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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Обработка BMP изображения

Студент гр. 3343 _____ Поддубный В.А.

Преподаватель _____ Государкин Я.С.

Санкт-Петербург

2024

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студент: Поддубный Владислав

Группа: 3343

Тема: Обработка BMP изображения

Условия задания (Вариант 5.4):

Программа должна иметь следующую функции по обработке изображений:

1. Инверсия цвета в заданной области. Флаг для выполнения данной операции: `--inverse`. Функционал определяется
    - Координатами левого верхнего угла области. Флаг `--left_up`, значение задаётся в формате `left.up`, где left – координата по x, up – координата по y
    - Координатами правого нижнего угла области. Флаг `--right_down`, значение задаётся в формате `right.down`, где right – координата по x, down – координата по y
2. Преобразовать в Ч/Б изображение (формулу можно посмотреть на wikipedia). Флаг для выполнения данной операции: `--gray`. Функционал определяется
    - Координатами левого верхнего угла области. Флаг `--left_up`, значение задаётся в формате `left.up`, где left – координата по x, up – координата по y
    - Координатами правого нижнего угла области. Флаг `--right_down`, значение задаётся в формате `right.down`, где right – координата по x, down – координата по y
3. Изменение размера изображения с его обрезкой или расширением фона. Флаг для выполнения данной операции: `--resize`. Функционал определяется:

- Количеством изменения пикселей с определенной стороны в формате: `--<side> <change>`, где `<side>` может принимать значения left (с левой стороны изменение), right (с правой стороны), above (с верхней стороны), below (с нижней стороны); `<side>` является числом: положительное означает расширение, отрицательное означает обрезку. Например, следующие флаги `--resize --left 100 --above -100 --below 30 --right -20` означает, что нужно расширить изображение слева на 100 пикселей и снизу на 30, и обрезать изображение сверху на 100 пикселей и справа на 20 пикселей.
- Цветом фона при расширении изображения. Флаг `--color` (цвет задаётся строкой `rrr.ggg.bbb`, где rrr/ggg/bbb – числа, задающие цветовую компоненту. пример `--color 255.0.0` задаёт красный цвет)

4. Рисование отрезка. Флаг для выполнения данной операции: `--line`. Отрезок определяется:
   - координатами начала. Флаг `--start`, значение задаётся в формате `x.y`, где x – координата по x, y – координата по y
   - координатами конца. Флаг `--end` (аналогично флагу `--start`)
   - 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)
   - толщиной. Флаг `--thickness`. На вход принимает число больше 0

Дата выдачи задания: 18.03.2024

Дата сдачи реферата: 15.05.2024

Дата защиты реферата: 15.05.2024

АННОТАЦИЯ

В ходе курсовой работы реализована программа, осуществляющая обработку BMP изображения. Для взаимодействия с программой реализован интерфейс командной строки (CLI). Программа реализует следующие функции: рисование линии, инвертирование цвета в рамке, чб цвета в рамке, ресайз изображения. Сборка проекта осуществляется с помощью утилиты make.

ВВЕДЕНИЕ

Цель работы:

Разработать интерактивное консольное приложение для обработки изображений в формате BMP, которое предоставляет следующие функции:

• Считывание и запись BMP-изображений.

• Изменение изображения

• Визуализация обработанного изображения.

1. ОПИСАНИЕ СТРУКТУРЫ ПРОГРАММЫ

Программа написана на языке C++ и использует объектно-ориентированный подход. Код разделен на несколько файлов:

- **main.cpp**: Содержит точку входа в программу (main функцию) и обрабаты вает аргументы командной строки.
- **opts_reader.cpp / opts_reader.hpp**: Отвечает за парсинг аргументов коман дной строки, валидацию входных данных и формирование структуры Opt s, хранящей параметры обработки.
- **picture.cpp / picture.hpp**: Реализует класс Picture, представляющий BMP изображение. Он отвечает за загрузку, сохранение, доступ к пикселям и в ывод информации о изображении.
- **image_processor.cpp / image_processor.hpp**: Содержит класс ImageProcess or, отвечающий за реализацию функций обработки изображений: инверси и цвета, преобразование в оттенки серого, изменение размера и рисование линии.

ОПИСАНИЕ ФУНКЦИЙ

Класс OptsReader

- **getOpts(int argc, char* argv[])**: Парсит аргументы командной строки, валидирует входные данны е и возвращает структуру Opts, содержащую параметры обработки изобра жения.
- **getColor(std::string arg)**: Преобразует строку rrr.ggg.bbb в структуру Color, представляющую цвет.
- **getCoordinate(std::string arg)**: Преобразует строку x.y в структуру Coordinate, представляющую ко ординаты точки.

