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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Обработка изображения в формате BMP.

Студент гр. 3343 _____ Гребнев Е.Д.

Преподаватель _____ Государкин Я.С.

Санкт-Петербург
2024

ЗАДАНИЕ
на курсовую работу

Студент Гребнев Е.Д.

Группа 3343

Тема работы: Обработка изображения в формате BMP.

Исходные данные:

- 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).

Содержание пояснительной записки:

1. Содержание
2. Введение
3. Описание задачи и требований
4. Описание архитектуры программы
5. Описание структур данных
6. Полученные результаты

7. Заключение
8. Список использованных источников

Предполагаемый объем пояснительной записки:

Не менее 23 страниц.

Дата выдачи задания: 18.03.2024

Дата сдачи реферата: 15.05.2024

Дата защиты реферата: 15.05.2024

Студент _____ Гребнев Е.Д.

Преподаватель _____ Государкин Я.С.

Аннотация

Курсовая работа "Обработка BMP-файлов с использованием CLI и (опционально) GUI" разработана для реализации различных операций над изображениями в формате BMP. Программа включает в себя обработку файлов с учетом основных характеристик формата: 24 бита на цвет, без сжатия, с учетом выравнивания и порядка записи пикселей.

Функционал программы включает:

1. Отражение заданной области относительно выбранной оси (горизонтальной или вертикальной).
2. Копирование определенной области изображения в другую область.
3. Замену всех пикселей определенного цвета на другой цвет.
4. Разделение изображения на N*M частей с заданными параметрами (количество частей по осям X и Y, толщина линии и ее цвет).

Реализация каждой операции осуществляется через соответствующие флаги командной строки, что обеспечивает удобство использования программы. Важным аспектом является структурирование функций по отдельным файлам и использование системы сборки (например, make и Makefile) для компиляции проекта.

Результаты работы включают в себя полноценный функционал обработки BMP-файлов с возможностью выбора нужных операций, а также соответствие всех параметров выходного файла параметрам входного, за исключением измененных значений.

Содержание

Введение

Целью данной работы является разработка программы для обработки изображений в формате BMP с использованием командной строки (CLI) с возможностью дополнительного использования графического интерфейса (GUI). Основной задачей программы является реализация различных операций над изображениями, таких как отражение, копирование, замена цвета и разделение на части, с соблюдением всех характеристик формата BMP.

Для достижения поставленной цели предполагается реализация следующих задач:

1. Разработка алгоритмов обработки изображений, включая отражение, копирование, замену цвета и разделение на части.
2. Создание структуры программы, позволяющей управлять функционалом через командную строку.
3. Разработка алгоритмов проверки и обеспечения соответствия входных файлов формату BMP.
4. Разделение функций обработки изображений на отдельные файлы и организация сборки проекта при помощи системы сборки

Для решения поставленных задач планируется использовать язык программирования, поддерживающий работу с бинарными файлами и обработку изображений, а также системы сборки для управления процессом компиляции и сборки программы. Весь функционал программы будет разделен на отдельные функции для обеспечения модульности и повторного использования кода.

Структура работы предполагает последовательное выполнение задач, что обеспечит логический порядок выполнения работы и достижение поставленной цели.

Описание задачи и требований

Вариант 2
Программа **обязательно должна иметь CLI**

Программа должна реализовывать весь следующий функционал по обработке bmp-файла

Программа должна иметь следующую функции по обработке изображений:

1. Отражение заданной области. Флаг для выполнения данной операции: `--mirror`. Этот функционал определяется:
    - выбором оси относительно которой отражать (горизонтальная или вертикальная). Флаг `--axis`, возможные значения `x` и `y`
    - 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
2. Копирование заданной области. Флаг для выполнения данной операции: `--copy`. Функционал определяется:
    - Координатами левого верхнего угла области-источника. Флаг `--left_up`, значение задаётся в формате `left.up`, где left – координата по x, up – координата по y
    - Координатами правого нижнего угла области-источника. Флаг `--right_down`, значение задаётся в формате `right.down`, где right – координата по x, down – координата по y

- Координатами левого верхнего угла области-назначения. Флаг `--dest_left_up`, значение задаётся в формате `left.up`, где left – координата по x, up – координата по y

3. Заменяет все пиксели одного заданного цвета на другой цвет. Флаг для выполнения данной операции: `--color_replace`. Функционал определяется:
   - Цвет, который требуется заменить. Флаг `--old_color` (цвет задаётся строкой `rrr.ggg.bbb`, где rrr/ggg/bbb – числа, задающие цветовую компоненту. пример `--old_color 255.0.0` задаёт красный цвет)
   - Цвет на который требуется заменить. Флаг `--new_color` (работает аналогично флагу `--old_color`)
4. Разделяет изображение на N*M частей. Флаг для выполнения данной операции: `--split`. Реализация: провести линии заданной толщины. Функционал определяется:
   - Количество частей по “оси” Y. Флаг `--number_x`. На вход принимает число больше 1
   - Количество частей по “оси” X. Флаг `--number_y`. На вход принимает число больше 1
   - Толщина линии. Флаг `--thickness`. На вход принимает число больше 0
   - Цвет линии. Флаг `--color` (цвет задаётся строкой `rrr.ggg.bbb`, где rrr/ggg/bbb – числа, задающие цветовую компоненту. пример `--color 255.0.0` задаёт красный цвет)

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

Описание архитектуры программы

Программа разработана с использованием принципов модульности, что обеспечивает структурированный и удобный подход к разработке. Все функциональности программы разделены на отдельные модули (файлы), каждый из которых предоставляет определенные возможности. Далее представлен обзор ключевых аспектов организации программы.

2.1 Модульная структура

Программа состоит из нескольких модулей, каждый из которых отвечает за конкретную функциональность:

- /docs/*: Директория с документацией к проекту.
- /imgs/*: Картинки для тестирования.
- *bmp.cpp*: Функции для работы с изображением.
- *operation_params.cpp*: Парсинг флагов запуска программы.
- *logger.cpp*: Форматированный вывод ошибок и информации.
- *main.cpp*: Основной файл программы.

2.2 Компиляция с использованием Makefile

Все файлы компилируются с использованием Makefile, что обеспечивает автоматизацию процесса компиляции и легкость управления зависимостями между модулями.

