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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Обработка PNG файла

Студент гр. 3341 _____ Перевалов П.И.

Преподаватель _____ Глазунов С.А.

Санкт-Петербург

2024

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студент Перевалов П.И.

Группа 3341

Вариант 16

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. Флаг для выполнения данной операции: `--square`. 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. 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) Поменять местами 4 куска области. Флаг для выполнения данной операции: `--exchange`. Выбранная пользователем прямоугольная область делится на 4 части и эти части меняются местами. 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
    - Способом обмена частей: “по кругу”, по диагонали. Флаг `--exchange_type`, возможные значения: `clockwise`, `counterclockwise`, `diagonals`
- (3) Находит самый часто встречаемый цвет и заменяет его на другой

заданный цвет. Флаг для выполнения данной операции: `--freq_color`. Функционал определяется:

- Цветом, в который надо перекрасить самый часто встречаемый цвет. Флаг `--color` (цвет задаётся строкой `rrr.ggg.bbb`, где rrr/ggg/bbb – числа, задающие цветовую компоненту. пример `--color 255.0.0` задаёт красный цвет)

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

Содержание пояснительной записки:
разделы «Аннотация», «Содержание», «Введение», «Ход работы», «Пример работы программы», «Заключение», «Список использованных источников»

Предполагаемый объем пояснительной записки:
Не менее 15 страниц.

Дата выдачи задания: 18.03.2024

Дата сдачи реферата: 19.05.2024

Дата защиты реферата: 23.05.2024

Студент		Перевалов П.И.
Преподаватель		Глазунов С.А.

АННОТАЦИЯ

В данной курсовой работе была реализована программа, обрабатывающая PNG изображения. Программа проверяет тип изображения, его версию, при соответствии требованиям в дальнейшем обрабатывает его и подаёт на выход изменённую копию изображения. Взаимодействие с программой осуществляется с помощью CLI (интерфейс командной строки).

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель работы заключается в создании программы для обработки bmp-файлов с использованием командной строки (CLI) и, при необходимости, графического интерфейса пользователя (GUI). Программа должна проверять соответствие файла формату PNG, а также реализовывать следующий функционал:

1) Рисование квадрата с возможностью указания координат точек, толщины линий и цвета.
2) Отражение частей изображения с возможностью указания оси отражения.
3) Нахождение самого часто встречаемого цвета с возможностью замены его на другой заданный цвет.

Программа должна также обеспечивать сохранение всех полей стандартных PNG заголовков с соответствующими значениями и обеспечивать выравнивание данных в файле. Каждая функция должна быть реализована отдельно, а ввод и вывод данных также должны быть организованы в виде отдельных функций.

1. РАБОТА С ФАЙЛАМИ

Функция read_png_file отвечает за чтение данных из png-файла. Она принимает имя файла, а также ссылки на высоту, ширину, указатель где будет храниться информация о пикселях изображения, тип цвета и глубину цвета. Функция открывает файл в двоичном режиме, считывает заголовки и пиксели изображения, выделяя память для массива пикселей и осуществляя необходимое выравнивание данных. В случае неудачи или некорректного формата файла функция прекращает работу программы.

Функция write_png_file записывает данные в png-файл. Она принимает имя файла, высоту и ширину изображения, а также ссылку на двумерный массив содержащий цветовую информацию изображения. Функция открывает файл в двоичном режиме для записи, записывает заголовки и пиксели изображения с учетом выравнивания данных и закрывает файл.

2. ВВОД АРГУМЕНТОВ

CLI (Command Line Interface) в данной программе реализована с использованием опций командной строки. Для обработки аргументов командной строки используются структуры option, которые определяют различные действия, доступные в программе.

Для каждой основной команды (line, mirror, color) определены соответствующие наборы опций командной строки. Например, для команды line опции определены в структуре option_line, а для команды color — в структуре option_penta.

Функция processing_flags осуществляет анализ аргументов командной строки и выбор нужного действия в зависимости от команды и вызывает функцию обработки этой функции.

Каждая функция обработки команды (processSquareCommand, processExchangeCommand, processFreqCommand) осуществляет парсинг опций командной строки и вызов соответствующей функции для выполнения задачи (например, line для команды line). В случае неверных данных или ошибочных аргументов функции выводят сообщение об ошибке и завершают работу программы с соответствующим кодом ошибки.