- **getValues(std::string arg)**: Вспомогательная функция, разбивает строку на отдельные значения, разделенные точкой.

Класс Picture

- **Picture(std::string path)**: Конструктор класса, загружающий BMP изображение из файла по указанному пути.
- **validate()**: Проверяет заголовок BMP файла на корректность.
- **setPixel(int x, int y, Color color)**: Устанавливает цвет пикселя с заданными координатами.
- **getPixel(int x, int y)**: Возвращает цвет пикселя с заданными координатами.
- **save(std::string fileName)**: Сохраняет изображение в файл с указанным именем.
- **setPixelToData(...)**: Устанавливает цвет пикселя в буфере данных изображения.
- **copyPixelToData(...)**: Копирует пиксель из старого буфера данных в новый.
- **printInfo()**: Выводит информацию о изображении (заголовок BMP файла).

Класс ImageProcessor

- **gray(Picture& picture, Coordinate leftUp, Coordinate rightDown)**: Преобразует заданную область изображения в оттенки серого.
- **inverse(Picture& picture, Coordinate leftUp, Coordinate rightDown)**: Инвертирует цвета в заданной области изображения.
- **resize(Picture& picture, int32_t left, int32_t above, int32_t right, int32_t below, Color color)**: Изменяет размер изображения, обрезая или расширяя фон.
- **line(Picture& picture, Coordinate start, Coordinate end, Color color, int32_t thickness)**: Рисует линию с заданными параметрами.

ТЕСТИРОВАНИЕ

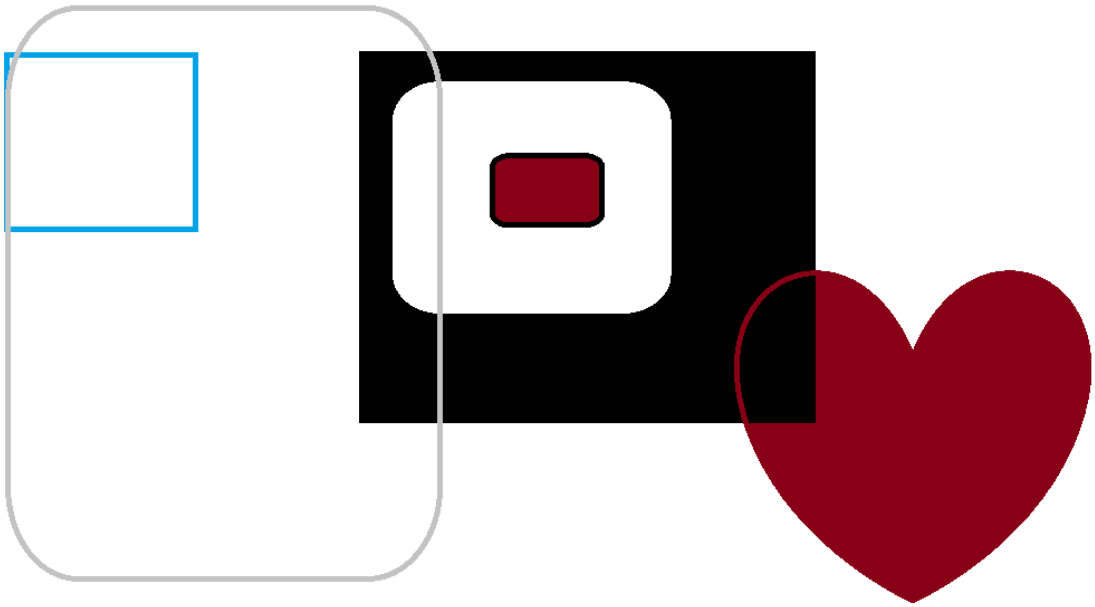

Рисунок 1 – изображение для тестирования

1. Тестирование функции *line*:

   Аргументы для запуска: --thickness 800 --start 0.0 --line --input ../24.bmp --end 700.700 --color 17.234.201 --output ./output.bmp

