return true;
    }
    fprintf(stderr, ERROR_UNCORRECT_EXCHANGE_TYPE, exchange_type_str);
    return false;
}

void info(BMP *bmp)
{
```

```
    printf("biSize: %u\n", bmp->BMPInfoHeader->biSize);
    printf("biWidth: %d\n", bmp->BMPInfoHeader->biWidth);
    printf("biHeight: %d\n", bmp->BMPInfoHeader->biHeight);
    printf("biPlanes: %u\n", bmp->BMPInfoHeader->biPlanes);
    printf("biBitCount: %u\n", bmp->BMPInfoHeader->biBitCount);
    printf("biCompression: %u\n", bmp->BMPInfoHeader->biCompression);
    printf("biSizeImage: %u\n", bmp->BMPInfoHeader->biSizeImage);
    printf("biXPelsPerMeter: %u\n", bmp->BMPInfoHeader-
>biXPelsPerMeter);
    printf("biYPelsPerMeter: %u\n", bmp->BMPInfoHeader-
>biYPelsPerMeter);
    printf("biClrUsed: %u\n", bmp->BMPInfoHeader->biClrUsed);
    printf("biClrImportant: %u\n", bmp->BMPInfoHeader-
>biClrImportant);
}


void compare(BMP * first, BMP * second)
{
    int rowSizeBmp = ((first->BMPInfoHeader->biWidth * 3 + 3) / 4) *
4;
    for(int y = 0; y < first->BMPInfoHeader->biHeight; y++)
    {
        for(int x = 0; x < first->BMPInfoHeader->biWidth; x++)
        {
            uint8_t * pixel_first = &first->pixels[rowSizeBmp *
(first->BMPInfoHeader->biHeight - y -1) + x * 3];
            uint8_t * pixel_second = &second->pixels[rowSizeBmp *
(second->BMPInfoHeader->biHeight - y - 1) + x * 3];
            if(pixel_first[0] != pixel_second[0] || pixel_first[1] !=
pixel_second[1] || pixel_first[2] != pixel_second[2])
            {
                printf("x, y: %d, %d\n", x, y);
            }
        }
    }
}

void help()
{
    FILE * file = fopen(HELP_FILE_NAME, "r");

    if(!file)
    {
        fprintf(stderr, ERROR_OPEN_FILE, HELP_FILE_NAME);
        return;
    }

    int ch;
```

```
    while((ch = getc(file)) != EOF)
    {
        putchar(ch);
    }

    fclose(file);
}

void draw_line(RGB color, Point p0, Point p1, BMP * bmp)
{
    int rowSize = ((bmp->BMPInfoHeader->biWidth * 3 + 3) / 4) * 4;

    int x0 = p0.x, x1 = p1.x;
    int y0 = p0.y, y1 = p1.y;

    int dx = abs(x1 - x0);
    int dy = -abs(y1 - y0);

    int sx = (x0 < x1) ? 1 : -1;
    int sy = (y0 < y1) ? 1 : -1;

    int err = dx + dy;

    while (1) {
        if (x0 >= 0 && x0 < bmp->BMPInfoHeader->biWidth && y0 >= 0 &&
y0 < bmp->BMPInfoHeader->biHeight) {
            int offset = (bmp->BMPInfoHeader->biHeight - y0 - 1) *
rowSize + x0 * 3;
            uint8_t * pixel = &bmp->pixels[offset];
            pixel[0] = color.blue;
            pixel[1] = color.green;
            pixel[2] = color.red;
        }

        if (x0 == x1 && y0 == y1) break;

