МИНОБРНАУКИ РОССИИ
Санкт-Петербургский государственный
электротехнический университет
«ЛЭТИ» им. В.И. Ульянова (Ленина)
Кафедра МОЭВМ

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

ТЕМА: СОЗДАНИЕ ПРОГРАММ В СИ

Студент гр. 3341 _____ Гребенюк В.А.

Преподаватель _____ Глазунов С.А.

Санкт-Петербург
2024

ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

Студент Гребенюк В.А.

Группа 3341

Тема работы: Обработка изображений

Дата выдачи задания: 18.03.2024

Дата сдачи реферата: 13.05.2024

Дата защиты реферата: 15.05.2024

Студент _____ Гребенюк В.А.

Преподаватель _____ Глазунов С.А.

Исходные данные

Вариант 24

Программа обязательно должна иметь CLI (опционально дополнительное использование GUI). Более подробно тут: http://se.moevm.info/doku.php/courses:programming:rules_extra_kurs

Программа должна реализовывать весь следующий функционал по обработке png-файла

Общие сведения

Формат картинки PNG (рекомендуем использовать библиотеку libpng)

без сжатия

файл может не соответствовать формату PNG, т.е. необходимо проверка на PNG формат. Если файл не соответствует формату PNG, то программа должна завершиться с соответствующей ошибкой.

обратите внимание на выравнивание; мусорные данные, если их необходимо дописать в файл для выравнивания, должны быть нулями.

все поля стандартных PNG заголовков в выходном файле должны иметь те же значения что и во входном (разумеется кроме тех, которые должны быть изменены).

Программа должна иметь следующую функции по обработке изображений:

(1) Рисование квадрата с диагоналями. Флаг для выполнения данной операции: `--squared_lines`. Квадрат определяется:

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

Размером стороны. Флаг `--side_size`. На вход принимает число больше 0

Толщиной линий. Флаг `--thickness`. На вход принимает число больше 0

Цветом линий. Флаг `--color` (цвет задаётся строкой `rrr.ggg.bbb`, где rrr/ggg/bbb – числа, задающие цветовую компоненту. пример `--color 255.0.0` задаёт красный цвет)

Может быть залит или нет (диагонали располагаются “поверх” заливки). Флаг `--fill`. Работает как бинарное значение: флага нет – false , флаг есть – true.

Цветом которым он залит, если пользователем выбран залитый. Флаг `--fill_color` (работает аналогично флагу `--color`)

(2) Фильтр rgb-компонент. Флаг для выполнения данной операции: `--rgbfilter`. Этот инструмент должен позволять для всего изображения либо установить в диапазоне от 0 до 255 значение заданной компоненты. Функционал определяется

Какую компоненту требуется изменить. Флаг `--component_name`. Возможные значения `red`, `green` и `blue`.

В какой значение ее требуется изменить. Флаг `--component_value`. Принимает значение в виде числа от 0 до 255

(3) Поворот изображения (части) на 90/180/270 градусов. Флаг для выполнения данной операции: `--rotate`. 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

Углом поворота. Флаг `--angle`, возможные значения: `90`, `180`, `270`

Все подзадачи, ввод/вывод должны быть реализованы в виде отдельной функции.

Содержание пояснительной записки:

«Аннотация», «Содержание», «Введение», «Ход выполнения работы», «Заключение», «Список использованных источников», «Пример работы программы», «Исходный код программы».

АННОТАЦИЯ

Курсовой проект по варианту 24 включает в себя разработку программы с CLI (и опционально GUI), способной обрабатывать PNG-изображения. Программа должна поддерживать работу с несжатыми PNG-файлами, проверять соответствие файла формату PNG и завершать работу с ошибкой в случае несоответствия. Важно обеспечить корректное выравнивание данных в файле, заполняя мусорные данные нулями. Все поля стандартных PNG-заголовков в выходном файле должны соответствовать значениям входного файла, за исключением тех, которые подлежат изменению.

Функционал программы включает:

Рисование квадрата с диагоналями (--squared_lines), с параметрами:

Координаты левого верхнего угла (--left_up).

Размер стороны (--side_size).

Толщина линий (--thickness).

Цвет линий (--color).

Заливка квадрата (--fill) и цвет заливки (--fill_color).

Фильтр RGB-компонент (--rgbfilter), позволяющий изменять значение цветовой компоненты (красной, зеленой или синей) для всего изображения.