2.3 Основная часть программы (main.cpp)

Главная часть программы, расположенная в файле main.cpp, отвечает за управление вводом команд пользователя и вызов соответствующих функций обработки. Это центральный модуль, координирующий работу программы.

2.4 Заголовочные файлы

Каждый файл с исходным кодом, за исключением main.cpp, сопровождается соответствующим заголовочным файлом (.hpp), который содержит описание сигнатур функций, необходимых для взаимодействия с другими модулями. Заголовочные файлы также включают макросы для предотвращения повторного включения (*header guards*).

Этот организационный принцип обеспечивает четкость, структурированность и возможность легкого расширения программы, так как каждый модуль отвечает за конкретный аспект функциональности.

3. Описание структур данных (*BMPHeader, RGB и другие*)

3.1 Структура *BMPHeader*

Структура данных *Sentence* предназначена для хранения предложений.

В структуре *Sentence* есть соответствующие поле для хранения массива букв (*wchar_t *sentence*).

3.2 BMPHeader (Заголовок BMP-файла):

- **Назначение**: Эта структура представляет собой заголовок BMP-файла, который содержит информацию о самом файле и о характеристиках изображения, хранящегося в этом файле.
- **Данные**:
  - **signature**: Два символа, обозначающие сигнатуру файла BMP.
  - **fileSize**: Размер файла в байтах.
  - **reserved1** и **reserved2**: Зарезервированные поля, используемые для будущего расширения.
  - **dataOffset**: Смещение, с которого начинаются данные изображения в файле.
  - **headerSize**: Размер заголовка в байтах.
  - **width** и **height**: Ширина и высота изображения в пикселях соответственно.
  - **planes**: Количество плоскостей изображения (обычно 1).
  - **bitsPerPixel**: Глубина цвета пикселя в битах (например, 24 бита на пиксель).
  - **compression**: Тип сжатия (обычно 0 для отсутствия сжатия).
  - **imageSize**: Размер данных изображения в байтах.
  - **xPixelsPerMeter** и **yPixelsPerMeter**: Горизонтальное и вертикальное разрешение в пикселях на метр соответственно.

- **colorsUsed**: Количество используемых цветов изображения (0 для всех цветов).
- **colorsImportant**: Количество важных цветов изображения (0, если все цвета важны).

- **Типы данных**: В структуре используются целочисленные типы данных, такие как `char`, `uint32_t` и `uint16_t`, для хранения значений различных полей заголовка.

3.3 RGB (Цвет в формате RGB):

- **Назначение**: Эта структура представляет цвет в формате RGB (красный, зеленый, синий).
- **Данные**:
  - **red**: Компонента красного цвета.
  - **green**: Компонента зеленого цвета.
  - **blue**: Компонента синего цвета.
- **Типы данных**: В структуре используются беззнаковые 8-битные целочисленные типы (`uint8_t`), чтобы представить значения компонент цвета в диапазоне от 0 до 255.

3.4 Coordinate (Координаты точки на плоскости):

- **Назначение**: Эта структура представляет координаты точки на плоскости.
- **Данные**:
  - **x**: Координата x точки.
  - **y**: Координата y точки.
- **Типы данных**: Используются знаковые 32-битные целочисленные типы (`int`), представляющие координаты точки.

3.5 Operations (Параметры операции над изображением):

- **Назначение**: Эта структура представляет параметры операции над изображением, которые можно выполнить, такие как отражение,

выделение области, копирование, замена цвета и разделение изображения на части.

- **Данные**:
  - Различные параметры и флаги, управляющие выполнением операций, такие как путь к входному и выходному файлам, информация об изображении, параметры отражения, выделения области, копирования, замены цвета, разделения и другие.
- **Типы данных**: В структуре используются различные типы данных, такие как строки (`std::string`), логические значения (`bool`), целочисленные типы и структуры для координат и цветов.

Эти структуры представляют собой основу для обработки и работы с изображениями в формате BMP, предоставляя необходимую информацию о файлах и параметры для выполнения различных операций над изображениями.

Полученные результаты

Программа демонстрирует успешное выполнение поставленных задач, предоставляя пользователю гибкость в выборе операций для обработки текста. Полученные результаты включают в себя следующие ключевые аспекты:

Функциональность:

- Программа успешно реализует функции обработки текста, предоставляя пользователю возможность выбора различных операций.
- Каждая функция обработки текста выполняется корректно в соответствии с поставленными требованиями.

Гибкость и управление ошибками:

- Пользователю предоставляется удобный интерфейс для выбора операций, что обеспечивает гибкость использования программы.
- Реализована обработка ошибок, что позволяет программе адекватно реагировать на некорректные сценарии выполнения.

Вывод сообщений:

- Программа предоставляет понятные и информативные сообщения пользователю в случае успешного выполнения операций или возникновения ошибок.
- Вывод информации о выполненных операциях структурирован и понятен для пользователя

Стабильность:

- Программа демонстрирует стабильную работу, обеспечивая надежное выполнение операций над изображение.

Тестирование:

- Полученные результаты подтверждают успешное прохождение тестирования на различных сценариях использования, что поддерживает корректность и надежность программы.

В целом, программа достигла поставленных целей, предоставляя пользователям эффективные средства обработки текстовой информации с учетом заданных требований.

6.Заключение

Проект успешно реализован, выполнив все поставленные задачи и достигнув заявленных целей. В ходе разработки были задействованы стандартные средства языка программирования C, включая динамическое выделение памяти, использование структур данных и вызов стандартных библиотечных функций.

Основные характеристики проекта:

1. **Использование стандартных средств C++:**
   - Программа в полной мере использует возможности языка программирования C++, включая динамическое выделение памяти, структуры данных и стандартные библиотечные функции.
2. **Модульная структура:**
   - Программа разработана с учетом модульной структуры, что обеспечивает легкость поддержки и возможность дальнейшего расширения функциональности.
3. **Эффективность и надежность:**

- Реализованный функционал обеспечивает эффективное выполнение задач обработки изображения, а также обеспечивает стабильность и надежность работы программы.