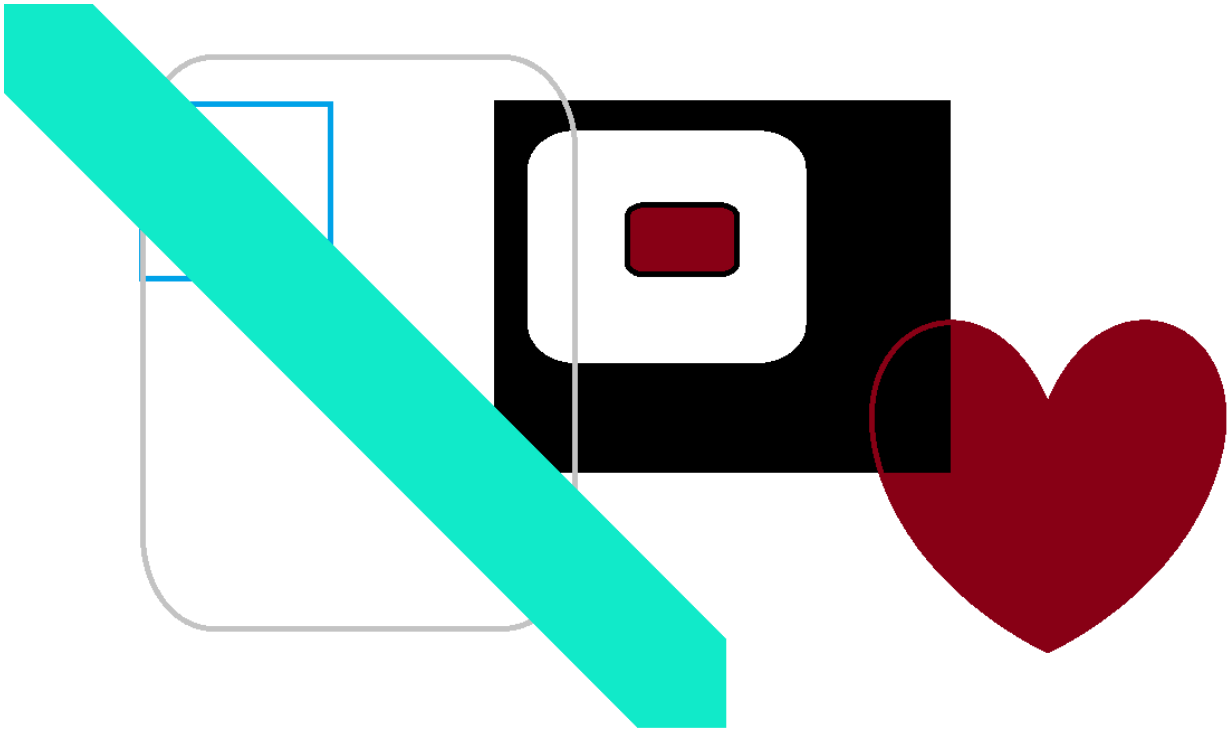

Рисунок 2 – результат работы функции *rect*

2. Тестирование функции *gray*:

   Аргументы для запуска: --left_up 0.0 --right_down 2000.3128 --gray ../24.bmp

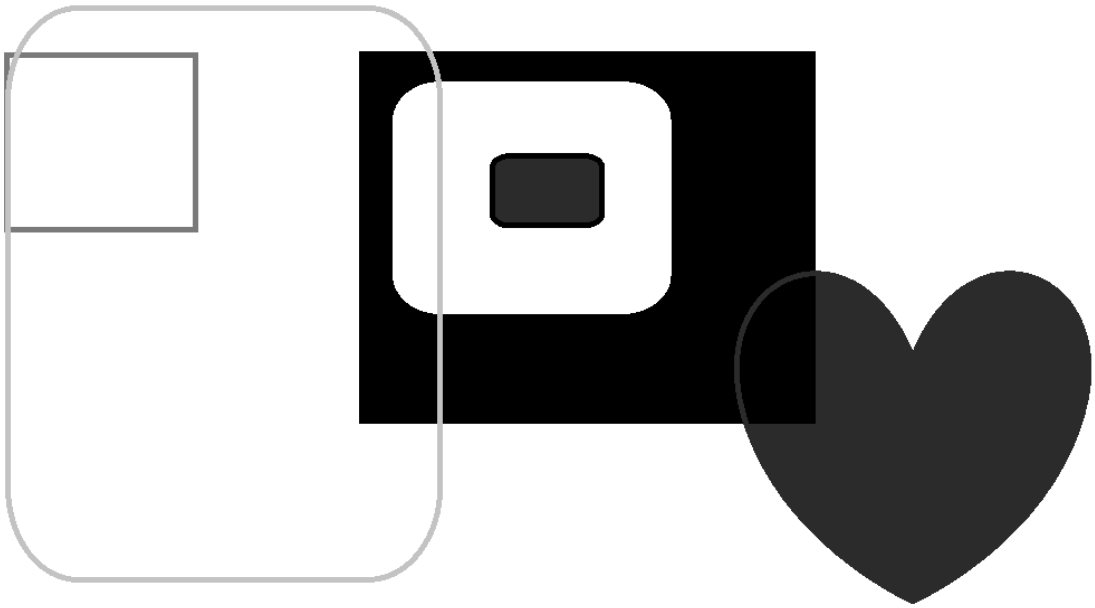

Рисунок 3 – результат работы функции *ornament (rectangle)*

3. Тестирование функции *resize*:

