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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Обработка изображений

Студент гр. 3341 _____ Трофимов В.О.

Преподаватель _____ Глазунов С.А.

Санкт-Петербург
2024

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студент Трофимов В.О.

Группа 3341

Тема работы: Обработка изображений

Вариант 21

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

(3) Копирование заданной области. Флаг для выполнения данной операции: `--copy`. Функционал определяется: Координатами левого верхнего угла области-источника. Флаг `--left_up`, значение задаётся в формате `left.up`, где left – координата по x, up – координата по y. Координатами правого нижнего угла области-источника. Флаг `--right_down`, значение задаётся в формате `right.down`, где right – координата по x, down – координата по y
Координатами левого верхнего угла области-назначения. Флаг `--dest_left_up`, значение задаётся в формате `left.up`, где left – координата по x, up – координата по y

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

Дата выдачи задания: 16.10.2023

Дата сдачи реферата: 17.12.2023

Дата защиты реферата: 23.05.2024

Студент _____ Трофимов В.О.

Преподаватель _____ Глазунов С.А.

_____

АННОТАЦИЯ

Данный проект предполагает разработку программы для обработки изображений в формате PNG с использованием библиотеки libpng. Программа должна функционировать через интерфейс командной строки (CLI) и может опционально поддерживать графический интерфейс (GUI). Основные задачи программы включают: Проверка формата входного файла на соответствие PNG с выводом ошибки при несоответствии; Обеспечение выравнивания данных в файле, где мусорные данные заполняются нулями; Сохранение всех стандартных полей заголовков PNG в выходном файле неизменными, за исключением требуемых модификаций; Программа должна выполнять следующие функции по обработке изображений: Рисование прямоугольника с заданными координатами, толщиной и цветом линий, опционально заполненного цветом, рисование правильного шестиугольника с заданными координатами центра, радиусом, толщиной и цветом линий, опционально заполненного цветом, а также копирование заданной области изображения в указанное место.

Все операции должны быть реализованы в виде отдельных функций

Исходный код программы: Приложение А.

Демонстрация работы программы: Приложение В.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Формат файлов PNG широко используется на веб-сайтах для отображения высококачественных цифровых изображений. Созданный для того, чтобы превзойти по производительности файлы GIF, PNG обеспечивает не только сжатие без потерь, но и более широкую и яркую цветовую палитру.

PNG – это сокращение от Portable Network Graphic, тип файла растрового изображения. Этот тип файлов особенно популярен среди вебдизайнеров, поскольку он позволяет работать с графикой с прозрачным или полупрозрачным фоном.

PNG – это следующая эволюция формата GIF, который к моменту появления PNG существовал уже восемь лет. У GIF было несколько недостатков, таких как необходимость получения патентной лицензии и ограниченный диапазон – всего 256 цветов, что не соответствовало постоянно растущему разрешению экрана компьютера. Чтобы избежать этих проблем, файлы PNG были сделаны свободными от патентов и включали значительно большую палитру цветов.

Целью курсовой работы является изучение форматов файлов BMP и PNG, а также реализация функций для работы с этими форматами файлов.

Для достижения поставленной цели требуется решить следующие задачи:

1) изучить BMP и PNG форматы изображений;
2) получить информацию об изображении: размеры, содержимое и др.;
3) обработать массив пикселей в соответствии с заданием;
4) обработать исключительные случаи;

сохранить итоговое изображение в новый файл.

ХОД ВЫПОЛНЕНИЯ РАБОТЫ

1. Класс ImageFileReader

Класс ImageFileReader предназначен для чтения, обработки и записи PNG-файлов. Он включает следующие компоненты:

Поля:

width и height - ширина и высота изображения.

color_type и bit_depth - тип цвета и глубина цвета изображения.

number_of_passes - количество проходов при обработке изображения.

png_ptr и info_ptr - указатели на структуры PNG, используемые библиотекой libpng для работы с файлами PNG.

row_pointers - указатель на массив строк изображения.

Методы:

Приватные методы:

allocateRowPointers() - выделяет память для строк изображения.

checkFileHeader(FILE* file) - проверяет заголовок файла на соответствие формату PNG.

getPngParam() - получает параметры изображения(значение полей класса) из файла PNG.

transforImage() - преобразует изображение для добавления альфа-канала, если он отсутствует.

Публичные методы:

ImageFileReader() - конструктор, инициализирующий объект класса.

~ImageFileReader() - деструктор, освобождающий память.

resolve(const char* filename) - читает изображение из файла.

getWidth(), getHeight(), getColortype(), getBitDepth() - возвращают соответствующие параметры изображения.

write(const char* filename) - записывает изображение в файл.

savePixel(int x, int y, Rgbacolour colour) - устанавливает пиксель в изображении.

getPixel(int x, int y) - получает цвет пикселя из изображения.

drawCircleBorder(int x0, int y0, int radius, Rgbacolour color) и drawCircleArea(int x0, int y0, int radius, Rgbacolour color) - рисуют границу и область внутри круга, не включая границ.

drawLine(int beginX, int beginY, int endX, int endY, int thickness, Rgbacolour color) - рисует линию алгоритмом Брезенхема.

swapCoordinates(int& leftX, int& leftY, int& rightX, int& rightY) - меняет местами координаты, возможно перепутанные пользователем при вводе их.

drawBorderRect(int leftX, int leftY, int rightX, int rightY, Rgbacolour color, int thickness, bool fill, Rgbacolour fill_color) - рисует границу прямоугольника.

drawAreaRect(int leftX, int leftY, int rightX, int rightY, Rgbacolour fill_color) - рисует залитую область внутри прямоугольника.

getCorners(int x0, int y0, int radius) - получает координаты вершин шестиугольника.

drawHexagon(int x0, int y0, int radius, Rgbacolour color, int thickness, bool fill, Rgbacolour fill_color) - рисует шестиугольник с различными параметрами.

coordinateInHexagon(int x, int y, const std::vector<std::pair<int, int>>& corners) - проверяет, находится ли точка внутри шестиугольника.

copyArea(int leftX, int leftY, int rightX, int rightY, int destX, int destY) - копирует область изображения.

