**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 |  |
| 1. | Ход выполнения работы |  |
| 1.1. | Структуры данных и функции |  |
|  | Заключение |  |
|  | Приложение А. Исходный код программы |  |
|  | Приложение Б. Демонстрация работы программы |  |

# 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)

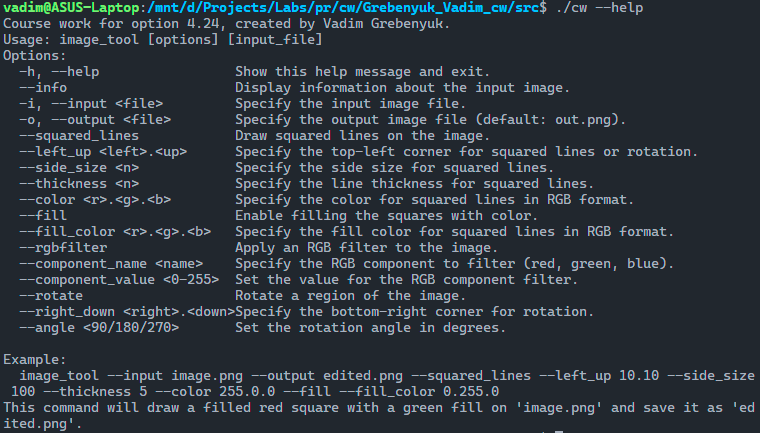
# приложение б Демонстрация работы программы

./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

****

./cw --output 8o.png --input 8new.png --angle 270 --left\_up 144.383 --rotate --right\_down 374.490