Поворот изображения или его части на 90, 180 или 270 градусов (--rotate), с параметрами:

Координаты левого верхнего (--left_up) и правого нижнего углов (--right_down).

Угол поворота (--angle).

Программа завершает работу после выполнения одного из действий, выбранных пользователем.

Исходный код программы: Приложение А.

Тестирование и демонстрация работы программы: Приложение Б.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью данной работы является создание программы для обработки PNG-изображений с использованием командной строки (CLI) и, опционально, графического пользовательского интерфейса (GUI). Программа будет обеспечивать проверку соответствия файлов формату PNG, их обработку согласно заданным параметрам и сборку с помощью Makefile.

Для достижения цели необходимо выполнить следующие задачи:

Изучение формата PNG и библиотеки libpng для работы с изображениями.

Разработка CLI для взаимодействия с пользователем и обработки команд.

Реализация функций для обработки изображений, включая:

Рисование квадрата с диагоналями и возможностью заливки.

Применение фильтра к RGB-компонентам изображения.

Поворот изображения на заданный угол.

Обеспечение проверки PNG-формата и корректной обработки ошибок.

Реализация выравнивания данных в файле и сохранение стандартных значений PNG-заголовков.

Сборка программы с использованием Makefile.

Тестирование программы на различных входных данных.

Программа должна быть удобной в использовании, с четко определенными функциями и параметрами для обработки изображений. Все операции должны быть реализованы в виде отдельных функций, что облегчит тестирование и дальнейшее расширение функционала программы.

ХОД ВЫПОЛНЕНИЯ РАБОТЫ

1.1. Структуры данных и функции

Структуры:

enum RGB: Перечисление для представления компонентов цвета RGB.

class PNGImage: Класс для работы с PNG изображениями.

struct p_filter: Структура для хранения информации о фильтре цвета.

struct p_rotate: Структура для хранения информации о вращении изображения.

s*truct p_sqared:* Структура для хранения информации о рисовании квадратных линий.

Функции:

p_filter parse_filter(raw_args raw): Функция для разбора аргументов фильтра цвета.

p_rotate parse_rotate(raw_args raw): Функция для разбора аргументов вращения изображения.

p_sqared parse_sqared(raw_args raw): Функция для разбора аргументов рисования квадратных линий.

void help(): Вывод справки;

IOERROR(char message*): Макрос для вывода ошибки ввода-вывода и выхода с кодом 43.

ARGERROR(char message*): Макрос для вывода ошибки аргументов и выхода с кодом 40.

PNGImage(const std::string filename): Конструктор класса, загружающий PNG из файла.

~PNGImage(): Деструктор класса, освобождающий ресурсы.

void write(const std::string filename): Функция для записи PNG изображения в файл.

void info(): Функция для вывода информации об изображении.

bool is_valid(): Функция для проверки валидности PNG изображения.

void rotate(int coords[2][2], int angle): Функция для вращения изображения на заданный угол.

void rgbfilter(enum RGB component, int value): Функция для применения фильтра к компоненту RGB.

void squared_lines(int coords[2], int side_size, int thickness, int color[3], bool fill, int fill_color[3]): Функция для рисования квадратных линий на изображении.

Дополнительные структуры и функции:

enum commands: Перечисление для представления команд обработки аргументов.

typedef std::map<std::string, std::string> raw_args: Тип для хранения сырых аргументов.

*raw_args read_args(int argc, char *argv[]):* Функция для чтения аргументов командной строки.

void process_args(raw_args raw): Функция для обработки аргументов командной строки.

*int main(int argc, char *argv[]):* Главная функция программы.

Классы:

PNGImage: Этот класс и функции предназначены для открытия, обработки и сохранения PNG изображений с различными операциями, такими как вращение, применение фильтров и рисование линий, использует для этого библиотеку *libpng* для работы с PNG файлами.

ЗАКЛЮЧЕНИЕ

Обработка изображений в Си++ успешно освоена, задания курсовой выполнены.