Таким образом, пользователь может использовать CLI для выполнения различных действий с png-файлами, таких как рисование линий, пентаграммы или отражение области изображения, передавая соответствующие аргументы командной строки.

3. ОСНОВНЫЕ ФУНКЦИИ

checkValidName: Проверяет валидность имени файла с помощью регулярного выражения.

findUnknownKey: Ищет неизвестные ключи среди аргументов командной строки.

outputHelp: Выводит справочную информацию о доступных опциях.

outputInfo: Выводит информацию о высоте и ширине изображения.

isPNG: Проверяет, является ли файл PNG-изображением.

convertCoords: Преобразует строку координат в целые числа для x и y.

read_png_file: Считывает PNG-файл и загружает его в структуру данных.

write_png_file: Записывает измененное изображение в новый PNG-файл.

set_pixel_color: Устанавливает цвет конкретного пикселя.

getRgb: Преобразует строку с цветом в формате RGB в массив целых чисел.

checkValidCoord: Проверяет, находятся ли координаты внутри границ изображения.

drawCircle: Рисует окружность на изображении.

drawLine: Рисует линию на изображении.

fillSquare: Заполняет квадрат заданным цветом.

drawSquare: Рисует квадрат на изображении.

processSquareCommand: Обрабатывает команду рисования квадрата.

processExchangeCommand: Обрабатывает команду обмена областями изображения.

findMostFrequentColor: Находит наиболее часто встречающийся цвет на изображении.

processFreqCommand: Обрабатывает команду замены наиболее часто встречающегося цвета.

ЗАКЛЮЧЕНИЕ

В ходе выполнения данного проекта была разработана программа для обработки изображений в формате PNG. Программа имеет командную строку (CLI), что обеспечивает удобство взаимодействия с пользователем. Она реализует следующий функционал:

Рисование квадрата: Пользователь может указать координаты начала и конца линии, его толщину линий и цвет.

Отражение области: Пользователь задает координаты левого верхнего и правого нижнего угла, а так же ось отражения.

Замена цвета: Позволяет заменить самый часто встречающийся цвет. Пользователь указывает цвет.

Важным аспектом является обработка входных данных и валидация параметров пользовательского ввода. Программа проверяет соответствие входного изображения формату PNG и корректность всех переданных параметров.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1.https://se.moevm.info/lib/exe/fetch.php/courses:programming:programming_cw_metoda_2nd_course_last_ver.pdf.pdf - методические материалы для написания курсовой работы

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include <iostream>
#include <fstream>
#include <getopt.h>
#include <png.h>
#include <regex>
#include <string.h>
#include <cmath>
#include <vector>

#define CW "Course work for option 4.16, created by Pasha Perevalov\n"
#define H "--square - рисование квадрата, --exchange - поменять 4 области местами, --freq_color - замена самого часто встречаемого цвета\n"

#define signature_size 8

typedef struct{
int R;
int G;
int B;
}Rgb;

static struct option keys[] = {
{"help", no_argument, 0, 'h'},
{"output", required_argument, 0, 'o'},
{"input", required_argument, 0, 'i'},
{"info", no_argument, 0, 'd'},
{"square", no_argument, 0, 's'},
{"exchange", no_argument, 0, 'e'},
{"freq_color", no_argument, 0, 'y'},
{"left_up", required_argument, 0, 'u'},
    {"right_down", required_argument, 0, 'p'},
{"side_size", required_argument, 0, 'q'},
{"thickness", required_argument, 0, 'w'},
{"color", required_argument, 0, 'z'},
{"fill", no_argument, 0, 'f'},
{"fill_color", required_argument, 0, 'g'},
```

```
{"exchange_type", required_argument, 0, 't'},
{"radius", required_argument, 0, 'l'},
{0, 0, 0, 0}
};

static struct option selectAction[] = {
{"help", no_argument, 0, 'h'},
{"output", required_argument, 0, 'o'},
{"input", required_argument, 0, 'i'},
{"info", no_argument, 0, 'd'},
{"square", no_argument, 0, 'r'},
{"exchange", no_argument, 0, 'x'},
{"freq_color", no_argument, 0, 'c'},
{0, 0, 0, 0}
};

static struct option squareKeys[] = {
{"left_up", required_argument, 0, 'u'},
{"side_size", required_argument, 0, 'd'},
{"thickness", required_argument, 0, 't'},
{"color", required_argument, 0, 'c'},
{"fill", no_argument, 0, 'f'},
{"fill_color", required_argument, 0, 'g'},
{0, 0, 0, 0}
};

