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 _____ Атоян М. А.

Преподаватель _____ Государкин Я.С.

Санкт-Петербург
2024

ЗАДАНИЕ
на курсовую работу

Студент Атоян М. А.

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: 23.05.2024

Дата защиты реферата: 23.05.2024

Студент _____ Атоян М. А.

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. Рисование правильного шестиугольника с обводкой и заполнением.
2. Копирование определенной области изображения в другую область.
3. Замену всех пикселей определенного цвета на другой цвет.
4. Рисование трёх различных видов орнамента изображения.

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 рисование шестиугольника, копирование, замену цвета и рисование орнамента.
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 иметь следующую функции по обработке изображений:

1. Рисование правильного шестиугольника. Флаг для выполнения данной операции: `--hexagon`. Шестиугольник определяется:
   - координатами его центра и радиусом в который он вписан. Флаги `--center` и `--radius`. Значение флаг `--center` задаётся в формате `x.y`, где x – координата по оси x, y – координата по оси y. Флаг `--radius` На вход принимает число больше 0
   - толщиной линий. Флаг `--thickness`. На вход принимает число больше 0
   - цветом линий. Флаг `--color` (цвет задаётся строкой `rrr.ggg.bbb`, где rrr/ggg/bbb – числа, задающие цветовую компоненту. пример `--color 255.0.0` задаёт красный цвет)
   - шестиугольник может быть залит или нет. Флаг `--fill`. Работает как бинарное значение: флага нет – false , флаг есть – true.
   - цветом которым залит шестиугольник, если пользователем выбран залитый. Флаг `--fill_color` (работает аналогично флагу `--color`)

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. Сделать рамку в виде узора. Флаг для выполнения данной операции: `--ornament`. Рамка определяется:
   - Узором. Флаг `--pattern`. Обязательные значения: rectangle и circle, semicircles. Также можно добавить свои узоры (красивый узор можно получить используя фракталы). Подробнее здесь: https://se.moevm.info/doku.php/courses:programming:cw_spring_ornament
   - Цветом. Флаг `--color` (цвет задаётся строкой `rrr.ggg.bbb`, где rrr/ggg/bbb – числа, задающие цветовую компоненту. пример `--color 255.0.0` задаёт красный цвет)
   - Шириной. Флаг `--thickness`. На вход принимает число больше 0
   - Количеством. Флаг `--count`. На вход принимает число больше 0
   - При необходимости можно добавить дополнительные флаги для необозначенных узоров

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:

- *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

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

#define IMG_DIR ""

int main(int argc, char *argv[])
{
    Logger::log(hello_message);

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

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

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

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

    // Рисование шестиугольника на изображении, если соответствующий флаг
установлен
    if (params.hexagon)
    {
        Logger::warn(hexagon_warning);
        bmp.hexagon(params.center, params.radius, params.thickness,
params.color, params.fill, params.fill_color);
        Logger::log(success_message);
    }

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

```
    }

    // Рисование орнамента для изображения, если соответствующий флаг
установлен
    if (params.ornament)
    {
        Logger::warn(ornamenet_warning);
        bmp.ornament(params.pattern, params.color, params.thickness,
params.count);
        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

```
#include "logger.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

```
#include <string>

#pragma pack(push, 1)
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)

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) {}
};

struct Coordinate
{
    int32_t x; /**< Координата x. */
    int32_t y; /**< Координата y. */
};

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 hexagon = false;                 /**< Флаг отражения
изображения. */
    Coordinate center;
    int radius = 0;
    int thickness = 0;
    RGB color;
    bool fill = false;
    RGB fill_color;
    bool copy = false;                   /**< Флаг копирования выделенной
области. */
    Coordinate dest_left_up;            /**< Левая верхняя точка для
вставки скопированной области. */
    Coordinate left_up;
    Coordinate right_down;
    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 ornament = false;                  /**< Флаг разделения
изображения на части. */
    std::string pattern;
    int count = 0;

    Operations()
    : input_file()
    , output_file("out.bmp")
    , info(false)
    , hexagon(false)
    , center()
    , radius(0)
    , thickness(0)
    , color()
    , fill(false)
    , fill_color()
    , copy(false)
    , dest_left_up()
    , left_up()
    , right_down()
    , color_replace(false)
    , old_color()
    , new_color()
    , ornament(false)
    , pattern()
    , count(0)
    {
```

```
    }
};
```

bmp.cpp

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

BMP::BMP(const std::string &fileName) : header(), pixelData()
{
    std::ifstream file(fileName, std::ios::binary);

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

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

    if (!validateHeader())
    {
        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;

    pixelData.resize(imageSize);