Лучше не использовать Си++ для обработки изображений без библиотек, если в этом нет необходимости.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.c

```
/** args
 * -h --help (if none)
 * --info
 * -i --input
 * -o --output
 * --squared_lines
--left_up <left>.<up>
--side_size <n>
--thickness <n>
--color <r>.<g>.<b>
--fill
--fill_color <r>.<g>.<b>
 * --rgbfilter
--component_name <red/green/blue>
--component_value <0-255>
 * --rotate
--left_up <left>.<up>
--right_down <right>.<down>
--angle <90/180/270>
 */
#include "args.h"
#include "image.h"
#include <getopt.h>
#include <iostream>
#include <map>
#include <set>
#include <string>

std::map<std::string, enum commands> commands = {
    {"info", _info},
    {"squared_lines", _squared_lines},
    {"rgbfilter", _rgbfilter},
    {"rotate", _rotate},
    {"help", _help},
};

raw_args read_args(int argc, char *argv[]) {
    raw_args raw;
    int c;
    int option_index;
    opterr = 0;
    static struct option long_options[] = {
        {"help", no_argument, nullptr, 'h'},
        {"info", no_argument, nullptr, 0},
        {"input", required_argument, nullptr, 'i'},
        {"output", required_argument, nullptr, 'o'},
        {"squared_lines", no_argument, nullptr, 0},
        {"left_up", required_argument, nullptr, 0},
        {"right_down", required_argument, nullptr, 0},
        {"side_size", required_argument, nullptr, 0},
```

```
            {"thickness", required_argument, nullptr, 0},
            {"color", required_argument, nullptr, 0},
            {"fill", no_argument, nullptr, 0},
            {"fill_color", required_argument, nullptr, 0},
            {"rgbfilter", no_argument, nullptr, 0},
            {"component_name", required_argument, nullptr, 0},
            {"component_value", required_argument, nullptr, 0},
            {"rotate", no_argument, nullptr, 0},
            {"angle", required_argument, nullptr, 0},
            {nullptr, 0, nullptr, 0},
        };
        while ((c = getopt_long(argc, argv, "hi:o:", long_options,
&option_index)) != -1) {
            switch (c) {
                case 'h':
                    if (raw.count("command"))
                        ARGERROR("argument already specified");
                    raw.insert({"command", "help"});
                    break;
                case 'i':
                    raw.insert({"input", optarg});
                    break;
                case 'o':
                    raw.insert_or_assign("output", optarg);
                    break;
                case 0: {
                    std::string               opt_name               =
long_options[option_index].name;
                    bool is_command = commands.count(opt_name);
                    if   (raw.count(opt_name)   ||   (is_command   &&
raw.count("command"))) {
                        ARGERROR("argument already specified");
                    } else if (is_command) {
                        raw.insert({"command", opt_name});
                    } else if (optarg) {
                        raw.insert({opt_name, optarg});
                    } else {
                        raw.insert({opt_name, ""});
                    }
                } break;
                case '?':
                default:
                    ARGERROR("unknown arg: '" << argv[optind - 1] <<
'\'');
                    break;
            }
        }
        if (!raw.count("command"))
            raw.insert({"command", "help"});

    #ifdef DEBUG
        std::cout << raw["command"] << ' ' << argv[optind] << ' ' <<
optind << ' ' << argc << ' ' << '\n';
    #endif
        if (argc > optind && (optind + 1 == argc) && raw["command"] !=
"help") {
            if (raw.count("input"))
                ARGERROR("input already specified");
```

```
            raw.insert({"input", argv[optind]});
        } else if (argc > optind)
            ARGERROR("exsessive args");

        if (!raw.count("output"))
            raw.insert({"output", "out.png"});
        if (!raw.count("input") && raw["command"] != "help")
            ARGERROR("missing input file");
        if (raw["output"] == raw["input"] && raw["command"] != "info")
            ARGERROR("output file should not be equal input file");
        return raw;
    }

    void help() {
        std::cout << "Usage: image_tool [options] [input_file]\n"
                  << "Options:\n"
                  << "  -h, --help                    Show this help
message and exit.\n"
                  << "  --info                      Display information
about the input image.\n"
                  << "  -i, --input <file>          Specify the input
image file.\n"
                  << "  -o, --output <file>         Specify the output
image file (default: out.png).\n"
                  << "  --squared_lines             Draw squared lines
on the image.\n"
                  << "  --left_up <left>.<up>       Specify the top-
left corner for squared lines or rotation.\n"
                  << "  --side_size <n>             Specify the side
size for squared lines.\n"
                  << "  --thickness <n>             Specify the line
thickness for squared lines.\n"
                  << "  --color <r>.<g>.<b>         Specify the color
for squared lines in RGB format.\n"
                  << "  --fill                      Enable filling the
squares with color.\n"
                  << "  --fill_color <r>.<g>.<b>    Specify the fill
color for squared lines in RGB format.\n"
                  << "  --rgbfilter                 Apply an RGB filter
