**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.

# Cодержание

[Cодержание 5](#_Toc166513999)

[Введение 5](#_Toc166514000)

[Описание задачи и требований 6](#_Toc166514001)

[Описание архитектуры программы 8](#_Toc166514002)

[2.1 Модульная структура 8](#_Toc166514003)

[2.2 Компиляция с использованием Makefile 9](#_Toc166514004)

[2.3 Основная часть программы (main.cpp) 9](#_Toc166514005)

[2.4 Заголовочные файлы 9](#_Toc166514006)

[3. Описание структур данных (*BMPHeader*, *RGB и другие*) 9](#_Toc166514007)

[3.1 Структура *BMPHeader* 9](#_Toc166514008)

[3.2 BMPHeader (Заголовок BMP-файла): 10](#_Toc166514009)

[3.3 RGB (Цвет в формате RGB): 11](#_Toc166514010)

[3.4 Coordinate (Координаты точки на плоскости): 11](#_Toc166514011)

[3.5 Operations (Параметры операции над изображением): 11](#_Toc166514012)

[Полученные результаты 12](#_Toc166514013)

[Функциональность: 12](#_Toc166514014)

[Гибкость и управление ошибками: 12](#_Toc166514015)

[Вывод сообщений: 12](#_Toc166514016)

[Стабильность: 12](#_Toc166514017)

[Тестирование: 13](#_Toc166514018)

[6.Заключение 13](#_Toc166514019)

[Основные характеристики проекта: 13](#_Toc166514020)

[Приложения 14](#_Toc166514021)

[Приложения A 14](#_Toc166514022)

# 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 A

**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();