static struct option exchangeKeys[] = {
{"exchange", no_argument, 0, 'd'},
{"right_down", required_argument, 0, 'w'},
{"left_up", required_argument, 0, 'z'},
{"exchange_type", required_argument, 0, 'y'},
{0, 0, 0, 0}
};

static struct option freqKeys[] = {
{"color", required_argument, 0, 'l'},
{0, 0, 0, 0}
};

bool checkValidName(std::string name){
```

```
return(std::regex_match(name, std::regex("((./)||((\\w+/)+))?(\\w+).(\\w+)")));
}

bool findUnknownKey(int argc, char *argv[]){
int keyIndex;
int opt;
char** argvCopy = new char*[argc];
    for (int i = 0; i < argc; ++i) {
        size_t len = strlen(argv[i]) + 1;
        argvCopy[i] = new char[len];
        strncpy(argvCopy[i], argv[i], len);
    }
while(true){
opt = getopt_long(argc, argvCopy, "io:h", keys, &keyIndex);
if(opt == -1){
opt = 0;
break;
}
switch(opt){
case '?':
optind = 1;
return true;
break;
}
}
optind = 1;
return false;
}

void outputHelp(){
   std::cout << H << std::endl;
}

void outputInfo(int &height, int &width){
    std::cout << "Image height: " << height << ", image width: " << width <<
std::endl;
}

bool isPNG(const std::string& filename) {
    const std::vector<unsigned char> pngSignature = {0x89, 0x50, 0x4E, 0x47,
0x0D, 0x0A, 0x1A, 0x0A};
```

```
    std::ifstream file(filename, std::ios::binary);
    if (!file.is_open()) {
        std::cout << "Can't open file: " << filename << std::endl;
        exit(41);
        return false;
    }
    std::vector<unsigned char> fileSignature(8);
    file.read(reinterpret_cast<char*>(fileSignature.data()),
fileSignature.size());
    if (file.gcount() != 8) {
        std::cout << "File too small: " << filename << std::endl;
        exit(41);
        return false;
    }
    return fileSignature == pngSignature;
}

bool convertCoords(std::string stringCoords, int &x, int &y){
if(!std::regex_match(stringCoords.c_str(), std::regex("(-?[0-9]+).(-?[0-9]+)")))
return false;
x = atoi(stringCoords.c_str());
int i = 0;
while(true){
if(stringCoords[i] == '.') break;
i++;
}
y = atoi(i + 1 + stringCoords.c_str());
return true;
}

bool read_png_file(std::string filename, png_bytep* &new_png, int &height, int
&width, png_byte &color_type, png_byte &bit_depth){

    if(isPNG(filename.c_str()) == false) return false;

    FILE *fp = fopen(filename.c_str(), "rb");


    png_structp png = png_create_read_struct(PNG_LIBPNG_VER_STRING, NULL, NULL,
NULL);
```

```
    if(!png) abort();

    png_infop info = png_create_info_struct(png);
    if(!info) abort();

    if(setjmp(png_jmpbuf(png))) abort();

    png_init_io(png, fp);

    png_read_info(png, info);

    width = png_get_image_width(png, info);
    height = png_get_image_height(png, info);
    color_type = png_get_color_type(png, info);
    bit_depth = png_get_bit_depth(png, info);

    if(bit_depth == 16)
        png_set_strip_16(png);

    if(color_type == PNG_COLOR_TYPE_PALETTE)
        png_set_palette_to_rgb(png);

    if(color_type == PNG_COLOR_TYPE_GRAY && bit_depth < 8)
        png_set_expand_gray_1_2_4_to_8(png);

    if(png_get_valid(png, info, PNG_INFO_tRNS))
        png_set_tRNS_to_alpha(png);

    if(color_type == PNG_COLOR_TYPE_RGB || color_type == PNG_COLOR_TYPE_GRAY ||
color_type == PNG_COLOR_TYPE_PALETTE)
        png_set_filler(png, 0xFF, PNG_FILLER_AFTER);

    if(color_type == PNG_COLOR_TYPE_GRAY || color_type ==
PNG_COLOR_TYPE_GRAY_ALPHA)
        png_set_gray_to_rgb(png);

    png_read_update_info(png, info);

    if (new_png) abort();
```

```
    new_png = (png_bytep*)malloc(sizeof(png_bytep) * height);
    for(int y = 0; y < height; y++) {
        new_png[y] = (png_byte*)malloc(png_get_rowbytes(png,info));
    }

    if (!new_png) abort();

    png_read_image(png, new_png);

    fclose(fp);

    png_destroy_read_struct(&png, &info, NULL);

    return true;
}

void write_png_file(std::string filename, png_bytep* new_png, int height, int
width){
    int y;