to the image.\n"
                  << "  --component_name <name>     Specify the RGB
component to filter (red, green, blue).\n"
                  << "  --component_value <0-255>  Set the value for
the RGB component filter.\n"
                  << "  --rotate                    Rotate a region of
the image.\n"
                  << "  --right_down <right>.<down>Specify the bottom-
right corner for rotation.\n"
                  << "  --angle <90/180/270>        Set the rotation
angle in degrees.\n"
                  << "\n"
                  << "Example:\n"
                  <<  "    image_tool  --input  image.png  --output
edited.png --squared_lines --left_up 10.10 --side_size 100 --thickness
5 --color 255.0.0 --fill --fill_color 0.255.0\n"
                  << "This command will draw a filled red square with
a green fill on 'image.png' and save it as 'edited.png'.\n";
        exit(0);
```

```
};

struct p_filter {
    RGB color;
    int val;
};

p_filter parse_filter(raw_args raw) {
    p_filter o;

    for (auto i : {"component_name", "component_value"})
        if (!raw.count(i))
            ARGERROR("missing " << i << " argument");

    std::string color = raw["component_name"];
    if (color == "red")
        o.color = RGB::RED;
    else if (color == "blue")
        o.color = RGB::BLUE;
    else if (color == "green")
        o.color = RGB::GREEN;
    else
        ARGERROR("invalid component_name");
    try {
        o.val = std::stoi(raw["component_value"]);
    } catch (std::exception &e) {
        ARGERROR("invalid component_value");
    }
    if (o.val < 0 || o.val > 255)
        ARGERROR("invalid component_value");
    return o;
}

struct p_rotate {
    int coords[2][2]; // left_up, right_down
    int angle;
};

p_rotate parse_rotate(raw_args raw) {
    p_rotate o;

    for (auto i : {"left_up", "right_down", "angle"})
        if (!raw.count(i))
            ARGERROR("missing " << i << " argument");

    if (sscanf(raw["left_up"].c_str(), "%d.%d", &o.coords[0][0],
&o.coords[0][1]) != 2)
        ARGERROR("invalid left_up");

    if (sscanf(raw["right_down"].c_str(), "%d.%d",
&o.coords[1][0], &o.coords[1][1]) != 2)
        ARGERROR("invalid right_down");

    if (o.coords[0][0] > o.coords[1][0] || o.coords[0][1] >
o.coords[1][1])
        ARGERROR("invalid coords");

    try {
```

```
            o.angle = std::stoi(raw["angle"]);
        } catch (std::exception &e) {
            ARGERROR("invalid angle");
        }
        if (!(o.angle == 90 || o.angle == 180 || o.angle == 270))
            ARGERROR("invalid angle");
        return o;
    }

    struct p_sqared {
        int coords[2];
        int side_size;
        int thickness;
        int color[3];
        bool fill = false;
        int fill_color[3] = {0, 0, 0};
    };

    p_sqared parse_sqared(raw_args raw) {
        p_sqared o;

        for (auto i : {"left_up", "side_size", "color", "thickness"})
            if (!raw.count(i))
                ARGERROR("missing " << i << " argument");

        if (raw.count("fill"))
            o.fill = true;

        if (o.fill && !raw.count("fill_color"))
            ARGERROR("missing fill_color argument");

        try {
            o.side_size = std::stoi(raw["side_size"]);
            o.thickness = std::stoi(raw["thickness"]);
        } catch (std::exception &e) {
            ARGERROR("invalid side_size or thickness");
        }

        if  (sscanf(raw["left_up"].c_str(),  "%d.%d",  &o.coords[0],
&o.coords[1]) != 2)
            ARGERROR("invalid left_up");

        if  (sscanf(raw["color"].c_str(),  "%d.%d.%d",  &o.color[0],
&o.color[1], &o.color[2]) != 3)
            ARGERROR("invalid color");

        if (o.fill && sscanf(raw["fill_color"].c_str(), "%d.%d.%d",
&o.fill_color[0], &o.fill_color[1], &o.fill_color[2]) != 3)
            ARGERROR("invalid fill_color");

        if (o.color[0] < 0 || o.color[0] > 255 ||
            o.color[1] < 0 || o.color[1] > 255 ||
            o.color[2] < 0 || o.color[2] > 255)
            ARGERROR("invalid color");

        if (o.fill && (o.fill_color[0] < 0 || o.fill_color[0] > 255 ||
                       o.fill_color[1] < 0 || o.fill_color[1] > 255 ||
                       o.fill_color[2] < 0 || o.fill_color[2] > 255))
```

```
            ARGERROR("invalid fill_color");
        return o;
    }

    void process_args(raw_args raw) {
        enum commands c = commands[raw["command"]];

        if (c == commands::_help)
            help();

        PNGImage image(raw["input"]);

        if (c != _info && !image.is_valid()) {
            std::cerr << "Error: not valid png format\n";
            exit(42);
        }
        switch (c) {
            case commands::_info:
                image.info();
                break;
            case commands::_rgbfilter: {
                p_filter p = parse_filter(raw);
                image.rgbfilter(p.color, p.val);
                image.write(raw["output"]);
            } break;
            case commands::_squared_lines: {
                p_sqared p = parse_sqared(raw);
                image.squared_lines(p.coords,            p.side_size,
p.thickness, p.color, p.fill, p.fill_color);
                image.write(raw["output"]);
            } break;
            case commands::_rotate: {
                p_rotate p = parse_rotate(raw);
                image.rotate(p.coords, p.angle);
                image.write(raw["output"]);
            } break;
            case commands::_help:
                break;
        }
    };
```