4. **Структурированный код:**
   - Исходный код программы поддерживает высокий уровень структурированности, что упрощает понимание и поддержку кодовой базы.

Приложения

Приложения А
main.cpp

```
/**
 * @file   main.cpp
 * @brief  Главный файл программы
 */

#include "bmp.hpp"
#include "logger.hpp"
#include "messages.hpp"
#include "operation_params.hpp"

#define IMG_DIR ""
/**
 * @brief  Главная функция программы
 * @param  argc Количество аргументов командной строки
 * @param  argv Массив аргументов командной строки
 * @return Код возврата
 */
int main(int argc, char* argv[])
{

    Logger::log(hello_message);

    // Парсинг параметров командной строки
    Operations params = parse_command_line(argc, argv);

    const std::string input_file = IMG_DIR + params.input_file;

    // Загрузка изображения BMP
    BMP bmp(input_file);
    if (!bmp.is_valid()) { Logger::exit(1, invalid_bmp_message); }

    // Вывод информации о изображении, если соответствующий флаг
установлен
    if (params.info) { bmp.get_info(); }

    // Зеркальное отображение изображения, если соответствующий флаг
установлен
    if (params.mirror)
    {
        Logger::warn(mirror_warning);
```

```
        bmp.mirror(params.axis, params.left_up, params.right_down);
        Logger::log(success_message);
    }

    // Замена цветов на изображении, если соответствующий флаг установлен
    if (params.color_replace)
    {
        Logger::warn(color_replace_warning);
        bmp.color_replace(params.old_color, params.new_color);
        Logger::log(success_message);
    }

    // Разделение изображения на части, если соответствующий флаг
установлен
    if (params.split)
    {
        Logger::warn(image_split_warning);
        bmp.split(params.number_x, params.number_y, params.thickness,
params.line_color);
        Logger::log(success_message);
    }

    // Копирование области изображения, если соответствующий флаг
установлен
    if (params.copy)
    {
        Logger::warn(image_copy_warning);
        bmp.copy(params.left_up, params.right_down, params.dest_left_up);
        Logger::log(success_message);
    }

    // Сохранение изображения
    bmp.save(params.output_file);

    return EXIT_SUCCESS;
}
```

logger.cpp

```
/**
 * @file logger.cpp
 * @brief Implementation file for the Logger class.
 */
#include "logger.hpp"
#include "messages.hpp"

bool Logger::colors_enabled = false;

void set_color(const Color color, std::ostream& stream = std::cout)
{
    switch (color)
```

```
    {
    case Color::RED: stream << "\033[31m"; break;
    case Color::GREEN: stream << "\033[32m"; break;
    case Color::YELLOW: stream << "\033[33m"; break;
    case Color::BLUE: stream << "\033[34m"; break;
    case Color::MAGENTA: stream << "\033[35m"; break;
    case Color::CYAN: stream << "\033[36m"; break;
    case Color::WHITE: stream << "\033[37m"; break;
    }
}

void reset_color(std::ostream& stream = std::cout)
{
    stream << "\033[0m"; // Reset color
}

Logger::Logger(bool enableColors) { set_colors_enabled(enableColors); }

void Logger::set_colors_enabled(bool enableColors) { colors_enabled =
enableColors; }
template <typename Message> void Logger::log(const Message& message,
Color color, std::ostream& stream)
{
    if (colors_enabled) { set_color(color, stream); }

    stream << message << std::endl;

    if (colors_enabled) { reset_color(stream); }
}

void Logger::warn(const std::string& message, std::ostream& stream) {
log(message, Color::YELLOW, stream); }

void Logger::error(const std::string& message, std::ostream& stream) {
log(message, Color::RED, stream); }

void Logger::exit(int exitCode, const std::string& exitMessage,
std::ostream& stream)
{
    if (!exitMessage.empty()) { error(exitMessage, stream); }
    std::exit(exitCode);
}
```

structures.hpp

```
/**
 * @file structures.h
 * @brief Заголовочный файл, содержащий определения структур и классов.
 */

#pragma once
```

```
#include <string>

#pragma pack(push, 1)

/**
 * @brief Структура, представляющая заголовок BMP-файла.
 */
struct BMPHeader
{
    char signature[2];        /**< Сигнатура файла BMP. */
    uint32_t fileSize;        /**< Размер файла в байтах. */
    uint16_t reserved1;       /**< Зарезервировано для использования. */
    uint16_t reserved2;       /**< Зарезервировано для использования. */
    uint32_t dataOffset;      /**< Смещение, с которого начинаются данные
изображения. */
    uint32_t headerSize;      /**< Размер заголовка в байтах. */
    int32_t width;            /**< Ширина изображения в пикселях. */
    int32_t height;           /**< Высота изображения в пикселях. */
    uint16_t planes;          /**< Количество плоскостей. */
    uint16_t bitsPerPixel;    /**< Глубина цвета пикселя в битах. */
    uint32_t compression;     /**< Тип сжатия. */
    uint32_t imageSize;       /**< Размер данных изображения. */
    int32_t xPixelsPerMeter;  /**< Горизонтальное разрешение в пикселях
на метр. */
    int32_t yPixelsPerMeter;  /**< Вертикальное разрешение в пикселях на
метр. */
    uint32_t colorsUsed;      /**< Количество используемых цветов
изображения. */
    uint32_t colorsImportant; /**< Количество важных цветов изображения.
*/
};

#pragma pack(pop)

/**
 * @brief Структура, представляющая цвет в формате RGB.
 */
struct RGB
{
    uint8_t red;   /**< Компонента красного цвета. */
    uint8_t green; /**< Компонента зеленого цвета. */
    uint8_t blue;  /**< Компонента синего цвета. */
    RGB(uint8_t r = 0, uint8_t g = 0, uint8_t b = 0)
    : red(r)
    , green(g)
    , blue(b)
    {
    }
};
```

```
/**
 * @brief Структура, представляющая координаты точки на плоскости.
 */
struct Coordinate
{
    int32_t x; /**< Координата x. */
    int32_t y; /**< Координата y. */
};

/**
 * @brief Структура, представляющая параметры операции над изображением.
 */
struct Operations
{
    std::string input_file;              /**< Путь к входному файлу. */
    std::string output_file = "out.bmp"; /**< Путь к выходному файлу (по
умолчанию "out.bmp"). */
    bool info = false;                   /**< Флаг вывода информации о
изображении. */
    bool mirror = false;                 /**< Флаг отражения изображения.
*/
    std::string axis;                    /**< Ось отражения
(горизонтальная или вертикальная). */
    Coordinate left_up;                  /**< Левая верхняя точка для
выделения области. */
    Coordinate right_down;               /**< Правая нижняя точка для
выделения области. */
    bool copy = false;                   /**< Флаг копирования выделенной
области. */
    Coordinate dest_left_up;             /**< Левая верхняя точка для
вставки скопированной области. */
    bool color_replace = false;          /**< Флаг замены цвета. */
    RGB old_color;                       /**< Старый цвет, который будет
заменен. */
    RGB new_color;                       /**< Новый цвет, на который
будет заменен старый цвет. */
    bool split = false;                  /**< Флаг разделения изображения
на части. */
    int32_t number_x = 1;                    /**< Количество частей по
горизонтали. */
    int32_t number_y = 1;                    /**< Количество частей по
вертикали. */
    int32_t thickness = 1;                   /**< Толщина линии при
разделении изображения на части. */
    RGB line_color;                      /**< Цвет линии разделения. */

    /**
     * @brief Конструктор по умолчанию для инициализации параметров
операции.
```

```
     */
    Operations()
    : input_file()
    , output_file("out.bmp")
    , info(false)
    , mirror(false)
    , axis()
    , left_up()
    , right_down()
    , copy(false)
    , dest_left_up()
    , color_replace(false)
    , old_color()
    , new_color()
    , split(false)
    , number_x(1)
    , number_y(1)
    , thickness(1)
    , line_color()
    {
    }
};
```