   Аргументы для запуска: --below -11 --resize --color 2.250.208 --output ./output.bmp --right -60 --above -55 --left 76 --input ../24.bmp

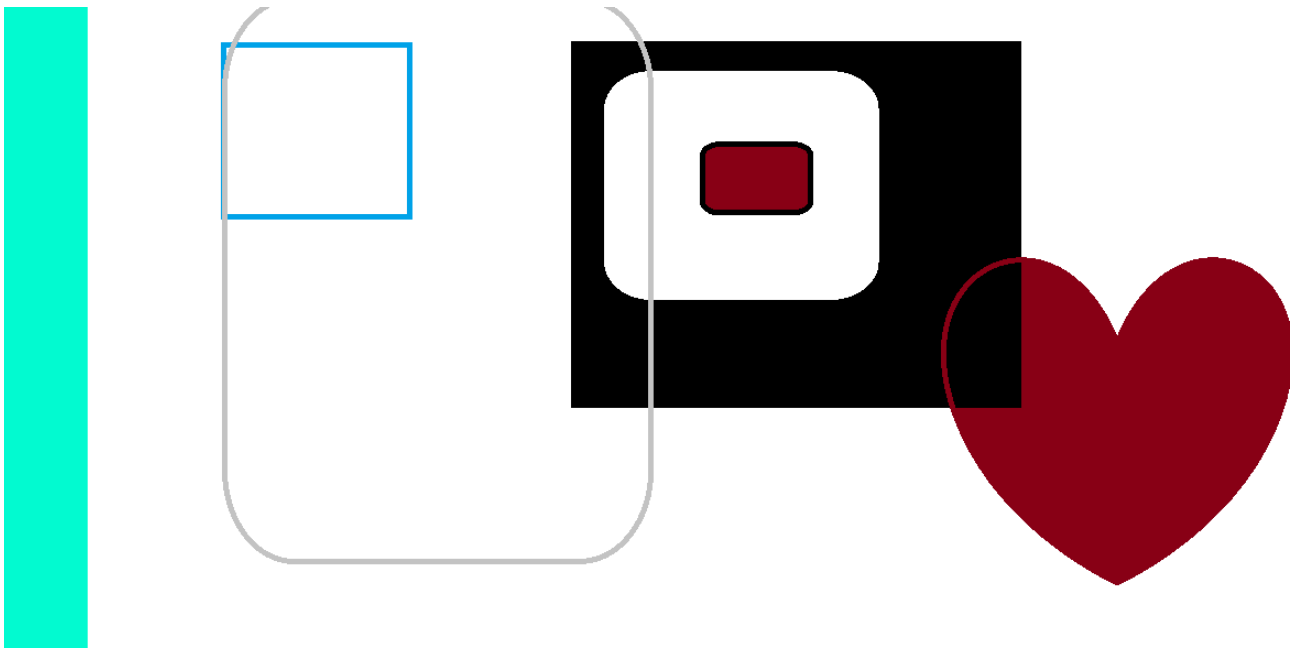

Рисунок 4 – результат работы функции *ornament (circle)*

4. Тестирование функции *inverse*:

   Аргументы для запуска: --inverse --left_up 00.0 --right_down 2000.7000 ../24.bmp

Рисунок 5 – результат работы функции *inverse*

ЗАКЛЮЧЕНИЕ

В рамках данной курсовой работы была разработана программа на языке программирования C для обработки изображений в формате BMP. Программа предоставляет набор функций, которые могут быть выбраны пользователем через командную строку. Сборка программы выполняется с помощью утилиты make. После сборки программа запускается из командной строки, где пользователь может выбрать одну из поддерживаемых функций для обработки изображения.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