Название файла: image.cpp

```
    #include "image.h"
    #include <cmath>
    #include <cstring>
    #include <iostream>
    #include <png.h>
    #include <stdexcept>
    #include <vector>

    PNGImage::PNGImage(const std::string filename) {
        png_byte header[8];
        // Open file
        FILE *fp = fopen(filename.c_str(), "rb");
        if (!fp || !fread(header, 1, 8, fp))
            IOERROR("file cannot be opened");

        if (png_sig_cmp(header, 0, 8))
```

```
            IOERROR("NOT A PNG");

        png_ptr = png_create_read_struct(PNG_LIBPNG_VER_STRING, NULL,
NULL, NULL);
        if (!png_ptr)
            IOERROR("png_create_read_struct fail");

        info_ptr = png_create_info_struct(png_ptr);
        if (!info_ptr) {
            png_destroy_read_struct(&png_ptr,        (png_infopp)NULL,
(png_infopp)NULL);
            IOERROR("png_create_info_struct fail");
        }

        // error handling set up
        if (setjmp(png_jmpbuf(png_ptr))) {
            png_destroy_read_struct(&png_ptr,              &info_ptr,
(png_infopp)NULL);
            fclose(fp);
            IOERROR("init_io");
        }

        png_init_io(png_ptr, fp);

        png_set_sig_bytes(png_ptr, 8);
        png_read_info(png_ptr, info_ptr);

        width = png_get_image_width(png_ptr, info_ptr);
        height = png_get_image_height(png_ptr, info_ptr);
        color_type = png_get_color_type(png_ptr, info_ptr);
        bit_depth = png_get_bit_depth(png_ptr, info_ptr);
        // number_of_passes = png_set_interlace_handling(png_ptr);
        png_read_update_info(png_ptr, info_ptr);

        if (setjmp(png_jmpbuf(png_ptr))) {
            png_destroy_read_struct(&png_ptr,              &info_ptr,
(png_infopp)NULL);
            fclose(fp);
            IOERROR("read_image failed");
        }
        row_pointers.resize(height);
        for (auto &row : row_pointers)
            row = new png_byte[png_get_rowbytes(png_ptr, info_ptr)];

        png_read_image(png_ptr, row_pointers.data());

        fclose(fp);
    }

    PNGImage::~PNGImage() {
        // Clean up
        if (png_ptr && info_ptr)
            png_destroy_read_struct(&png_ptr,              &info_ptr,
(png_infopp)NULL);
    }

    void PNGImage::write(const std::string filename) {
```

```
        // Open file
        FILE *fp = fopen(filename.c_str(), "wb");
        if (!fp) IOERROR("file cannot be written");

        // Initialize write structure
        png_ptr = png_create_write_struct(PNG_LIBPNG_VER_STRING, NULL,
NULL, NULL);
        if (!png_ptr) IOERROR("png_create_write_struct fail");

        // Initialize info structure
        info_ptr = png_create_info_struct(png_ptr);
        if (!info_ptr) {
            png_destroy_write_struct(&png_ptr, (png_infopp)NULL);
            IOERROR("png_create_info_struct fail");
        }

        // Set up error handling
        if (setjmp(png_jmpbuf(png_ptr))) {
            png_destroy_write_struct(&png_ptr, &info_ptr);
            fclose(fp);
            IOERROR("init_io fail");
        }

        png_init_io(png_ptr, fp);

        // Write header
        if (setjmp(png_jmpbuf(png_ptr))) {
            png_destroy_write_struct(&png_ptr, &info_ptr);
            fclose(fp);
            IOERROR("writing header fail");
        }

        //  png_write_png(png_ptr,  info_ptr,  PNG_TRANSFORM_IDENTITY,
NULL);
        // #ifdef lowlevel
        png_set_IHDR(png_ptr, info_ptr, width, height,
                     8, PNG_COLOR_TYPE_RGB, PNG_INTERLACE_NONE,
                     PNG_COMPRESSION_TYPE_BASE,
PNG_FILTER_TYPE_BASE);

        png_write_info(png_ptr, info_ptr);
        // Write image data
        if (setjmp(png_jmpbuf(png_ptr))) {
            png_destroy_write_struct(&png_ptr, &info_ptr);
            fclose(fp);
            IOERROR("write data fail");
        }

        png_write_image(png_ptr, row_pointers.data());
        // End write
        if (setjmp(png_jmpbuf(png_ptr))) {
            png_destroy_write_struct(&png_ptr, &info_ptr);
            fclose(fp);
            IOERROR("write end fail");
        }

        png_write_end(png_ptr, NULL);
        // #endif
```

```
        fclose(fp);
    }
    void PNGImage::info() {
        std::cout << "PNG file size: " << this->width << "x" <<
this->height << "\n"
                  << "color_type: " << (unsigned int)this->color_type
<< "\n"
                  << "bit_depth: " << (unsigned int)this->bit_depth <<
"\n"
                  << "Valid: " << this->is_valid() << "\n";
    }
    bool PNGImage::is_valid() {
        return    (this->color_type    ==    PNG_COLOR_TYPE_RGB)    &&
(this->bit_depth == 8);
    };

    void PNGImage::rgbfilter(enum RGB component, int value) {
        for (int y = 0; y < this->height; y++) {
            png_byte *row = this->row_pointers[y];
            for (int x = 0; x < this->width; x++) {
                png_byte *pix = &(row[x * 3]);
                pix[component] = value;
            }
        }
    }

    void PNGImage::rotate(int coords[2][2], int angle) {
        using std::min, std::max;
        // coords {left_up, right_down}
        // coords are coords of rectangle two corners to rotat
        int Rwidth = coords[1][0] - coords[0][0];
        int Rheight = coords[1][1] - coords[0][1];
        int centerX = coords[0][0] + Rwidth / 2;
        int centerY = coords[0][1] + Rheight / 2;

        std::vector<png_byte *> buffer;
        buffer.resize(this->height);

        for (auto i = 0; i < this->height; i++) {