bmp.cpp

```
/**
 * @file bmp.cpp
 * @brief Реализация методов класса BMP для работы с изображениями в
формате
 * BMP.
 */

#include "bmp.hpp"
#include "logger.hpp"
#include "messages.hpp"

BMP::BMP(const std::string& fileName)
: header()
, pixel_data()
{
    std::ifstream file(fileName, std::ios::binary);
    if (!file.is_open()) { Logger::exit(ERR_FILE_NOT_FOUND,
open_bmp_error + fileName); }

    file.read(reinterpret_cast<char*>(&header), sizeof(header));

    if (!validate_header())
    {
        file.close();
        Logger::exit(ERR_INCORRECT_FILE_FORMAT, invalid_header_error +
fileName);
```

```
    }

    const uint32_t bytesPerPixel = header.bitsPerPixel / 8;
    const uint32_t rowSize = ((header.width * bytesPerPixel + 3) / 4) *
4;
    const uint32_t imageSize = rowSize * header.height;

    pixel_data.resize(imageSize);

    file.seekg(header.dataOffset, std::ios_base::beg);

    file.read(reinterpret_cast<char*>(pixel_data.data()), imageSize);

    file.close();
}

bool BMP::validate_header() const
{
    if (std::strncmp(header.signature, "BM", 2) != 0)
    {
        Logger::exit(ERR_INCORRECT_FILE_FORMAT, invalid_signature_error);
        return false;
    }

    if (header.width <= 0 || header.height <= 0)
    {
        Logger::exit(ERR_INCORRECT_FILE_FORMAT,
invalid_dimensions_error);
        return false;
    }

    if (header.bitsPerPixel != 24) { Logger::warn(invalid_bpp_warning); }

    if (header.compression != 0)
    {
        Logger::exit(ERR_INCORRECT_FILE_FORMAT,
unsupported_compression_error);
        return false;
    }

    return true;
}

bool BMP::is_valid() const { return !pixel_data.empty(); }

RGB BMP::get_color(int x, int y) const
{
    if (x < 0 || x >= header.width || y < 0 || y >= header.height) return
RGB();
```

```
    const uint32_t bytesPerPixel = header.bitsPerPixel / 8;
    const uint32_t bytesPerRow = (bytesPerPixel * header.width + 3) & ~3;
    const uint32_t index = ((header.height - 1 - y) * bytesPerRow) + (x *
bytesPerPixel);

    return RGB(pixel_data[index + 2], pixel_data[index + 1],
pixel_data[index]);
}

void BMP::set_color(int x, int y, const RGB& newColor)
{
    if (x < 0 || x >= header.width || y < 0 || y >= header.height)
return;

    const uint32_t bytesPerPixel = header.bitsPerPixel / 8;
    const uint32_t bytesPerRow = (bytesPerPixel * header.width + 3) & ~3;
    const uint32_t index = ((header.height - 1 - y) * bytesPerRow) + (x *
bytesPerPixel);

    pixel_data[index] = newColor.blue;
    pixel_data[index + 1] = newColor.green;
    pixel_data[index + 2] = newColor.red;
}

void BMP::color_replace(const RGB& old_color, const RGB& new_color)
{
    for (int y = 0; y < header.height; y++)
    {
        for (int x = 0; x < header.width; x++)
        {
            RGB current_color = get_color(x, y);
            if (current_color.red == old_color.red && current_color.green
== old_color.green && current_color.blue == old_color.blue) {
set_color(x, y, new_color); }
        }
    }
}

void BMP::mirror(const std::string& axis, const Coordinate& left_up,
const Coordinate& right_down)
{
    int width = right_down.x - left_up.x;
    int height = right_down.y - left_up.y;

    if (axis != "x" && axis != "y") { Logger::exit(ERR_INVALID_ARGUMENT,
invalid_mirror_axis_error); }

    if (axis == "x")
    {
        for (int y = left_up.y; y <= right_down.y; ++y)
```

```
        {
            for (int x = left_up.x; x < left_up.x + width / 2; ++x)
            {
                int mirroredX = right_down.x - (x - left_up.x);
                RGB tempColor = get_color(x, y);
                set_color(x, y, get_color(mirroredX, y));
                set_color(mirroredX, y, tempColor);
            }
        }
    }
    else if (axis == "y")
    {
        for (int y = left_up.y; y < left_up.y + height / 2; ++y)
        {
            for (int x = left_up.x; x <= right_down.x; ++x)
            {
                int mirroredY = right_down.y - (y - left_up.y);
                RGB tempColor = get_color(x, y);
                set_color(x, y, get_color(x, mirroredY));
                set_color(x, mirroredY, tempColor);
            }
        }
    }
}
void BMP::split(int number_x, int number_y, int thickness, const RGB&
color)
{
    if ((number_x <= 0 || number_x > header.width) || (number_y <= 0 ||
number_y > header.height) || (thickness <= 0 || thickness >
header.width)) { Logger::exit(ERR_INVALID_ARGUMENT,
invalid_split_parameters_error); }

    int gap;

    for (int i = 1; i < number_y; i++)
    {
        gap = header.height / number_y;
        for (int x = 0; x < header.width; x++)
        {
            for (int y = 0; y < thickness; y++) { set_color(x, i * gap +
y, color); }
        }
    }

    for (int i = 1; i < number_x; i++)
    {
        gap = header.width / number_x;
        for (int x = 0; x < thickness; x++)
        {
```

```
            for (int y = 0; y < header.height; y++) { set_color(i * gap +
x, y, color); }
        }
    }
}

void BMP::copy(const Coordinate& src_left_up, const Coordinate&
src_right_down, const Coordinate& dest_left_up)
{
    int src_x_min = std::min(src_left_up.x, src_right_down.x);
    int src_x_max = std::max(src_left_up.x, src_right_down.x);
    int src_y_min = std::min(src_left_up.y, src_right_down.y);
    int src_y_max = std::max(src_left_up.y, src_right_down.y);

    for (int x = src_x_min; x <= src_x_max; x++)
    {
        for (int y = src_y_min; y <= src_y_max; y++) {
set_color(dest_left_up.x + (x - src_x_min), dest_left_up.y + (y -
src_y_min), get_color(x, y)); }
    }
}
void BMP::save(const std::string& fileName)
{
    std::ofstream file(fileName, std::ios::binary);
    if (!file.is_open())
    {
        Logger::exit(ERR_FILE_WRITE_ERROR, failed_create_output_file +
fileName);
        return;
    }

    const uint32_t bytesPerPixel = header.bitsPerPixel / 8;
    const uint32_t rowSize = ((header.width * bytesPerPixel + 3) / 4) *
4;

    file.write(reinterpret_cast<const char*>(&header), sizeof(header));
    file.seekp(header.dataOffset, std::ios::beg);
    for (int y = 0; y < header.height; ++y) {
file.write(reinterpret_cast<const char*>(pixel_data.data() + y *
rowSize), rowSize); }

    file.close();
}

void BMP::get_info() const
{
    Logger::log(signature_message + std::string(header.signature, 2));
    Logger::log(file_size_message + std::to_string(header.fileSize) + "
bytes");
```

```
    Logger::log(data_offset_message + std::to_string(header.dataOffset) +
" bytes");
    Logger::log(header_size_message + std::to_string(header.headerSize) +
" bytes");
    Logger::log(image_dimensions_message + std::to_string(header.width) +
"x" + std::to_string(header.height));
    Logger::log(bits_per_pixel_message +
std::to_string(header.bitsPerPixel));
    Logger::log(compression_message +
std::to_string(header.compression));
    Logger::log(image_size_message + std::to_string(header.imageSize) + "
bytes");
    Logger::log(pixels_per_meter_x_message +
std::to_string(header.xPixelsPerMeter));
    Logger::log(pixels_per_meter_y_message +
std::to_string(header.yPixelsPerMeter));
    Logger::log(colors_used_message + std::to_string(header.colorsUsed));
    Logger::log(important_colors_message +
std::to_string(header.colorsImportant));
}
```