image_processor.cpp

```
#include "../include/image_processor.hpp"

void ImageProcessor::inverse(Picture &picture, Coordinate leftUp, Coordinate
rightDown) {
    std::vector<uint8_t> data =  picture.data;

    for (int y = leftUp.y; y <= rightDown.y; ++y) {
        for (int x = leftUp.x; x <= rightDown.x; ++x) {
            Color pixel = picture.getPixel(x, y);
            pixel.blue = 255 - pixel.blue;
            pixel.red = 255 - pixel.red;
            pixel.green = 255 - pixel.green;
            picture.setPixel(x, y, pixel);
        }
    }
}

void ImageProcessor::gray(Picture &picture, Coordinate leftUp, Coordinate
rightDown){
    std::vector<uint8_t> data =  picture.data;

    for (int y = leftUp.y; y <= rightDown.y; ++y) {
        for (int x = leftUp.x; x <= rightDown.x; ++x) {
            Color pixel = picture.getPixel(x, y);
            uint8_t grayValue = std::round(0.299 * pixel.red + 0.587 * pixel.green +
0.114 * pixel.blue);
```

```
            picture.setPixel(x, y, Color(grayValue, grayValue, grayValue));
        }
    }
}

void ImageProcessor::resize(Picture &picture, int32_t left, int32_t above, int32_t
right, int32_t below, Color color){
    std::vector<uint8_t> data =  picture.data;
    int32_t oldHeight = picture.bmpHeader.height;
    int32_t oldWidth = picture.bmpHeader.width;
    int32_t newHeight = oldHeight + above + below;
    int32_t newWidth = oldWidth + right + left;

    uint32_t bytesPerPixel = picture.bmpHeader.bitsPerPixel / 8;
    uint32_t rowSize = ((newWidth * bytesPerPixel + 3) / 4) * 4;
    uint32_t imageSize = rowSize * newHeight;

    int32_t leftUpY = oldHeight + below;
    int32_t leftUpX = 0 + -left;
    std::vector<uint8_t> newData(imageSize);

    int oldY = leftUpY - 1;
    for (int y = newHeight-1; y >= 0; --y) {
        int oldX = leftUpX;
        for (int x = 0; x <= newWidth; ++x) {
            if (!(oldX <= oldWidth && oldY <= oldHeight && oldY >= 0 &&
oldX >=0)) {
                picture.setPixelToData(newData, x, y, color, newWidth, newHeight);
            } else {
```

```
                picture.copyPixelToData(newData, oldX, oldY, x, y, newWidth,
newHeight);
            }
        oldX++;
    }
    oldY--;
}

picture.bmpHeader.height = newHeight;
picture.bmpHeader.width = newWidth;
picture.bmpHeader.imageSize = imageSize;
picture.data = newData;
picture.bmpHeader.fileSize = picture.bmpHeader.dataOffset +
picture.bmpHeader.imageSize;
}

void ImageProcessor::line(Picture &picture, Coordinate startPoint, Coordinate
endPoint, Color color, int32_t lineThickness) {
    int deltaX = abs(endPoint.x - startPoint.x);
    int deltaY = abs(endPoint.y - startPoint.y);
    int stepX = (startPoint.x < endPoint.x) ? 1 : -1;
    int stepY = (startPoint.y < endPoint.y) ? 1 : -1;
    int error = deltaX - deltaY;

    while (true) {
        int rectX = startPoint.x - lineThickness / 2;
        int rectY = startPoint.y - lineThickness / 2;
        int rectWidth = lineThickness;
        int rectHeight = lineThickness;
        if (rectX >= 0 && rectX + rectWidth < picture.bmpHeader.width &&
```

```
            rectY >= 0 && rectY + rectHeight < picture.bmpHeader.height) {
            for (int y = rectY; y < rectY + rectHeight; ++y) {
                for (int x = rectX; x < rectX + rectWidth; ++x) {
                    picture.setPixel(x, y, color);
                }
            }
        }

        if (startPoint.x == endPoint.x && startPoint.y == endPoint.y) break;

        int doubleError = 2 * error;
        if (doubleError > -deltaY) {
            error -= deltaY;
            startPoint.x += stepX;
        }
        if (doubleError < deltaX) {
            error += deltaX;
            startPoint.y += stepY;
        }
    }

}
```

image_processor.hpp

```
#pragma once

#include <iostream>
#include <cmath>
#include "opts_reader.hpp"
#include "picture.hpp"
```

```
class ImageProcessor{
public:
    static void gray(Picture &picture, Coordinate leftUp, Coordinate rightDown);
    static void inverse(Picture &picture, Coordinate leftUp, Coordinate rightDown);
    static void resize(Picture &picture, int32_t left, int32_t above, int32_t right, int32_t
below, Color color);
    static void line(Picture &picture, Coordinate start, Coordinate end, Color color,
int32_t thickness);
};
```