            buffer[i] = new png_byte[this->width * 3];
            memcpy(buffer[i], this->row_pointers[i], this->width * 3);
        }
        for (int y = 0; y < this->height; y++) {
            for (int x = 0; x < this->width; x++) {
                int oldX = 0, oldY = 0;
                int delta_x = 0, delta_y = 0;
                switch (angle) {
                    case 270:
                        oldX = centerX + (y - centerY);
                        oldY = centerY - (x - centerX);
                        if ((Rheight % 2) != (Rwidth % 2)) {
                            if ((Rheight % 2)) {
                                delta_y = -1 + (Rheight % 2);
                                delta_x = -(Rwidth % 2);
                            } else {
                                delta_y = -(Rheight % 2);
                                delta_x = -(Rwidth % 2);
```

```
                        }
                    } else {
                        delta_y = -((1 + Rheight) % 2);
                        delta_x = -((1 + Rwidth) % 2);
                    }
                    break;
                case 180:
                    oldX = centerX - (x - centerX);
                    oldY = centerY - (y - centerY);
                    if (!(Rheight % 2) && !(Rwidth % 2)) {
                        delta_x = -1 + (Rheight % 2);
                        delta_y = -1 + (Rwidth % 2);
                    } else {
                        delta_x = -(Rheight % 2);
                        delta_y = -(Rwidth % 2);
                    }
                    break;
                case 90:
                    oldX = centerX - (y - centerY);
                    oldY = centerY + (x - centerX);
                    delta_y = -1 + (Rwidth % 2);
                    delta_x = -(Rheight % 2);
                    break;
                default:
                    return;
            }

            // Check if the old coordinates are within the bounds
of the buffer
            if ((oldX < 0 || oldY < 0 || oldX >= this->width ||
oldY >= this->height) ||
                (oldY < coords[0][1] || oldY >= coords[1][1] ||
oldX < coords[0][0] || oldX >= coords[1][0]))
                continue;
            png_byte *old_pix = &(this->row_pointers[oldY][oldX *
3]);
            if (y + delta_y > 0 && x + delta_x > 0) {
                png_byte *buf_pixel = &(buffer[y + delta_y][(x +
delta_x) * 3]);
                std::copy(old_pix, old_pix + 3, buf_pixel);
            }
        }
    }

    for (int i = 0; i < this->height; ++i) {
        delete[] this->row_pointers[i];
        this->row_pointers[i] = buffer[i];
    }
}

void PNGImage::squared_lines(int coords[2], int side_size, int
thickness, int color[3], bool fill, int fill_color[3]) {
    using std::max, std::min, std::abs;

    // min_1|   |max_1 -- min_2|   |max_2
    // int min_1 = coords - thickness / 2;
    // int max_1 = coords + thickness / 2 + thickness % 2;
```

```
        // int min_2 = coords + side_size - thickness / 2 - thickness %
2;
        // int max_2 = coords + side_size + thickness / 2;
        // clamped
        int  x_min_1  =  max(0,  min(coords[0]  -  thickness  /  2,
this->width));
        int  y_min_1  =  max(0,  min(coords[1]  -  thickness  /  2,
this->height));
        int x_max_1 = min(this->width, coords[0] + thickness / 2 +
thickness % 2);
        int y_max_1 = min(this->height, coords[1] + thickness / 2 +
thickness % 2);
        int x_min_2 = max(0, min(coords[0] + side_size - thickness / 2
- thickness % 2, this->width));
        int y_min_2 = max(0, min(coords[1] + side_size - thickness / 2
- thickness % 2, this->height));
        int  x_max_2  =  min(this->width,  coords[0]  +  side_size  +
thickness / 2);
        int  y_max_2  =  min(this->height,  coords[1]  +  side_size  +
thickness / 2);

        if (fill)
            for (int y = y_min_1; y < y_max_2; y++)
                for (int x = x_min_1; x < x_max_2; x++)
                    std::copy(fill_color,     fill_color     +     3,
&(this->row_pointers[y][x * 3]));

        // horizontal
        for (int y = y_min_1; y < y_max_2; y++) {
            for (int x = x_min_1; x < x_max_1; x++)
                std::copy(color, color + 3, &(this->row_pointers[y][x
* 3]));
            for (int x = x_min_2; x < x_max_2; x++)
                std::copy(color, color + 3, &(this->row_pointers[y][x
* 3]));
        }
        for (int x = x_min_1; x < x_max_2; x++) {
            for (int y = y_min_1; y < y_max_1; y++)
                std::copy(color, color + 3, &(this->row_pointers[y][x
* 3]));
            for (int y = y_min_2; y < y_max_2; y++)
                std::copy(color, color + 3, &(this->row_pointers[y][x
* 3]));
        }

        for (int t = 0; t < thickness + 1 - thickness % 2; t++) {
            int _x = coords[0] + t - thickness / 2;
            for (int i = 0; i < side_size; i++) {
                if (coords[1] + i < this->height &&
                    coords[1] + i >= 0) {
                    if (_x + i >= 0 &&
                        _x + i - thickness / 2 < this->width)
                        std::copy(color,       color       +       3,
&(this->row_pointers[coords[1] + i][(_x + i) * 3]));
                    if (_x - 1 + side_size - i >= 0 &&
                        _x - 1 + side_size - i < this->width)
                        std::copy(color,       color       +       3,
&(this->row_pointers[coords[1] + i][(_x + side_size - i - 1) * 3]));
```

```
            }
        }
    }
}
```