    FILE *fp = fopen(filename.c_str(), "wb");
    if(!fp) abort();

    png_structp png = png_create_write_struct(PNG_LIBPNG_VER_STRING, NULL, NULL,
NULL);
    if (!png) abort();

    png_infop info = png_create_info_struct(png);
    if (!info) abort();

    if (setjmp(png_jmpbuf(png))) abort();

    png_init_io(png, fp);

    png_set_IHDR(
        png,
        info,
        width, height,
```

```
        8,
        PNG_COLOR_TYPE_RGBA,
        PNG_INTERLACE_NONE,
        PNG_COMPRESSION_TYPE_DEFAULT,
        PNG_FILTER_TYPE_DEFAULT
    );
    png_write_info(png, info);

    if (!new_png) abort();

    png_write_image(png, new_png);
    png_write_end(png, NULL);

    for(int y = 0; y < height; y++) {
        free(new_png[y]);
    }
    free(new_png);

    fclose(fp);

    png_destroy_write_struct(&png, &info);
}

void set_pixel_color(int x, int y, int *color, png_bytep* png){
png[y][(x * 4)] = color[0];
png[y][(x * 4)+1] = color[1];
png[y][(x * 4)+2] = color[2];
}

void getRgb(std::string &color, int* arrayRgb){
char colorRgb[color.size()+1];
char* pointer;
strcpy(colorRgb, color.c_str());
pointer = strtok(colorRgb, ".");
arrayRgb[0] = atoi(pointer);
pointer = strtok(NULL, ".");
arrayRgb[1] = atoi(pointer);
pointer = strtok(NULL, ".");
arrayRgb[2] = atoi(pointer);
}
```

```
bool checkValidCoord(int& x, int& y, int &height, int &width){
if(x >= width || x < 0) return false;
if(y >= height || y < 0) return false;
return true;
}

void drawCircle(int& x, int& y, int thickness, std::string& color,
                png_bytep* &newPng, int &height, int &width){
    int r = thickness / 2;
    int arrayRgb[3];
    getRgb(color, arrayRgb);
    if(r < 1){
if(checkValidCoord(x, y, height, width)){
            set_pixel_color(x, y, arrayRgb, newPng);
}
        return;
    }
    int xc = r;
    int yc = 0;
    int P = 1 - r;
    while (xc >= yc) {
int xMinusXC  = x - xc;
int xPlusXC = x + xc;
int yPlusYC = y + yc;
int yMinusYC = y - yc;
int yPlusXC = y + xc;
int xPlusYC = x + yc;
int xMinusYC  = x - yc;
int yMinusXC = y - xc;
        for (int i = xMinusXC; (i <= xPlusXC)&&(i < width); i++) {
            if ((yPlusYC >= 0)&&(i >= 0)&&(yPlusYC < height)){
                set_pixel_color(i, yPlusYC, arrayRgb, newPng);
            }
            if ((yMinusYC >= 0)&&(i >= 0)&&(yMinusYC < height)) {
                set_pixel_color(i, yMinusYC, arrayRgb, newPng);
            }
        }
        for (int i = xMinusYC; (i <= xPlusYC)&&(i < width); i++) {
            if ((yPlusXC >= 0)&&(i >= 0)&&(yPlusXC < height)) {
                set_pixel_color(i, yPlusXC, arrayRgb, newPng);
```

```
            }
            if ((yMinusXC >= 0)&&(i >= 0)&&(yMinusXC < height)) {
                set_pixel_color(i, yMinusXC, arrayRgb, newPng);
            }
}
yc++;
if (P <= 0) {
            P = P + 2 * yc + 1;
} else {
            xc--;
            P = P + 2 * (yc - xc) + 1;
}
}
}

void drawLine(int firstX, int firstY, int secondX, int secondY, int thickness,
              std::string& color, png_bytep* &newPng, int &height, int &width){
    int dx = abs(secondX - firstX);
    int dy = abs(secondY - firstY);
    int sx = firstX < secondX ? 1 : -1;
    int sy = firstY < secondY ? 1 : -1;
    int err = dx - dy;
    int x = firstX;
    int y = firstY;
    while(x != secondX || y != secondY){
        drawCircle(x, y, thickness, color, newPng, height, width);
        int err2 = 2 * err;
        if(err2 > -dy){
            err -= dy;
            x += sx;
        }
        if(err2 < dx){
            err += dx;
            y += sy;
        }
    }
drawCircle(secondX, secondY, thickness, color, newPng, height, width);
}

void fillSquare(int leftX, int leftY, int rightX, int rightY, int thickness,
png_bytep* &newPng, std::string &fillColor, int &height, int &width){
```