picture.cpp

```
#include "../include/picture.hpp"

Picture::Picture(std::string path) {
    std::ifstream file(path, std::ios::binary);
    if (!file.is_open()) throw std::runtime_error("File does not exist!");
    file.read(reinterpret_cast<char*>(&bmpHeader), sizeof(bmpHeader));
    validate();

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

    data.resize(imageSize);
    file.seekg(bmpHeader.dataOffset, std::ios_base::beg);
    file.read(reinterpret_cast<char*>(data.data()), imageSize);
    file.close();
    if (data.empty()) throw std::runtime_error("BMP data is empty!");
}
```

```
void Picture::validate() {
    if (std::strncmp(bmpHeader.signature, "BM", 2) != 0) throw
std::runtime_error("Wrong BMP signature!");
    if (bmpHeader.width <= 0 || bmpHeader.height <= 0) throw
std::runtime_error("Wrong image width or height!");
    if (bmpHeader.bitsPerPixel != 24) throw std::runtime_error("Wrong bits per
pixel!");
    if (bmpHeader.compression != 0) throw std::runtime_error("Does not support files
with compression!");
}

void Picture::setPixel(int x, int y, Color color) {
    if (x < 0 || x >= bmpHeader.width || y < 0 || y >= bmpHeader.height) return;

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

    data[index] = color.blue;
    data[index + 1] = color.green;
    data[index + 2] = color.red;
}



void Picture::copyPixelToData(std::vector<uint8_t> &newData, int oldX, int oldY,
int x, int y, int32_t width,
                              int32_t height) {
    if (x < 0 || x >= width || y < 0 || y >= height) return;
```

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

    Color color = getPixel(oldX, oldY);
    newData[index] = color.blue;
    newData[index + 1] = color.green;
    newData[index + 2] = color.red;
}

void Picture::setPixelToData(std::vector<uint8_t> &newData, int x, int y, Color
color, int32_t width, int32_t height) {
    if (x < 0 || x >= width || y < 0 || y >= height) return;

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

    newData[index] = color.blue;
    newData[index + 1] = color.green;
    newData[index + 2] = color.red;
}

Color Picture::getPixel(int x, int y) {
    if (x < 0 || x >= bmpHeader.width || y < 0 || y >= bmpHeader.height) return Color();

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

```
    return {data[index + 2], data[index + 1], data[index]};
}

void Picture::save(std::string fileName) {
    std::ofstream file(fileName, std::ios::binary);
    if (!file.is_open()) throw std::runtime_error("Error while saving file!");

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

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

void Picture::printInfo() {
    std::cout << "signature:\t" << std::hex << bmpHeader.signature << " (" << std::dec
<< bmpHeader.signature << ")"
              << std::endl;
    std::cout << "filesize:\t" << std::hex << bmpHeader.fileSize << " (" << std::dec <<
bmpHeader.fileSize << ")"
              << std::endl;
    std::cout << "reserved1:\t" << std::hex << bmpHeader.reserved1 << " (" <<
std::dec << bmpHeader.reserved1 << ")"
              << std::endl;
```

```
    std::cout << "reserved2:\t" << std::hex << bmpHeader.reserved2 << " (" <<
std::dec << bmpHeader.reserved2 << ")"
              << std::endl;
    std::cout << "pixelArrOffset:\t" << std::hex << bmpHeader.dataOffset << " (" <<
std::dec << bmpHeader.dataOffset
              << ")" << std::endl;
    std::cout << "headerSize:\t" << std::hex << bmpHeader.headerSize << " (" <<
std::dec << bmpHeader.headerSize << ")"
              << std::endl;
    std::cout << "width: \t" << std::hex << bmpHeader.width << " (" << std::dec <<
bmpHeader.width << ")" << std::endl;
    std::cout << "height: \t" << std::hex << bmpHeader.height << " (" << std::dec <<
bmpHeader.height << ")"
              << std::endl;
    std::cout << "planes: \t" << std::hex << bmpHeader.planes << " (" << std::dec <<
bmpHeader.planes << ")"
              << std::endl;
    std::cout << "bitsPerPixel:\t" << std::hex << bmpHeader.bitsPerPixel << " (" <<
std::dec << bmpHeader.bitsPerPixel
              << ")" << std::endl;
    std::cout << "compression:\t" << std::hex << bmpHeader.compression << " (" <<
std::dec << bmpHeader.compression
              << ")" << std::endl;
    std::cout << "imageSize:\t" << std::hex << bmpHeader.imageSize << " (" <<
std::dec << bmpHeader.imageSize << ")"
              << std::endl;
    std::cout << "xPixelsPerMeter:\t" << std::hex << bmpHeader.xPixelsPerMeter << "
(" << std::dec
              << bmpHeader.xPixelsPerMeter << ")" << std::endl;
```

```
    std::cout << "yPixelsPerMeter:\t" << std::hex << bmpHeader.yPixelsPerMeter << "
(" << std::dec
              << bmpHeader.yPixelsPerMeter << ")" << std::endl;
    std::cout << "colorsInColorTable:\t" << std::hex << bmpHeader.colorsUsed << " ("
<< std::dec << bmpHeader.colorsUsed
              << ")" << std::endl;
    std::cout << "importantColorCount:\t" << std::hex << bmpHeader.colorsImportant
<< " (" << std::dec
              << bmpHeader.colorsImportant << ")" << std::endl;
}
```