Название файла: image.h

```
#pragma once
#include <iostream>
#include <png.h>
#include <string>
#include <vector>

typedef unsigned char pixel[3];
enum RGB { RED = 0,
           GREEN = 1,
           BLUE = 2 };

#define IOERROR(msg)                                    \
    do {                                                \
        std::cout << "IOError: " << msg << "\n"; \
        exit(43);                                       \
    } while (0)

class PNGImage {
private:
    png_structp png_ptr;
    png_infop info_ptr;
    std::vector<png_bytep> row_pointers;
    int width, height;
    png_byte color_type;
    png_byte bit_depth;

public:
    PNGImage(const std::string filename);
    ~PNGImage();
    void write(const std::string filename);
    void info();
    bool is_valid();
    void rotate(int coords[2][2], int angle);
    void rgbfilter(enum RGB component, int value);
    void   squared_lines(int   coords[2],   int   side_size,   int
thickness, int color[3], bool fill, int fill_color[3]);
};
```

Название файла: args.h

```
#pragma once

#include <iostream>
#include <map>

#define ARGERROR(msg)                                   \
    do {                                                \
        std::cout << "ArgError: " << msg << "\n" \
                  << "seek --help\n";           \
        exit(40);                                       \
    } while (0)
```

```
enum commands { _info,
                _squared_lines,
                _rgbfilter,
                _rotate,
                _help };

typedef std::map<std::string, std::string> raw_args;

raw_args read_args(int argc, char *argv[]);
void process_args(raw_args raw);
```