        int e2 = 2 * err;
        if (e2 >= dy) {
            err += dy;
            x0 += sx;
        }
        if (e2 <= dx) {
            err += dx;
            y0 += sy;
        }
    }
}
```

bmp.o

Исходный код bmp.o

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>
#include <stdbool.h>
#pragma pack(push, 1)

typedef struct {
    uint16_t bfType;
    uint32_t bfSize;
    uint16_t bfReserved1;
    uint16_t bfReserved2;
    uint32_t bfOffBits;
} BMPHeader;

typedef struct {
    uint32_t biSize;
    uint32_t biWidth;
    uint32_t biHeight;
    uint16_t biPlanes;
    uint16_t biBitCount;
    uint32_t biCompression;
    uint32_t biSizeImage;
    uint32_t biXPelsPerMeter;
    uint32_t biYPelsPerMeter;
    uint32_t biClrUsed;
    uint32_t biClrImportant;
} BMPInfoHeader;

#pragma pack(pop)

typedef struct{
    BMPHeader * BMPHeader;
    BMPInfoHeader * BMPInfoHeader;
    uint8_t * pixels;
}BMP;

typedef struct{
    char component_name[5];
    int component_value;
}FilterParams;

typedef struct
{
    int red, green, blue;
}RGB;

typedef struct
```

```
{
    int x, y;
}Point;

typedef struct {
    Point left_up;
    int side_size;
    int thickness;
    bool fill;
    RGB color;
    RGB fill_color;
}SquareParams;

typedef struct
{
    Point left_up, right_down;
    int mode;
}ExchangeParams;

typedef struct
{
    Point left_up, right_down;
}FragmentParams;

typedef struct
{
    Point left_up, right_down;
    RGB color;
}ReplaceParams;

typedef struct
{
    int width;
    RGB color;
}FrameParams;
```

```
#define ERROR_OPEN_FILE "Couldn't open %s.\n"
#define ERROR_IS_NOT_BMP "%s format is not BMP.\n"
#define ERROR_BI_BIT_COUNT "BiBitCount isn't 24.\n"
#define ERROR_UNKNOWN_COMPONENT_NAME "Unknown component_name:
%s\nPossible options: red, blue, green.\n"
#define ERROR_UNCORRECT_COMPONENT_VALUE "Uncorrect component value:
%d\nPossible values are from 0 to 255.\n"
#define ERROR_UNCORRECT_COORDINATES "Uncorrect coordinates x: %s y:
%s\nThe coordinates must be positive integers.\n"
#define ERROR_UNCORRECT_SIDE_SIZE "Uncorrect side_size: %s\nMust be an
integer greater than 0.\n"
```

```
#define ERROR_UNCORRECT_THICKNESS "Uncorrect thickness: %s\nMust be an
integer greater than 0.\n"
#define ERROR_UNCORRECT_COLOR "Uncorrect color: %s.\n"
#define ERROR_UNCORRECT_FILL_VALUE "Uncorrect fill value: %s\nPossible
options: true, false.\n"
#define ERROR_MORE_ACTIONS "You have entered more than one command.\n"
#define ERROR_INPUT_AND_OUTPUT_SIMILAR "Input and output file are
similar.\n"
#define ERROR_UNCORRECT_EXCHANGE_TYPE "Uncorrect exchange_type:
%s\nPossible options: clockwise, diagonals.\n"
#define BMP_TYPE 0x4D42 //"BM"
#define CLOCKWISE_EXCHANGE_TYPE "clockwise"
#define DIAGONALS_EXCHANGE_TYPE "diagonals"
#define COUNTERCLOCKWISE_EXCHANCE_TYPE "counterclockwise"
#define HELP_FILE_NAME "help.txt"

BMP * read_bmp(const char * filename);

bool save_bmp(const char * filename, BMP * bmp);

void filter(FilterParams * filter_params, BMP * bmp);

void draw_square(SquareParams * square_params, BMP * bmp);

void freq_color(RGB * color, BMP * bmp);

bool exchange(ExchangeParams * exchange_params, BMP * bmp);

bool ckeck_correct_component_name(const char * component_name);

bool check_correct_component_value(int component_value);

bool to_coordinate(Point * point, char * params);

bool to_side_size(int * size, const char * side_size);

bool to_thickness(int * thickness, const char * thickness_str);

bool to_color(RGB * color, char * color_str);

bool to_exchange_type(int * exchange_type,const char *
exchange_type_str);

void info(BMP * bmp);

void compare(BMP * first, BMP * second);

void help();
```