```
int arrayFillRgb[3];
getRgb(fillColor, arrayFillRgb);
for(int i = leftX; i <= rightX; i++){
for(int k = leftY; k <= rightY; k++){
if(checkValidCoord(i, k, height, width)){
                set_pixel_color(i, k, arrayFillRgb, newPng);
}
}
}
}

void drawSquare(int leftX, int leftY, int side_size, int thickness,
              std::string& color, bool fill, std::string fillColor, png_bytep*
&newPng, int &height, int &width){
    int rightX = leftX + side_size;
    int rightY = leftY + side_size;
if(fill == true) fillSquare(leftX, leftY, rightX, rightY, thickness, newPng,
fillColor, height, width);
drawLine(leftX, leftY, rightX, leftY, thickness, color, newPng, height, width);
drawLine(rightX, leftY, rightX, rightY, thickness, color, newPng, height,
width);
drawLine(leftX, rightY, rightX, rightY, thickness, color, newPng, height,
width);
drawLine(leftX, leftY, leftX, rightY, thickness, color, newPng, height, width);
}

void processSquareCommand(int &height, int &width, png_bytep* &newPng, int argc,
char* argv[]){
int opt;
int keyIndex;
int leftX = -1;
int leftY = -1;
int side_size = -1;
int thickness = -1;
std::string color = "";
std::string fillColor = "";
bool fill = false;
while(true){
opt = getopt_long(argc, argv, "", squareKeys, &keyIndex);
if(opt == -1){
opt = 0;
```

```
break;
}
switch(opt){
case 'u':
convertCoords(optarg, leftX, leftY);
break;
case 'd':
side_size = atoi(optarg);
break;
case 't':
if(std::regex_match(optarg, std::regex("([0-9]+)"))) thickness = atoi(optarg);
break;
case 'c':
if(std::regex_match(optarg, std::regex
("(25[0-5]|2[0-4][0-9]|[01]?[0-9]?[0-9]).(25[0-5]|2[0-4][0-9]|[01]?[0-9]?[0-
9]).(25[0-5]|2[0-4][0-9]|[01]?[0-9]?[0-9])")))
color = optarg;
break;
case 'f':
fill = true;
break;
case 'g':
if(std::regex_match(optarg, std::regex
("(25[0-5]|2[0-4][0-9]|[01]?[0-9]?[0-9]).(25[0-5]|2[0-4][0-9]|[01]?[0-9]?[0-
9]).(25[0-5]|2[0-4][0-9]|[01]?[0-9]?[0-9])")))
fillColor = optarg;
break;
}
}
if(leftX == -1 || leftY == -1 || side_size == -1 || thickness == -1 || color ==
""){
std::cout << "Invalid data" << std::endl;
exit(41);
return;
}
if(fill == true && fillColor == ""){
std::cout << "Invalid data" << std::endl;
exit(41);
return;
}
drawSquare(leftX, leftY, side_size, thickness, color, fill, fillColor, newPng,
height, width);
```

```
}

void processExchangeCommand(int &height, int &width, png_bytep* & newPng, int
argc, char* argv[]){
int opt;
int keyIndex;
int leftX = -1;
int leftY = -1;
int rightX = -1;
int rightY = -1;
std::string exchangeType;
while(true){
opt = getopt_long(argc, argv, "", exchangeKeys, &keyIndex);
if(opt == -1){
opt = 0;
break;
}
switch(opt){
case 'w':
convertCoords(optarg, rightX, rightY);
break;
case 'z':
convertCoords(optarg, leftX, leftY);
break;
case 'y':
exchangeType = optarg;
break;
}
}

if(leftX < 0 || leftY < 0 || rightX < 0 || rightY < 0 || rightX > width ||
rightY > height){
std::cout << "Invalid data" << std::endl;
exit(41);
return;
}

int midX = (leftX + rightX) / 2;
    int midY = (leftY + rightY) / 2;
    int exchange_width = rightX - leftX + 1;
    int exchange_height = rightY - leftY + 1;
```