Operation_params.cpp

```
#include "operation_params.hpp"
#include "logger.hpp"
#include "messages.hpp"

std::vector<int> parse_values(const std::string& str)
{
    std::vector<int> values;
    std::stringstream ss(str);
    std::string token;
    while (std::getline(ss, token, '.'))
    {
        try
        {
            values.push_back(std::stoi(token));
        }
        catch (const std::invalid_argument& e)
        {
            Logger::exit(ERR_INVALID_ARGUMENT, invalid_argument_error +
token );
        }
    }
    return values;
}

RGB parse_RGB(const std::string& str)
{
    std::vector<int> values = parse_values(str);
```

```
    if (values.size() != 3) { Logger::exit(ERR_INVALID_ARGUMENT,
invalid_color_format_error); }
    for (int value : values)
    {
        if (value < 0 || value > 255) {
Logger::exit(ERR_INVALID_ARGUMENT, invalid_color_range_error +
std::to_string(value)); }
    }
    return { static_cast<uint8_t>(values[0]),
static_cast<uint8_t>(values[1]), static_cast<uint8_t>(values[2]) };
}

Coordinate parseCoordinate(const std::string& str)
{
    Coordinate coord;
    std::vector<int> values = parse_values(str);
    if (values.size() != 2) { Logger::exit(ERR_INVALID_ARGUMENT,
invalid_color_format_error); }
    coord.x = values[0];
    coord.y = values[1];
    return coord;
}

void display_help()
{
    Logger::log(help_usage_description);
    Logger::log(help_usage_start);
    Logger::log(mirror_option_description);
    Logger::log(axis_option_description);
    Logger::log(left_up_option_description);
    Logger::log(right_down_option_description);
    Logger::log(dest_left_up_option_description);
    Logger::log(old_color_option_description);
    Logger::log(new_color_option_description);
    Logger::log(color_option_description);
    Logger::log(copy_option_description);
    Logger::log(color_replace_option_description);
    Logger::log(split_option_description);
    Logger::log(number_x_option_description);
    Logger::log(number_y_option_description);
    Logger::log(thickness_option_description);
    Logger::log(output_option_description);
    Logger::log(input_option_description);
    Logger::log(help_option_description);
}

Operations parse_command_line(int argc, char* argv[])
{
    Operations params;
```

```
    const std::map<int, std::function<void(const char*)>> optionHandlers
= {
        { 'h',
          [&](const char*)
          {
              if (argc != 2) Logger::exit(ERR_INVALID_ARGUMENT,
invalid_argument_error + "--help (-h)");
              display_help();
              Logger::exit(EXIT_SUCCESS, "");
          } },
        { 'i', [&](const char* option_argument) { params.input_file =
option_argument; } },
        { 'o', [&](const char* option_argument) { params.output_file =
option_argument; } },
        { 256, [&](const char*) { params.mirror = true; } },
        { 257,
          [&](const char* option_argument)
          {
              if (strcmp(option_argument, "x") != 0 &&
strcmp(option_argument, "y") != 0) Logger::exit(ERR_INVALID_ARGUMENT,
invalid_argument_error + "--axis (x/y)");
              params.axis = option_argument;
          } },
        { 258, [&](const char* option_argument) { params.left_up =
parseCoordinate(option_argument); } },
        { 259, [&](const char* option_argument) { params.right_down =
parseCoordinate(option_argument); } },
        { 260, [&](const char* option_argument) { params.dest_left_up =
parseCoordinate(option_argument); } },
        { 261, [&](const char* option_argument) { params.old_color =
parse_RGB(option_argument); } },
        { 262, [&](const char* option_argument) { params.new_color =
parse_RGB(option_argument); } },
        { 263, [&](const char* option_argument) { params.line_color =
parse_RGB(option_argument); } },
        { 264, [&](const char*) { params.copy = true; } },
        { 265, [&](const char*) { params.color_replace = true; } },
        { 266, [&](const char*) { params.split = true; } },
        { 267, [&](const char* option_argument) { params.number_x =
parse_values(option_argument)[0]; } },
        { 268, [&](const char* option_argument) { params.number_y =
parse_values(option_argument)[0]; } },
        { 269, [&](const char* option_argument) { params.thickness =
parse_values(option_argument)[0]; } },
        { 270, [&](const char*) { params.info = true; } },
        { 271, [&](const char*) { Logger::set_colors_enabled(true); } },
    };

    const char* short_options = "hi:o:";
```

```
    static struct option long_options[] = { { "help", no_argument,
nullptr, 'h' }, { "input", required_argument, nullptr, 'i' }, { "output",
required_argument, nullptr, 'o' }, { "mirror", no_argument, nullptr, 256
}, { "axis", required_argument, nullptr, 257 }, { "left_up",
required_argument, nullptr, 258 }, { "right_down", required_argument,
nullptr, 259 }, { "dest_left_up", required_argument, nullptr, 260 }, {
"old_color", required_argument, nullptr, 261 }, { "new_color",
required_argument, nullptr, 262 }, { "color", required_argument, nullptr,
263 }, { "copy", no_argument, nullptr, 264 }, { "color_replace",
no_argument, nullptr, 265 }, { "split", no_argument, nullptr, 266 }, {
"number_x", required_argument, nullptr, 267 }, { "number_y",
required_argument, nullptr, 268 }, { "thickness", required_argument,
nullptr, 269 }, { "info", no_argument, nullptr, 270 }, { "colorful",
no_argument, nullptr, 271 }, { nullptr, 0, nullptr, 0 } };

    int opt;

    while ((opt = getopt_long(argc, argv, short_options, long_options,
nullptr)) != -1)
    {
        auto handler = optionHandlers.find(opt);
        if (handler != optionHandlers.end()) { handler->second(optarg); }
    }

    if (params.mirror && params.copy) {
Logger::exit(ERR_INVALID_ARGUMENT, double_function_use_err); }

    if (params.input_file.empty())
    {
        if (optind == argc - 1) { params.input_file = argv[optind]; }
        else if (optind < argc - 1) { Logger::exit(ERR_INVALID_ARGUMENT,
too_many_args_err); }
        else { Logger::exit(ERR_INVALID_ARGUMENT, invalid_bmp_message); }
    }

    if (params.input_file == params.output_file) {
Logger::exit(ERR_INVALID_ARGUMENT, same_input_output_message); }

    return params;
}
```