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

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

    file.close();
}

void BMP::getInfo() 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));
}

bool BMP::validateHeader() 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::isValid() const
```

```
{
    return !pixelData.empty();
}

RGB BMP::getColor(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(pixelData[index + 2], pixelData[index + 1],
pixelData[index]);
}

void BMP::setColor(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);

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

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;
    }

    int rowSize = ((header.width * header.bitsPerPixel + 31) / 32) * 4;

    int imageSize = rowSize * header.height;

    header.fileSize = header.dataOffset + imageSize;
    header.imageSize = imageSize;
```

```
    file.write(reinterpret_cast<const char *>(&header), sizeof(header));

    for (int y = 0; y < header.height; ++y)
    {
        file.write(reinterpret_cast<const char *>(pixelData.data() + y *
rowSize),
                   rowSize);
    }

    file.close();
}

void BMP::colorReplace(const RGB &color_replace_old_color, const RGB
&color_replace_new_color)
{
    for (int y = 0; y < header.height; y++)
    {
        for (int x = 0; x < header.width; x++)
        {
            RGB current_color = getColor(x, y);
            if (current_color.red == color_replace_old_color.red &&
                current_color.green == color_replace_old_color.green &&
                current_color.blue == color_replace_old_color.blue)
            {
                setColor(x, y, color_replace_new_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++) {
setColor(dest_left_up.x + (x - src_x_min), dest_left_up.y + (y -
src_y_min), getColor(x, y)); }
    }
}

void BMP::drawRectangle(const Coordinate left_top, const Coordinate
right_bottom, const RGB color)
{
    for (int x = left_top.x; x <= right_bottom.x; x++)
```

```
    {
        setColor(x, left_top.y, color);
        setColor(x, right_bottom.y, color);
    }

    for (int y = left_top.y; y <= right_bottom.y; y++)
    {
        setColor(left_top.x, y, color);
        setColor(right_bottom.x, y, color);
    }
}

void BMP::drawCircle(const Coordinate center, const int radius, const int
thickness, const RGB color)
{
    for (int x = center.x - radius - thickness; x <= center.x + radius +
thickness; x++)
    {
        for (int y = center.y - radius - thickness; y <= center.y +
radius + thickness; y++)
        {
            if (pow(x - center.x, 2) + pow((y - center.y), 2) >=
pow(radius, 2) &&
                pow(x - center.x, 2) + pow((y - center.y), 2) <
pow(radius + thickness, 2))
            {
                setColor(x, y, color);
            }
        }
    }

}

void BMP::ornament(const std::string pattern, const RGB color, const int
thickness = 0, const int count = 0)
{
    if (pattern == "circle")
    {
        struct Coordinate center = {header.width / 2, header.height / 2};
        int radius = std::min(header.height, header.width) / 2;

        for (int x = 0; x <= header.width; x++)
            for (int y = 0; y <= header.height; y++)
                if (pow((center.y - y), 2) + pow((center.x - x), 2) >
pow(radius, 2))
                    setColor(x, y, color);

        return;
    }
```

```
    if (thickness <= 0 || count <= 0)
        Logger::exit(ERR_INVALID_ARGUMENT, invalid_ornament_parameters);

    if (pattern == "rectangle")
    {
        struct Coordinate left_top = {0, 0};
        struct Coordinate right_bottom = {header.width - 1, header.height
- 1};

        for (int cnt = 0; cnt < count; cnt++)
        {

            if ((left_top.x + thickness >= right_bottom.x) || (left_top.y
+ thickness >= right_bottom.y))
            {
                Logger::warn(rectangle_overflow_warning);
                return;
            }

            for (int layer = 0; layer < thickness; layer++)
            {
                drawRectangle(left_top, right_bottom, color);
                left_top.x += 1;
                left_top.y += 1;
                right_bottom.x -= 1;
                right_bottom.y -= 1;
            }

            left_top.x += thickness;
            left_top.y += thickness;
            right_bottom.x -= thickness;
            right_bottom.y -= thickness;
        }
        return;
    }

    if (pattern == "semicircles")
    {
        int horizontal_radius = ceil(float(header.width) / count / 2) -
thickness / 2;
        int vertical_radius = ceil(float(header.height) / count / 2) -
thickness / 2;

        for (int oXcenter = horizontal_radius + thickness / 2; oXcenter -
horizontal_radius < header.width; oXcenter += horizontal_radius * 2 +
thickness)
        {
            struct Coordinate center = {oXcenter, 0};
            drawCircle(center, horizontal_radius, thickness, color);
            center.y = header.height;
```