```
    std::vector<png_bytep> topLeft(exchange_height / 2);
    std::vector<png_bytep> topRight(exchange_height / 2);
    std::vector<png_bytep> bottomLeft(exchange_height / 2);
    std::vector<png_bytep> bottomRight(exchange_height / 2);

    for (int i = 0; i < exchange_height / 2; ++i) {
        topLeft[i] = new png_byte[exchange_width / 2 * 4];
        topRight[i] = new png_byte[exchange_width / 2 * 4];
        bottomLeft[i] = new png_byte[exchange_width / 2 * 4];
        bottomRight[i] = new png_byte[exchange_width / 2 * 4];
    }

    for (int y = 0; y < exchange_height / 2; ++y) {
        std::copy(newPng[leftY + y] + leftX * 4, newPng[leftY + y] + midX * 4,
topLeft[y]);
        std::copy(newPng[leftY + y] + midX * 4, newPng[leftY + y] + rightX * 4,
topRight[y]);
        std::copy(newPng[midY + y] + leftX * 4, newPng[midY + y] + midX * 4,
bottomLeft[y]);
        std::copy(newPng[midY + y] + midX * 4, newPng[midY + y] + rightX * 4,
bottomRight[y]);
    }

    if (exchangeType == "clockwise") {
        for (int y = 0; y < exchange_height / 2; ++y) {
            std::copy(bottomLeft[y], bottomLeft[y] + exchange_width / 2 * 4,
newPng[leftY + y] + leftX * 4);
            std::copy(topLeft[y], topLeft[y] + exchange_width / 2 * 4,
newPng[leftY + y] + midX * 4);
            std::copy(bottomRight[y], bottomRight[y] + exchange_width / 2 * 4,
newPng[midY + y] + leftX * 4);
            std::copy(topRight[y], topRight[y] + exchange_width / 2 * 4,
newPng[midY + y] + midX * 4);
        }
    } else if (exchangeType == "counterclockwise") {
        for (int y = 0; y < exchange_height / 2; ++y) {
            std::copy(topRight[y], topRight[y] + exchange_width / 2 * 4,
newPng[leftY + y] + leftX * 4);
            std::copy(bottomRight[y], bottomRight[y] + exchange_width / 2 * 4,
newPng[leftY + y] + midX * 4);
```

```
            std::copy(topLeft[y], topLeft[y] + exchange_width / 2 * 4,
newPng[midY + y] + leftX * 4);
            std::copy(bottomLeft[y], bottomLeft[y] + exchange_width / 2 * 4,
newPng[midY + y] + midX * 4);
        }
    } else if (exchangeType == "diagonals") {
        for (int y = 0; y < exchange_height / 2; ++y) {
            std::copy(bottomRight[y], bottomRight[y] + exchange_width / 2 * 4,
newPng[leftY + y] + leftX * 4);
            std::copy(bottomLeft[y], bottomLeft[y] + exchange_width / 2 * 4,
newPng[leftY + y] + midX * 4);
            std::copy(topRight[y], topRight[y] + exchange_width / 2 * 4,
newPng[midY + y] + leftX * 4);
            std::copy(topLeft[y], topLeft[y] + exchange_width / 2 * 4,
newPng[midY + y] + midX * 4);
        }
    }
}

Rgb findMostFrequentColor(int &height, int &width, png_bytep* &newPng) {
    std::map<int, std::map<int, std::map<int, int>>> colorCount;

    for (int y = 0; y < height; ++y) {
        png_bytep row = newPng[y];
        for (int x = 0; x < width; ++x) {
            png_bytep px = &(row[x * 3]);
            int r = px[0];
            int g = px[1];
            int b = px[2];
            colorCount[r][g][b]++;
        }
    }
    int maxCount = 0;
    Rgb mostFrequentColor;
    for (const auto &r : colorCount) {
        for (const auto &g : r.second) {
            for (const auto &b : g.second) {
                if (b.second > maxCount) {
                    mostFrequentColor.R = r.first;
                    mostFrequentColor.G = g.first;
                    mostFrequentColor.B = b.first;
```