picture.hpp

```
//
// Created by rect on 09.05.2024.
//



#pragma once

#include "opts_reader.hpp"
#include <cstring>
#include <fstream>
#include <vector>

#pragma pack(push, 1)
struct BMPHeader
{
    char signature[2];
    uint32_t fileSize;
    uint16_t reserved1;
```

```
    uint16_t reserved2;
    uint32_t dataOffset;
    uint32_t headerSize;
    int32_t width;
    int32_t height;
    uint16_t planes;
    uint16_t bitsPerPixel;
    uint32_t compression;
    uint32_t imageSize;
    int32_t xPixelsPerMeter;
    int32_t yPixelsPerMeter;
    uint32_t colorsUsed;
    uint32_t colorsImportant;
};
#pragma pack(pop)

class Picture{
public:
    Picture(std::string path);
    BMPHeader bmpHeader;
    std::vector<uint8_t> data;
    void setPixel(int x, int y, Color color);
    Color getPixel(int x, int y);
    void save(std::string fileName);
    void setPixelToData(std::vector<uint8_t> &newData, int x, int y, Color color,
int32_t width, int32_t height);
    void copyPixelToData(std::vector<uint8_t> &newData, int oldX, int oldY, int x, int
y, int32_t width, int32_t height);
    void printInfo();
private:
```

```
    void validate();
};
```

opts_reader.cpp

```
#include "../include/opts_reader.hpp"

const option OptsReader::long_opts[] = {
        {"inverse", no_argument, nullptr, 0},
        {"left_up", required_argument, nullptr, 1},
        {"right_down", required_argument, nullptr, 2},
        {"gray", no_argument, nullptr, 3},
        {"resize", no_argument, nullptr, 4},
        {"left", required_argument, nullptr, 5},
        {"right", required_argument, nullptr, 6},
        {"above", required_argument, nullptr, 7},
        {"below", required_argument, nullptr, 8},
        {"color", required_argument, nullptr, 9},
        {"line", no_argument, nullptr, 10},
        {"start", required_argument, nullptr, 11},
        {"end", required_argument, nullptr, 12},
        {"thickness", required_argument, nullptr, 13},
        {"help", no_argument, nullptr, 'h'},
        {"info", no_argument, nullptr, 14},
        {"output", required_argument, nullptr, 'o'},
        {"input", required_argument, nullptr, 'i'},
};
```

```
std::vector<int32_t> OptsReader::getValues(std::string arg) {
    std::vector<int32_t> values;
    std::string token;
    std::stringstream ss(arg);
    while (std::getline(ss, token, '.'))
    {
        int32_t value = std::stoi(token);
        values.push_back(value);
    }
    return values;
}

Coordinate OptsReader::getCoordinate(std::string arg) {
    std::vector<int32_t> values = getValues(arg);
    if (values.size()!=2) throw std::runtime_error("One of coordinates have not enough
args!");
    return {values[0],values[1]};
}

Color OptsReader::getColor(std::string arg){
    std::vector<int32_t> values = getValues(arg);
    if (values.size() != 3) throw std::runtime_error("Wrong number of params for
color");
    for (int value : values){
        if (value < 0 || value > 255) throw std::runtime_error("Wrong color range!");
    }
    return { static_cast<uint8_t>(values[0]), static_cast<uint8_t>(values[1]),
static_cast<uint8_t>(values[2]) };
}
```

```
Opts OptsReader::getOpts(int argc, char **argv) {
    int opt;
    Opts opts{};

    while ((opt = getopt_long(argc, argv, "hi:o:", long_opts, nullptr)) != -1) {
        switch (opt) {
            case 0:
                opts.inverse = true;
                break;
            case 1:
                opts.left_up = getCoordinate(optarg);
                break;
            case 2:
                opts.right_down = getCoordinate(optarg);
                break;
            case 3:
                opts.gray = true;
                break;
            case 4:
                opts.resize = true;
                break;
            case 5:
                opts.left = std::stoi(optarg);
                break;
            case 6:
                opts.right = std::stoi(optarg);
                break;
            case 7:
                opts.above = std::stoi(optarg);
                break;
```

```
case 8:
    opts.below = std::stoi(optarg);
    break;
case 9:
    opts.color = getColor(optarg);
    break;
case 10:
    opts.line = true;
    break;
case 11:
    opts.start = getCoordinate(optarg);
    break;
case 12:
    opts.end = getCoordinate(optarg);
    break;
case 13:
    opts.thickness = std::stoi(optarg);
    break;
case 'h':
    opts.help = true;
    return opts;
case 14:
    opts.info = true;
    break;
case 'o':
    opts.output_file = optarg;
    break;
case 'i':
    opts.input_file = optarg;
    break;
```

```
            default:
                std::cerr << "Неизвестный аргумент: " << argv[optind - 1] << std::endl;
                std::exit(1);
        }
    }
    if (opts.input_file.empty()) {
        if (optind == argc - 1) {
            opts.input_file = argv[optind];
        }
    }
    if (opts.output_file.empty()) {
        opts.output_file = "./out.bmp";
    }
    if (opts.output_file == opts.input_file) throw std::runtime_error("Input file equals output file!");
    return opts;
}
```