bmp.hpp

```
/**
 * @file bmp.h
 * @brief Заголовочный файл, содержащий определения структур и классов.
 */

#pragma once

#include "operation_params.hpp"
#include "structures.hpp"
```

```
#include <cstring>
#include <fstream>

/**
 * @brief Класс для работы с изображениями в формате BMP.
 */
class BMP
{
private:
    BMPHeader header;            ///< Заголовок BMP файла.
    bool validate_header() const; ///< Проверка корректности заголовка.

    std::vector<uint8_t> pixel_data; ///< Пиксельные данные изображения.

public:
    /**
     * @brief Конструктор класса BMP.
     * @param fileName Путь к файлу изображения.
     */
    BMP(const std::string& file_name);

    /**
     * @brief Получить информацию о изображении.
     */
    void get_info() const;

    /**
     * @brief Отразить изображение относительно указанной оси.
     * @param axis Ось отражения ("x" или "y").
     * @param left_up Координаты левого верхнего угла области отражения.
     * @param right_down Координаты правого нижнего угла области
отражения.
     */
    void mirror(const std::string& axis, const Coordinate& left_up, const
Coordinate& right_down);

    /**
     * @brief Сохранить изображение в файл.
     * @param fileName Имя файла для сохранения изображения.
     */
    void save(const std::string& fileName);

    /**
     * @brief Проверить, является ли изображение валидным.
     * @return true, если изображение валидно, иначе false.
     */
    bool is_valid() const;

    /**
     * @brief Копировать область изображения.
```

```
     * @param src_left_up Координаты левого верхнего угла исходной
области.
     * @param src_right_down Координаты правого нижнего угла исходной
области.
     * @param dest_left_up Координаты левого верхнего угла целевой
области.
     */
    void copy(const Coordinate& src_left_up, const Coordinate&
src_right_down, const Coordinate& dest_left_up);

    /**
     * @brief Заменить цвет на изображении.
     * @param old_color Старый цвет.
     * @param new_color Новый цвет.
     */
    void color_replace(const RGB& old_color, const RGB& new_color);

    /**
     * @brief Разделить изображение на части.
     * @param number_x Количество частей по горизонтали.
     * @param number_y Количество частей по вертикали.
     * @param thickness Толщина разделительных линий.
     * @param color Цвет разделительных линий.
     */
    void split(int number_x, int number_y, int thickness, const RGB&
color);

private:
    /**
     * @brief Получить цвет пикселя по координатам.
     * @param x Координата x пикселя.
     * @param y Координата y пикселя.
     * @return Цвет пикселя.
     */
    RGB get_color(int x, int y) const;

    /**
     * @brief Установить цвет пикселя по координатам.
     * @param x Координата x пикселя.
     * @param y Координата y пикселя.
     * @param newColor Новый цвет пикселя.
     */
    void set_color(int x, int y, const RGB& new_color);
};
```