Название файла: main.cpp

```
#include "args.h"
#include <png.h>

#define GREETING "Course work for option 4.24, created by Vadim
Grebenyuk.\n"

int main(int argc, char *argv[]) {
    std::cout << GREETING;
    process_args(read_args(argc, argv));
    return 0;
}
```

Название файла: Makefile

```
# g++ main.cpp args.cpp image.cpp -o args -lpng -g
CC = g++
CFLAGS = -O2 -flto -pipe -lpng -std=gnu++1z
EXE = cw
SRC = $(wildcard *.cpp) $(wildcard **/*.cpp)

all: $(EXE)

$(EXE): $(OBJ)
     $(CC) $(CFLAGS) $(SRC) -lpng -o $(EXE)

run: $(EXE)
     ./$(EXE)
```

ПРИЛОЖЕНИЕ Б

ДЕМОНСТРАЦИЯ РАБОТЫ ПРОГРАММЫ

./cw --side_size 414 --fill --fill_color 3.27.53 --input ss.png --left_up 465.251 --thickness 18 --color 199.24.242 --squared_lines

./cw –help

```
vadim@ASUS-Laptop:/mnt/d/Projects/Labs/pr/cw/Grebenyuk_Vadim_cw/src$ ./cw --help
Course work for option 4.24, created by Vadim Grebenyuk.
Usage: image_tool [options] [input_file]
Options:
  -h, --help                  Show this help message and exit.
  --info                      Display information about the input image.
  -i, --input <file>          Specify the input image file.
  -o, --output <file>         Specify the output image file (default: out.png).
  --squared_lines             Draw squared lines on the image.
  --left_up <left>.<up>       Specify the top-left corner for squared lines or rotation.
  --side_size <n>             Specify the side size for squared lines.
  --thickness <n>             Specify the line thickness for squared lines.
  --color <r>.<g>.<b>         Specify the color for squared lines in RGB format.
  --fill                      Enable filling the squares with color.
  --fill_color <r>.<g>.<b>    Specify the fill color for squared lines in RGB format.
  --rgbfilter                 Apply an RGB filter to the image.
  --component_name <name>     Specify the RGB component to filter (red, green, blue).
  --component_value <0-255>   Set the value for the RGB component filter.
  --rotate                    Rotate a region of the image.
  --right_down <right>.<down>Specify the bottom-right corner for rotation.
  --angle <90/180/270>        Set the rotation angle in degrees.

Example:
  image_tool --input image.png --output edited.png --squared_lines --left_up 10.10 --side_size
 100 --thickness 5 --color 255.0.0 --fill --fill_color 0.255.0
This command will draw a filled red square with a green fill on 'image.png' and save it as 'ed
ited.png'.
```

./cw --output 8o.png --input 8new.png --angle 270 --left_up 144.383 --rotate --right_down 374.490