```
            drawCircle(center, horizontal_radius, thickness, color);
        }

        for (int oYcenter = vertical_radius + thickness / 2; oYcenter -
vertical_radius < header.height; oYcenter += vertical_radius * 2 +
thickness)
        {
            struct Coordinate center = {0, oYcenter};
            drawCircle(center, vertical_radius, thickness, color);
            center.x = header.width;
            drawCircle(center, vertical_radius, thickness, color);
        }
        return;
    }

    Logger::exit(ERR_INVALID_ARGUMENT, invalid_ornament_pattern);
}

bool isInHexagonArea(const Coordinate center, int x, int y, int radius)
{
    // Просто конченная математическая формула для проверки на вхождение
точки в область шестиугольника.
    return abs(float(x) + float(radius) / 2 - center.x) + abs(float(x) -
float(radius) / 2 - center.x) + float(2 * abs(y - center.y)) / sqrt(3) <
2 * radius + 1;
}

void BMP::drawHexagon(const Coordinate center, const int radius, const
RGB color)
{
    for (int x = center.x - radius; x <= center.x; x++)
    {
        for (int y = center.y - radius; y <= center.y; y++)
        {


            if (isInHexagonArea(center, x, y, radius))
            {
                setColor(x, y, color);
                setColor(x, 2*center.y - y, color);
                setColor(2*center.x - x, y, color);
                setColor(2*center.x - x, 2*center.y - y, color);
            }
        }
    }
}

void BMP::drawThickLine(int x0, int y0, int x1, int y1, int thickness,
const RGB color)
{
```

```
    int dx = std::abs(x1 - x0);
    int dy = std::abs(y1 - y0);
    int sx = (x0 < x1) ? 1 : -1;
    int sy = (y0 < y1) ? 1 : -1;
    int err = dx - dy;
    int radius = thickness / 2;
    int radiusSquared = radius * radius;
    int x, y;

    int iSquared, jSquared;
    for (x = -radius; x <= radius; ++x)
    {
        iSquared = x * x;
        for (y = -radius; y <= radius; ++y)
        {
            jSquared = y * y;
            if (iSquared + jSquared <= radiusSquared) { setColor(x0 + x,
y0 + y, color); }
        }
    }

    while (x0 != x1 || y0 != y1)
    {
        int e2 = 2 * err;
        if (e2 > -dy)
        {
            err -= dy;
            x0 += sx;
        }
        if (e2 < dx)
        {
            err += dx;
            y0 += sy;
        }
        for (x = -radius; x <= radius; ++x)
        {
            iSquared = x * x;
            for (y = -radius; y <= radius; ++y)
            {
                jSquared = y * y;
                if (iSquared + jSquared <= radiusSquared) { setColor(x0 +
x, y0 + y, color); }
            }
        }
    }
}

void BMP::hexagon(const Coordinate center, const int radius, const int
thickness, const RGB color,
                  const bool fill, const RGB fill_color)
```

```
{
    if (thickness <= 0 || radius <= 0)
        Logger::exit(ERR_INVALID_ARGUMENT, invalid_hexagon_parameters);