```
                    maxCount = b.second;
                }
            }
        }
    }
    return mostFrequentColor;
}

void processFreqCommand(int &height, int &width,png_bytep* & newPng, int argc,
char* argv[]){
int opt;
int keyIndex;
std::string color = "";
int arrayColor[3];
while(true){
opt = getopt_long(argc, argv, "", freqKeys, &keyIndex);
if(opt == -1){
opt = 0;
break;
}
switch(opt){
case 'l':
if(std::regex_match(optarg, std::regex
("(25[0-5]|2[0-4][0-9]|[01]?[0-9]?[0-9]).(25[0-5]|2[0-4][0-9]|[01]?[0-9]?[0-
9]).(25[0-5]|2[0-4][0-9]|[01]?[0-9]?[0-9])"))){
color = optarg;
getRgb(color, arrayColor);
}
break;
}
}
if(color == ""){
std::cout << "Invalid data\n";
exit(41);
}
Rgb freqColor = findMostFrequentColor(height, width, newPng);
for(int i = 0; i < width; i++){
for(int k = 0; k < height; k++){
if(newPng[k][4*i] == freqColor.R &&
newPng[k][4*i + 1] == freqColor.G &&
newPng[k][4*i + 2] == freqColor.B){
```

```
newPng[k][4*i] = arrayColor[0];
newPng[k][4*i + 1] = arrayColor[1];
newPng[k][4*i + 2] = arrayColor[2];
}
}
}

}

bool processCommand(png_bytep* &newPng, int &height, int &width,  png_byte
&colorType, png_byte &bitDepth, int argc, char* argv[], std::string &outputName,
std::string &inputName){
    int opt;
int keyIndex;
opterr = 0;
bool validFileName = false;
bool square = false;
bool exchange = false;
bool freq_color = false;
bool printInfo = false;
int count = 0;
char** argvCopy = new char*[argc];
    for (int i = 0; i < argc; ++i) {
        size_t len = strlen(argv[i]) + 1;
        argvCopy[i] = new char[len];
        strncpy(argvCopy[i], argv[i], len);
    }
if(argc == 1) outputHelp();
while(true){
opt = getopt_long(argc, argvCopy, "ho:i", selectAction, &keyIndex);
if(opt == -1){
opt = 0;
break;
}
switch(opt){
case 'h':
outputHelp();
break;
case 'o':
if(checkValidName(optarg)) outputName = optarg;
break;
```

```
case 'i':
if(checkValidName(optarg)){
                    inputName = optarg;
if(read_png_file(inputName, newPng, height, width, colorType, bitDepth))
validFileName = true;
}
break;
case 'd':
printInfo = true;
break;
case 'r':
square = true;
count++;
break;
case 'x':
exchange = true;
count++;
break;
case 'c':
freq_color = true;
count++;
break;
}
}
if(printInfo){
if(validFileName) outputInfo(height, width);
else{
std::cout << "Error: the input file name is invalid or the input file is
corrupted" << std::endl;
exit(41);
}
}
optind = 1;
if(count > 1){
std::cout << "Error: too many arguments" << std::endl;
exit(41);
}
else{
if(validFileName&&square) processSquareCommand(height, width, newPng, argc,
argv);
else if(validFileName&&exchange) processExchangeCommand(height, width, newPng,
argc, argv);
```

```
else if(validFileName&&freq_color) processFreqCommand(height, width, newPng,
argc, argv);
else if(square || exchange || freq_color){
std::cout << "Invalid input file name" << std::endl;
exit(41);
}
}
return validFileName;
};