main.c

Исходный код main.c

```
#include "bmp.h"
#include <getopt.h>

int main(int agrc, char * agrs[])
{
    struct option long_options[] =
    {
        {"rgbfilter", no_argument, NULL, 'r'},
        {"input", required_argument, NULL, 'i'},
        {"output", required_argument, NULL, 'o'},
        {"component_name", required_argument, NULL, 'n'},
        {"component_value", required_argument, NULL, 'v'},
        {"help", no_argument, NULL, 'h'},
        {"info", no_argument, NULL, 'I'},
        {"square", no_argument, NULL, 's'},
        {"left_up", required_argument, NULL, 'l'},
        {"side_size", required_argument, NULL, 'S'},
        {"thickness", required_argument, NULL , 't'},
        {"color", required_argument, NULL, 'c'},
        {"fill", no_argument, NULL, 'f'},
        {"fill_color", required_argument, NULL, 'F'},
        {"freq_color", no_argument, NULL, 'q'},
        {"exchange", no_argument, NULL, 'e'},
        {"right_down", required_argument, NULL, 'R'},
        {"exchange_type", required_argument, NULL, 'T'},
        {"compare", no_argument, NULL, 'C'},
        {0, 0, 0, 0}
    };

    char input_file[50];
    char output_file[50];
    char add_file[50];
    char mirror_arg[50];
    FilterParams * filter_params = malloc(sizeof(FilterParams));
    SquareParams * square_params = malloc(sizeof(SquareParams));
    ExchangeParams * exchange_params = malloc(sizeof(ExchangeParams));
    BMP * bmp;
    char mode = ' ';
    RGB color = {-1, -1, -1};

    int opt;
    while((opt = getopt_long(agrc, agrs, "ri:o:n:v:l:c:t:",
long_options, NULL)) != -1)
    {
        switch (opt)
        {
        case 'i':
            strcpy(input_file, optarg);
```

```
            break;
        case 'o':
            strcpy(output_file, optarg);
            break;
        case 'n':
            strcpy(filter_params->component_name, optarg);
            if(!ckeck_correct_component_name(filter_params-
>component_name))
                return 40;
            break;
        case 'v':
            filter_params->component_value = atoi(optarg);
            if(!check_correct_component_value(filter_params-
>component_value))
                return 41;
            break;
        case 'r':
            if(mode != ' ')
            {
                fprintf(stderr, ERROR_MORE_ACTIONS);
                return 41;
            }
            mode = 'f';
            break;
        case 'I':
            if(mode != ' ')
            {
                fprintf(stderr, ERROR_MORE_ACTIONS);
                return 41;
            }
            mode = 'i';
            break;
        case 's':
            if(mode != ' ')
            {
                fprintf(stderr, ERROR_MORE_ACTIONS);
                return 41;
            }
            mode = 's';
            break;
        case 'l':
            if(!to_coordinate(&square_params->left_up, optarg))
                    return 41;
            exchange_params->left_up = square_params->left_up;
            break;
        case 'S':
            if(!to_side_size(&square_params->side_size, optarg))
                return 41;
            break;
        case 't':
```

```
            if(!to_thickness(&square_params->thickness, optarg))
                return 41;
            break;
        case 'c':
            if(!to_color(&square_params->color, optarg))
                return 41;
            color = square_params->color;
            break;
        case 'f':
            square_params->fill = true;
            break;
        case 'F':
            if(!to_color(&square_params->fill_color, optarg))
                return 41;
            break;
        case 'q':
            if(mode != ' ')
            {
                fprintf(stderr, ERROR_MORE_ACTIONS);
                return 41;
            }
            mode = 'q';
            break;
        case 'e':
            if(mode != ' ')
            {
                fprintf(stderr, ERROR_MORE_ACTIONS);
                return 41;
            }
            mode = 'e';
            break;
        case 'R':
            if(!to_coordinate(&exchange_params->right_down, optarg))
                return 41;
            break;
        case 'T':
            if(!to_exchange_type(&exchange_params->mode, optarg))
                return 41;
            break;
        case 'h':
            help();
            return 0;
        default:
            break;
        }
    }

    if(strcmp(input_file, "") == 0)
    {
        strcpy(input_file, agrs[agrc - 1]);
```

```
    }

    bmp = read_bmp(input_file);
    if(!bmp) return 41;

    if(strcmp(input_file, output_file) == 0)
    {
        fprintf(stderr, ERROR_INPUT_AND_OUTPUT_SIMILAR);
        return 41;
    }

    switch (mode)
    {
    case 'i':
        info(bmp);
        return 0;
        break;
    case 's':
        draw_square(square_params, bmp);
        break;
    case 'f':
        filter(filter_params, bmp);
        break;
    case 'q':
        freq_color(&color, bmp);
        break;
    case 'e':
        exchange(exchange_params, bmp);
        break;
    default:
        break;
    }
    if(!save_bmp(output_file, bmp))
    {
        return 41;
    }

    return 0;
}
```

Makefile

Исходный код Makefile

```
OBJECTS= main.o bmp.o
all: $(OBJECTS)
    gcc $(OBJECTS) -o cw
    make clean

main.o: main.c
    gcc -c main.c
```

```
bmp.o: bmp.c
    gcc -c bmp.c
clean:
    rm *.o
```