    if (fill){
        drawHexagon(center, radius, fill_color);
    }

    drawThickLine(center.x - radius, center.y, center.x - radius/2,
center.y - radius*sqrt(3)/2, thickness, color);
    drawThickLine(center.x - radius/2, center.y - radius*sqrt(3)/2,
center.x + radius/2, center.y - radius*sqrt(3)/2,  thickness, color);
    drawThickLine(center.x + radius/2, center.y - radius*sqrt(3)/2,
center.x + radius, center.y, thickness, color);
    drawThickLine(center.x + radius, center.y, center.x + radius/2,
center.y + radius*sqrt(3)/2 + 1, thickness, color);
    drawThickLine(center.x + radius/2, center.y + radius*sqrt(3)/2 + 1,
center.x - radius/2, center.y + radius*sqrt(3)/2 + 1, thickness, color);
    drawThickLine(center.x - radius/2, center.y + radius*sqrt(3)/2 + 1,
center.x - radius, center.y, thickness, color);
}
```

Operation_params.cpp

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

std::vector<int> parseValues(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 parseRGB(const std::string& str)
{
    std::vector<int> values = parseValues(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 = parseValues(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 displayHelp()
{
    Logger::log(help_usage_description);
    Logger::log(help_usage_start);
    //=====================================
    Logger::log(copy_option_description);
    Logger::log(left_up_option_description);
    Logger::log(right_down_option_description);
    Logger::log(dest_left_up_option_description);
    //=====================================
    Logger::log(color_replace_option_description);
    Logger::log(old_color_option_description);
    Logger::log(new_color_option_description);
    //=====================================
    Logger::log(hexagon_option_description);
    Logger::log(fill_option_description);
    Logger::log(fill_color_option_description);
    //=====================================
    Logger::log(ornament_option_description);
    Logger::log(color_option_description);
    Logger::log(radius_option_description);
    Logger::log(thickness_option_description);
    Logger::log(output_option_description);
    Logger::log(input_option_description);
    Logger::log(help_option_description);
}
```

```
Operations parseCommandLine(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)");
              displayHelp();
              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.hexagon = true; } },
        { 257, [&](const char* option_argument) { params.center =
parseCoordinate(option_argument); } },
        { 258, [&](const char* option_argument) { params.radius =
parseValues(option_argument)[0]; } },
        { 259, [&](const char* option_argument) { params.thickness =
parseValues(option_argument)[0]; } },
        { 260, [&](const char* option_argument) { params.color =
parseRGB(option_argument); } },
        { 261, [&](const char*) { params.fill = true; } },
        { 262, [&](const char* option_argument) { params.fill_color =
parseRGB(option_argument); } },
        { 263, [&](const char*) { params.copy = true; } },
        { 264, [&](const char* option_argument) { params.dest_left_up =
parseCoordinate(option_argument); } },
        { 265, [&](const char* option_argument) { params.left_up =
parseCoordinate(option_argument); } },
        { 266, [&](const char* option_argument) { params.right_down =
parseCoordinate(option_argument); } },
        { 267, [&](const char*) { params.color_replace = true; } },
        { 268, [&](const char* option_argument) { params.old_color =
parseRGB(option_argument); } },
        { 269, [&](const char* option_argument) { params.new_color =
parseRGB(option_argument); } },
        { 270, [&](const char*) { params.ornament = true; } },
        { 271, [&](const char* option_argument) { params.pattern =
option_argument; } },
        { 272, [&](const char* option_argument) { params.count =
parseValues(option_argument)[0]; } },
        { 273, [&](const char*) { params.info = true; } },
```

```
        { 274, [&](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' }, { "hexagon", no_argument, nullptr, 256
}, { "center", required_argument, nullptr, 257 }, { "radius",
required_argument, nullptr, 258 }, { "thickness", required_argument,
nullptr, 259 }, { "color", required_argument, nullptr, 260 }, { "fill",
no_argument, nullptr, 261 }, { "fill_color", required_argument, nullptr,
262 }, { "copy", no_argument, nullptr, 263 }, { "dest_left_up",
required_argument, nullptr, 264 }, { "left_up", required_argument,
nullptr, 265 }, { "right_down", required_argument, nullptr, 266 }, {
"color_replace", no_argument, nullptr, 267 }, { "old_color",
required_argument, nullptr, 268 }, { "new_color", required_argument,
nullptr, 269 }, { "ornament", no_argument, nullptr, 270 }, { "pattern",
required_argument, nullptr, 271 }, { "count", required_argument, nullptr,
272 }, { "info", no_argument, nullptr, 273 }, { 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.fill && !params.hexagon) {
        Logger::warn(filling_a_nonexistent_hexagon_err);
        params.fill = false;
    }

    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

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

```
#include <cstring>
#include <fstream>
#include <math.h>

class BMP
{
private:
    RGB getColor(int x, int y) const;

    void setColor(int x, int y, const RGB &newColor);

    void drawHexagon(const Coordinate center, const int radius, const RGB
color);

    void drawCircle(const Coordinate center, const int radius, const int
thickness, const RGB color);

    void drawRectangle(const Coordinate left, const Coordinate right,
const RGB color);

    void drawThickLine(int x0, int y0, int x1, int y1, int thickness,
const RGB color);
public:

    void getInfo() const;

    BMP(const std::string &fileName);

    bool isValid() const;

    void save(const std::string &fileName);

    void hexagon(const Coordinate center, const int radius, const int
thickness, const RGB color,
                 const bool fill = false, const RGB fill_color = {0, 0,
0});

    void copy(const Coordinate &src_left_up, const Coordinate
&src_right_down,
              const Coordinate &dest_left_up);

    void colorReplace(const RGB &color_replace_old_color, const RGB
&color_replace_new_color);

    void ornament(const std::string pattern, const RGB colour, const int
thikness, const int count);

private:
    BMPHeader header;             ///< Заголовок BMP файла.
    bool validateHeader() const; ///< Проверка корректности заголовка.
```