int main(int argc, char* argv[]){
    png_bytep* newPng = nullptr;
    int height;
    int width;
    png_byte colorType;
    png_byte bitDepth;
char filename[] = "fd.png";
std::string outputName = "out.png";
std::string inputName = "";
std::cout << CW << std::endl;
if(findUnknownKey(argc, argv)){
exit(41);
return 0;
}
if(processCommand(newPng, height, width, colorType, bitDepth, argc, argv,
outputName, inputName)){
      if(outputName == inputName){
std::cout << "Error: the names of the input and output files are the same" <<
std::endl;
exit(41);
} else write_png_file(outputName, newPng, height, width);
  return 0;
  }
}
```

ПРИЛОЖЕНИЕ Б
ТЕСТИРОВАНИЕ

Рисование квадрата: `./a.out --square --left_up 220.300 --side_size 70 --thickness 10 --color 255.20.147 --input FunnyMonkey.png --fill_color 255.20.147 --fill`

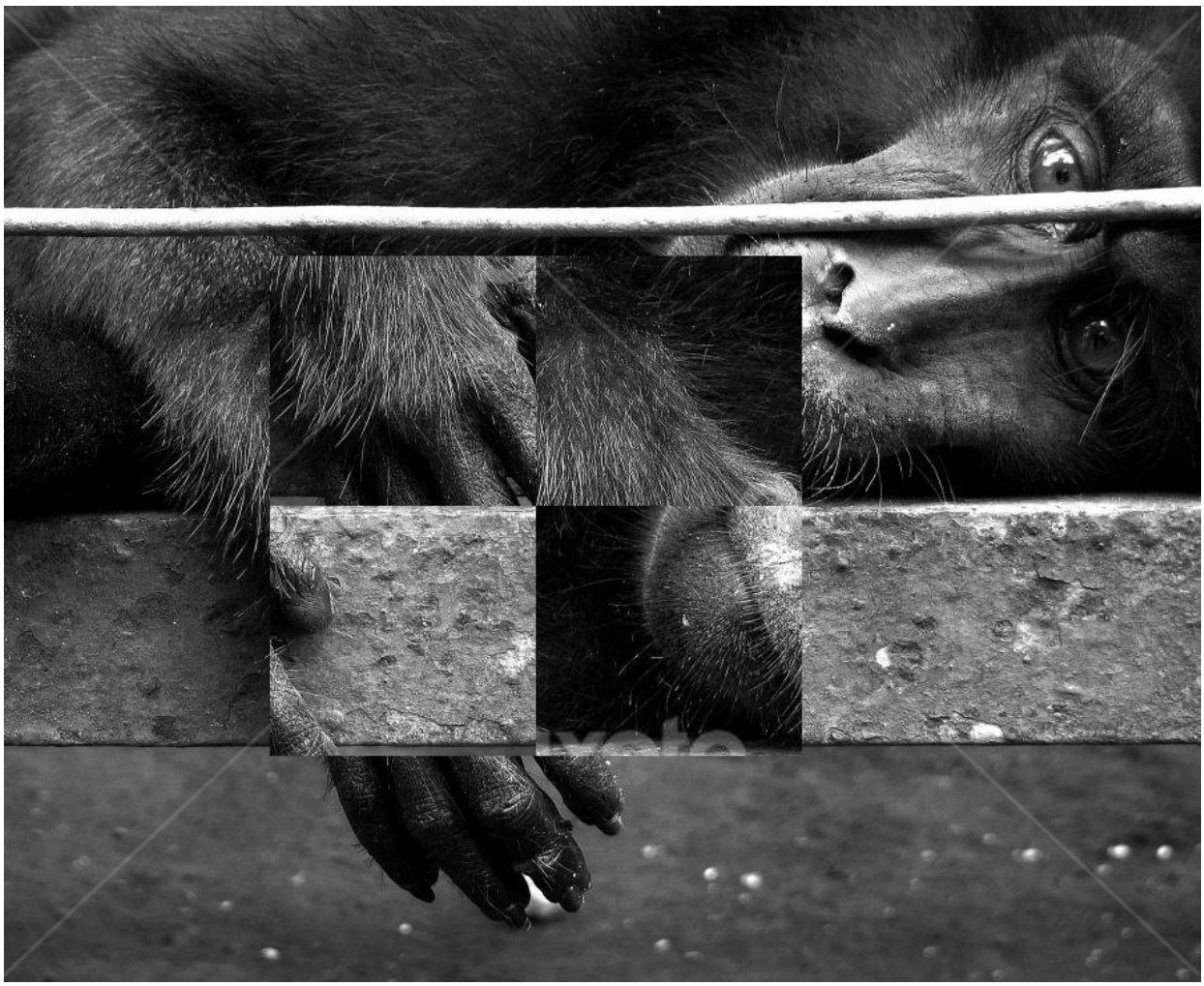
Рисунок 1. Входное изображение

Рисунок 2. Выходное изображение

Отражение области: `./a.out --exchange --left_up 200.200 --right_down 600.600 --exchange_type clockwise --input SadMonkey.png`

Рисунок 3. Входное изображение

Рисунок 4. Выходное изображение



Замена цвета: `./a.out --freq_color --color 0.0.255 --input CuteMonkey.png`

Рисунок 5. Входное изображение

Рисунок 6. Выходное изображение