opts_reader.hpp

```
#pragma once

#include <iostream>
#include <getopt.h>
#include <vector>
#include <sstream>

struct Coordinate{
    int32_t x;
    int32_t y;
    Coordinate(int x=0, int y=0){
        this->x=x;
```

```
        this->y=y;
    }
};

struct Color{
    uint8_t red;
    uint8_t green;
    uint8_t blue;
    Color(uint8_t red = 0, uint8_t green = 0, uint8_t blue = 0){
        this->red=red;
        this->green=green;
        this->blue=blue;
    }
};

struct Opts{
    std::string input_file;
    bool inverse;
    Coordinate left_up;
    Coordinate right_down;
    bool gray;
    bool resize;
    int32_t left;
    int32_t right;
    int32_t above;
    int32_t below;
    Color color;
    bool line;
    Coordinate start;
    Coordinate end;
```

```
    int32_t thickness;
    bool help;
    bool info;
    std::string output_file;
};

class OptsReader {
public:
    static Opts getOpts(int argc, char* argv[]);
private:
    static Color getColor(std::string arg);
    static Coordinate getCoordinate(std::string arg);
    static std::vector<int32_t> getValues(std::string arg);
    const static struct option long_opts[];
};
```

main.cpp

```
#include "../include/main.hpp"

int main(int argc, char *argv[]) {
    try {
        std::cout << "Course work for option 5.4, created by Vladislav Poddubnyi." <<
std::endl;
        Opts opts = OptsReader::getOpts(argc, argv);
        if (opts.help) {
            printHelp();
            return 0;
        }
        Picture picture(opts.input_file);
        if (opts.info) picture.printInfo();
        if (opts.gray) ImageProcessor::gray(picture, opts.left_up, opts.right_down);
```

```
        if (opts.inverse) ImageProcessor::inverse(picture, opts.left_up, opts.right_down);
        if (opts.resize) ImageProcessor::resize(picture, opts.left, opts.above, opts.right,
opts.below, opts.color);
        if (opts.line) ImageProcessor::line(picture, opts.start, opts.end, opts.color,
opts.thickness);

        picture.save(opts.output_file);
    } catch (std::invalid_argument &e){
        std::cerr << "Use only nums for flags with nums!" << std::endl;
        return 40;
    } catch (std::exception &e) {
        std::cerr << e.what() << std::endl;
        return 40;
    }
    return 0;
}

void printHelp() {
    std::cout << "Available flags:" << std::endl;

    std::cout << "--inverse: Invert colors within a specified region." << std::endl;
    std::cout << "\t--left_up x.y: Coordinates of the top-left corner (e.g., --left_up
10.20)." << std::endl;
    std::cout << "\t--right_down x.y: Coordinates of the bottom-right corner (e.g., --
right_down 50.80)." << std::endl;

    std::cout << "--gray: Convert the image to grayscale within a specified region." <<
std::endl;
    std::cout << "\t--left_up x.y: Coordinates of the top-left corner." << std::endl;
```

```
    std::cout << "\t--right_down x.y: Coordinates of the bottom-right corner." << std::endl;

    std::cout << "--resize: Resize the image by cropping or extending the background." << std::endl;
    std::cout << "\t--<side> <change>: Specify the change in pixels for each side (left, right, above, below)." << std::endl;
    std::cout << "\t\tPositive values for extension, negative for cropping (e.g., --left 100, --above -50)." << std::endl;
    std::cout << "\t--color rrr.ggg.bbb: Set the background color for extension (e.g., --color 255.255.255 for white)." << std::endl;

    std::cout << "--line: Draw a line segment." << std::endl;
    std::cout << "\t--start x.y: Coordinates of the starting point." << std::endl;
    std::cout << "\t--end x.y: Coordinates of the ending point." << std::endl;
    std::cout << "\t--color rrr.ggg.bbb: Set the color of the line." << std::endl;
    std::cout << "\t--thickness <value>: Set the thickness of the line (value > 0)." << std::endl;
}
```

main.hpp

```
#pragma once

#include "opts_reader.hpp"
#include "image_processor.hpp"
#include "picture.hpp"

void printHelp();
```