logger.hpp

```
/**
 * @file logger.h
 * @brief Заголовочный файл, содержащий определения структур и классов.
 */
```

```
#pragma once

#include <iostream> // Include <iostream> for std::ostream
#include <string>

// Enum для цветов
enum class Color
{
    RED,     /**< Красный цвет */
    GREEN,   /**< Зеленый цвет */
    YELLOW,  /**< Желтый цвет */
    BLUE,    /**< Синий цвет */
    MAGENTA, /**< Пурпурный цвет */
    CYAN,    /**< Голубой цвет */
    WHITE    /**< Белый цвет */
};

class Logger
{
private:
    static bool colors_enabled; /**< Флаг, определяющий, разрешены ли
цвета в выводе. */

public:
    /**
     * @brief Конструктор для класса Logger.
     * @param enable_colors Если true, разрешает использование цветов в
логах, иначе - нет.
     */
    Logger(bool enable_colors);

    /**
     * @brief Устанавливает разрешение использования цветов в выводе.
     * @param enableColors Если true, разрешает использование цветов в
логах, иначе - нет.
     */
    static void set_colors_enabled(bool enableColors);

    /**
     * @brief Записывает сообщение в лог с определенным цветом.
     * @param message Сообщение для записи в лог.
     * @param color Цвет сообщения (по умолчанию GREEN).
     * @param stream Поток вывода (по умолчанию std::cout).
     */
    template<typename Message>
    static void log(const Message& message, Color color = Color::GREEN,
std::ostream& stream = std::cout);

    /**
     * @brief Записывает предупреждение в лог с желтым цветом.
```

```
     * @param message Сообщение предупреждения для записи в лог.
     * @param stream Поток вывода (по умолчанию std::cout).
     */
    static void warn(const std::string& message, std::ostream& stream =
std::cout);

    /**
     * @brief Записывает ошибку в лог с красным цветом.
     * @param message Сообщение ошибки для записи в лог.
     * @param stream Поток вывода (по умолчанию std::cerr).
     */
    static void error(const std::string& message, std::ostream& stream =
std::cerr);

    /**
     * @brief Записывает сообщение и завершает программу с заданным кодом
выхода.
     * @param exitCode Код выхода.
     * @param exitMessage Сообщение о завершении программы.
     * @param stream Поток вывода (по умолчанию std::cerr).
     */
    static void exit(int exitCode, const std::string& exitMessage = "",
std::ostream& stream = std::cerr);
};
```