```
    std::vector<char> pixelData; ///< Пиксельные данные изображения.
};
```

logger.hpp

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

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:
    Logger(bool enable_colors);

    static void set_colors_enabled(bool enableColors);

    template <typename Message>
    static void log(const Message &message, Color color = Color::GREEN,
std::ostream &stream = std::cout);

    static void warn(const std::string &message, std::ostream &stream =
std::cout);

    static void error(const std::string &message, std::ostream &stream =
std::cerr);

    static void exit(int exitCode, const std::string &exitMessage = "",
std::ostream &stream = std::cerr);
};
```

messages.hpp

```
#include <string>

#define ERR_FILE_NOT_FOUND 40
```

```
#define ERR_INCORRECT_FILE_FORMAT 41

#define ERR_FILE_WRITE_ERROR 42

#define ERR_INVALID_ARGUMENT 43

#define ERR_INSUFFICIENT_ARGUMENTS 45

const std::string hello_message = "Course work for option 5.7, created by
Atoyan Mikhail.";

const std::string invalid_bmp_message = "Invalid bmp file!";

const std::string same_input_output_message = "Input file is the same as
output file!";

const std::string success_message = "Success!";

const std::string invalid_signature_error = "Invalid BMP file signature";

const std::string invalid_dimensions_error = "Invalid BMP dimensions";

const std::string invalid_bpp_warning = "Invalid BMP bits per pixel,
output image may be incorrect";

const std::string unsupported_compression_error = "Unsupported BMP
compression type";

const std::string invalid_header_error = "BMP file header is invalid: ";

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

const std::string invalid_copy_region = "Invalid copy region or
destination parameters";

const std::string copy_exceeds_bounds_error = "Copying region exceeds
destination image boundaries.";

const std::string signature_message = "Signature: ";

const std::string file_size_message = "File size: ";

const std::string data_offset_message = "Data offset: ";

const std::string header_size_message = "Header size: ";

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

const std::string bits_per_pixel_message = "Bits per pixel: ";
```

```
const std::string compression_message = "Compression: ";

const std::string image_size_message = "Image size: ";

const std::string pixels_per_meter_x_message = "Pixels per meter (X
axis): ";

const std::string pixels_per_meter_y_message = "Pixels per meter (Y
axis): ";

const std::string colors_used_message = "Colors used: ";

const std::string important_colors_message = "Important colors: ";

const std::string invalid_ornament_pattern = "Invalid ornament pattern";

const std::string rectangle_overflow_warning = "The number of possible
rectangles is exceeded";

const std::string invalid_ornament_parameters = "Ornament parametrs are
invaild";

const std::string invalid_hexagon_parameters = "Hexagon parametrs are
invaild";

const std::string invalid_argument_error = "Invalid argument for ";

const std::string invalid_color_format_error = "Invalid color format";

const std::string invalid_color_range_error = "Color out of range [0-255]
got: ";

const std::string filling_a_nonexistent_hexagon_err = "Tried to fill a
non-existent hexagon. Operation aborted";

const std::string too_many_args_err = "Too many arguments";

const std::string hexagon_warning = "~Hexagon operation is requested";

const std::string color_replace_warning = "~Color replace operation is
requested";

const std::string ornamenet_warning = "~Ornament operation is requested";

const std::string image_copy_warning = "~Image copy operation is
requested";

const std::string help_usage_description = "Usage: program_name [options]
filename";
```

```
const std::string help_usage_start = "Options: ";

const std::string hexagon_option_description = "  --
hexagon              Hexagon operation";

const std::string left_up_option_description = "  --left_up
<x.y>        Coordinates of left-up corner";

const std::string right_down_option_description = "  --right_down
<x.y>     Coordinates of right-down corner";

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

const std::string old_color_option_description = "  --old_color
<r.g.b>    Old color to replace";

const std::string new_color_option_description = "  --new_color
<r.g.b>    New color to replace with";

const std::string color_option_description = "  --color
<r.g.b>        Color of hexagon/ornament";

const std::string copy_option_description = "  --
copy                 Copy operation";

const std::string color_replace_option_description = "  --
color_replace        Color replace operation";

const std::string ornament_option_description = "  --
split                Ornament operation";

const std::string radius_option_description = "  --radius
<value>       Radius of hexagon";

const std::string thickness_option_description = "  --thickness
<value>    Thickness of hexagon/ornament";

const std::string output_option_description = "  -o, --output
<file>    Output file";

const std::string input_option_description = "  -i, --input
<file>     Input file";

const std::string help_option_description = "  -h, --
help             Display this information";

const std::string fill_option_description = "  --
fill                 Filling hexagon with color";
```

```
const std::string fill_color_option_description = "  --
fill_color           Color of hexagon's insides";
```

operation_params.hpp

```
#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 parseCommandLine(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> parseValues(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 parseRGB(const std::string& str);

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