messages.hpp

```
/**
 * @file messages.h
 * @brief Заголовочный файл, содержащий определения структур и классов.
 */

#pragma once
/**
 * @brief Error code for file not found.
 */
#define ERR_FILE_NOT_FOUND 40

/**
 * @brief Error code for incorrect file format.
 */
#define ERR_INCORRECT_FILE_FORMAT 41

/**
 * @brief Error code for file write error.
 */
#define ERR_FILE_WRITE_ERROR 42

/**
 * @brief Error code for invalid argument.
 */
```

```
#define ERR_INVALID_ARGUMENT 43

/**
 * @brief Error code for insufficient arguments.
 */
#define ERR_INSUFFICIENT_ARGUMENTS 45

const std::string hello_message = "Course work for option 5.2, created by
Egor Grebnev.";
/**
 * @brief Message for invalid BMP file.
 */
const std::string invalid_bmp_message = "Invalid bmp file!";

/**
 * @brief Message when input file is same as output file.
 */
const std::string same_input_output_message = "Input file is the same as
output file!";

/**
 * @brief Success message.
 */
const std::string success_message = "Success!";

/**
 * @brief Warning for requested mirror operation.
 */
const std::string mirror_warning = "=Mirror operation requested!=";

/**
 * @brief Warning for requested color replace operation.
 */
const std::string color_replace_warning = "=Color replace operation
requested!=";

/**
 * @brief Warning for requested image split operation.
 */
const std::string image_split_warning = "=Image split operation
requested=";

/**
 * @brief Warning for requested image copy operation.
 */
const std::string image_copy_warning = "=Image copy operation
requested=";

// OPERATIONS
```

```
/**
 * @brief Error for invalid color format.
 */
const std::string invalid_color_format_error = "Invalid color format";

/**
 * @brief Error when color is out of range [0-255].
 */
const std::string invalid_color_range_error = "Color out of range [0-255]
got: ";

/**
 * @brief Error for invalid coordinate format.
 */
const std::string invalid_coordinate_format_error = "Invalid coordinate
format";

/**
 * @brief Error for invalid argument for certain operation.
 */
const std::string invalid_argument_error = "Invalid argument for ";

/**
 * @brief Warning for unexpected option.
 */
const std::string unexpected_option_warning = "Unexpected option: ";

// BMP

/**
 * @brief Error when failed to open input BMP file.
 */
const std::string open_bmp_error = "Failed to open input BMP file: ";

/**
 * @brief Error when BMP file header is invalid.
 */
const std::string invalid_header_error = "BMP file header is invalid: ";

/**
 * @brief Error for invalid BMP file signature.
 */
const std::string invalid_signature_error = "Invalid BMP file signature";

/**
 * @brief Error for invalid BMP dimensions.
 */
const std::string invalid_dimensions_error = "Invalid BMP dimensions";

/**
```

```
 * @brief Warning for invalid BMP bits per pixel.
 */
const std::string invalid_bpp_warning = "Invalid BMP bits per pixel,
output image may be incorrect";

/**
 * @brief Error for unsupported BMP compression type.
 */
const std::string unsupported_compression_error = "Unsupported BMP
compression type";

/**
 * @brief Error for invalid BMP image size.
 */
const std::string invalid_image_size_error = "Invalid BMP image size,
output image may be incorrect";

/**
 * @brief Warning for trying to access color outside image bounds.
 */
const std::string access_outside_bounds_warning = "Trying to access color
outside image bounds";

/**
 * @brief Warning for trying to set color outside image bounds.
 */
const std::string set_outside_bounds_warning = "Trying to set color
outside image bounds";

/**
 * @brief Error for invalid mirror axis specified.
 */
const std::string invalid_mirror_axis_error = "Invalid mirror axis
specified";

/**
 * @brief Error for invalid split parameters.
 */
const std::string invalid_split_parameters_error = "Invalid split
parameters";

/**
 * @brief Error when copying region exceeds destination image boundaries.
 */
const std::string copy_exceeds_bounds_error = "Copying region exceeds
destination image boundaries.";

/**
 * @brief Error when failed to create output BMP file.
 */
```

```
const std::string failed_create_output_file = "Failed to create output
BMP file: ";

/**
 * @brief Error for invalid copy region or destination parameters.
 */
const std::string invalid_copy_region = "Invalid copy region or
destination parameters";

// Help message descriptions

/**
 * @brief Usage description of the program.
 */
const std::string help_usage_description = "Usage: program_name [options]
filename";

/**
 * @brief Start of the description of options.
 */
const std::string help_usage_start = "Options: ";

/**
 * @brief Description of --mirror option.
 */
const std::string mirror_option_description = "  --
mirror               Mirror operation";

/**
 * @brief Description of --axis option.
 */
const std::string axis_option_description = "  --axis
<value>         Axis of operation";

/**
 * @brief Description of --left_up option.
 */
const std::string left_up_option_description = "  --left_up
<x.y>        Coordinates of left-up corner";

/**
 * @brief Description of --right_down option.
 */
const std::string right_down_option_description = "  --right_down
<x.y>     Coordinates of right-down corner";

/**
 * @brief Description of --dest_left_up option.
 */
```

```
const std::string dest_left_up_option_description = "  --dest_left_up
<x.y>   Coordinates of destination left-up corner";

/**
 * @brief Description of --old_color option.
 */
const std::string old_color_option_description = "  --old_color
<r.g.b>    Old color to replace";

/**
 * @brief Description of --new_color option.
 */
const std::string new_color_option_description = "  --new_color
<r.g.b>    New color to replace with";

/**
 * @brief Description of --color option.
 */
const std::string color_option_description = "  --color
<r.g.b>        Color of line";

/**
 * @brief Description of --copy option.
 */
const std::string copy_option_description = "  --
copy                Copy operation";

/**
 * @brief Description of --color_replace option.
 */
const std::string color_replace_option_description = "  --
color_replace        Color replace operation";

/**
 * @brief Description of --split option.
 */
const std::string split_option_description = "  --
split               Split operation";

/**
 * @brief Description of --number_x option.
 */
const std::string number_x_option_description = "  --number_x
<value>     Number of elements along x-axis";

/**
 * @brief Description of --number_y option.
 */
const std::string number_y_option_description = "  --number_y
<value>     Number of elements along y-axis";
```

```
/**
 * @brief Description of --thickness option.
 */
const std::string thickness_option_description = "  --thickness
<value>    Thickness of operation";

/**
 * @brief Description of --output option.
 */
const std::string output_option_description = "  -o, --output
<file>    Output file";

/**
 * @brief Description of --input option.
 */
const std::string input_option_description = "  -i, --input
<file>     Input file";

/**
 * @brief Description of --help option.
 */
const std::string help_option_description = "  -h, --
help             Display this information";

// Info message

/**
 * @brief Signature of the file message.
 */
const std::string signature_message = "Signature: ";

/**
 * @brief File size message.
 */
const std::string file_size_message = "File size: ";

/**
 * @brief Data offset message.
 */
const std::string data_offset_message = "Data offset: ";

/**
 * @brief Header size message.
 */
const std::string header_size_message = "Header size: ";

/**
 * @brief Image dimensions message.
 */
```

```
const std::string image_dimensions_message = "Image dimensions: ";

/**
 * @brief Bits per pixel message.
 */
const std::string bits_per_pixel_message = "Bits per pixel: ";

/**
 * @brief Compression message.
 */
const std::string compression_message = "Compression: ";

/**
 * @brief Image size message.
 */
const std::string image_size_message = "Image size: ";

/**
 * @brief Pixels per meter message for X axis.
 */
const std::string pixels_per_meter_x_message = "Pixels per meter (X
axis): ";

/**
 * @brief Pixels per meter message for Y axis.
 */
const std::string pixels_per_meter_y_message = "Pixels per meter (Y
axis): ";

/**
 * @brief Colors used message.
 */
const std::string colors_used_message = "Colors used: ";

/**
 * @brief Important colors message.
 */
const std::string important_colors_message = "Important colors: ";

/**
 * @brief Error message for the usage of two incompatible functions.
 */
const std::string double_function_use_err = "Incompatible functions";

/**
 * @brief Error message for too many arguments provided.
 */
const std::string too_many_args_err = "Too many arguments";
```

operation_params.hpp

```
/**
 * @file operation_params.h
 * @brief Заголовочный файл, содержащий определения структур и классов.
 */

#pragma once

#include "structures.hpp"
#include <getopt.h>
#include <vector>

#include <cstring>
#include <functional>
#include <map>
#include <sstream>
#include <stdexcept>
/**
 * @brief Парсинг командной строки и создание объекта Operations.
 *
 * @param argc Количество аргументов командной строки.
 * @param argv Массив строк, содержащих аргументы командной строки.
 * @return Operations объект, содержащий параметры операции.
 */
Operations parse_command_line(int argc, char* argv[]);

/**
 * @brief Парсинг строки с числами, разделенными пробелами.
 *
 * @param str Строка, содержащая числа, разделенные пробелами.
 * @return Вектор целых чисел, полученных из строки.
 */
std::vector<int> parse_values(const std::string& str);

/**
 * @brief Парсинг строки с RGB-значением цвета.
 *
 * @param str Строка, содержащая RGB-значение цвета в формате "R,G,B".
 * @return Структура RGB, представляющая цвет.
```

```
 */
RGB parse_RGB(const std::string& str);

/**
 * @brief Отображение справки о возможных параметрах командной строки.
 */
void display_help();
```