2. Структуры данных

Структура Flag представляет собой флаг, который содержит два поля: inputed - логическое значение, указывающее, был ли флаг установлен; value - строка, содержащая значение флага.

Структура Rgbacolour представляет собой цвет в формате RGBA, который содержит следующие поля:

1. red - значение красного компонента цвета.
2. green - значение зеленого компонента цвета.
3. blue - значение синего компонента цвета.
4. alpha - значение альфа-канала (прозрачности).

3. Обработка CLI входных данных

3.1. std::pair<int, int> processCoordinatesCli(std::string string_coordinates) - функция обрабатывает строку координат, переданную через командную строку(CLI), и возвращает пару целых чисел, представляющих координаты (x, y). Формат входной строки должен быть "x.y", где x и y - целые числа, который проверяется с помощью регулярного выражения.

3.2. int processNumberCli(std::string string_number) - функция обрабатывает строку, представляющую целое число, переданное через CLI, и возвращает это число. Если строка не соответствует формату (регулярному выражению) целого числа или число меньше либо равно нулю, функция выводит сообщение об ошибке и завершает программу.

3.3. void processColorCli(Rgbacolour& color,std::string string_color) - функция обрабатывает строку, представляющую цвет в формате "rrr.ggg.bbb", где rrr, ggg и bbb - целые числа от 0 до 255, и устанавливает соответствующие значения полей структуры Rgbacolour. Если строка не соответствует ожидаемому формату (регулярному выражению) или значения выходят за пределы диапазона, функция выводит сообщение об ошибке и завершает программу.

3.4. void processCLI(int argc, char** argv) - Функция processCLI обрабатывает аргументы командной строки, которые передаются программе. Она создает набор флагов, каждый из которых может быть использован для хранения информации о том, была ли введена определенная опция, и если да, то какое значение ей было передано. Определяет, какие короткие и длинные опции командной строки программа может принимать. Короткие опции выглядят как -i, длинные как --input.В цикле обрабатываются все переданные опции командной строки: для каждой опции проверяется, что существует ли она в наборе флагов. Если опция найдена, помечает её как введенную и сохраняет её значение, если оно есть. Если опция не найдена, выводит сообщение об ошибке и завершает программу. В случае если пользователь не введёт название входного

изображения принято считать последний аргумент названием входного изображения, поэтому сохраняем последний аргумент командной строки в отдельную переменную. После вызывается функция, selectionFunction, передавая ей обработанные флаги и последний аргумент командной строки для дальнейшей обработки.

3.5. В функциях bool flags (rectangle, hexagon, copy, help, info) - функция проверяет, корректно ли заданы флаги для команды. Рассмотрим на примере прямоугольника. Обязательные флаги: RECT_INDEX, LEFT_UP_INDEX, RIGHT_DOWN_INDEX, THICKNESS_INDEX, COLOR_INDEX.
Дополнительные флаги: INPUT_INDEX, OUTPUT_INDEX, FILL_INDEX, FILL_COLOR_INDEX. Дальше функция вызывает getUnusableFlags для получения набора флагов, которые нельзя использовать с этими опциями, также происходит проверкан на правильность комбинации флагов с помощью checkValidFlags и проверка на то, что если задан флаг заливки (FILL_INDEX), то должен быть задан и флаг цвета заливки (FILL_COLOR_INDEX). Таким образом и для остальных команд прописаны функции с наборами флагов.

3.6. std::set<int> getUnusableFlags(std::map<int, Flag> flags, std::set<int> necessaryFlags, std::set<int> functionalFlags) - функция определяет флаги, которые не должны использоваться для определенной команды.

3.7. bool checkValidFlags(std::map<int, Flag> flags, std::set<int> necessaryFlags, std::set<int> unusableFlags) - функция проверяет, правильно ли заданы необходимые и недопустимые флаги.

3.8. void proccessFileOutname(std::map<int, Flag> flags, std::string& inputFilename, std::string& outputFilename, std::string lastArgument) - функция определяет входное и выходное имя файла на основе переданных флагов и последнего аргумента командной строки.

4. Вспомогательные функции

4.1. void description() – функция выводит в консоль краткую информацию о курсовой работе.

4.2. void drawRectangle(std::map<int, Flag> flags, std::string lastArgument) – функция рисует прямоугольник на изображении.

4.3. void drawHexagon(std::map<int, Flag> flags, std::string lastArgument) – функция рисует шестиугольник на изображении.

4.4. void copyArea(std::map<int, Flag> flags, std::string lastArgument) – функция копирует заданную область на изображении.

4.5. void helpUser() - функция выводит справочную информацию о возможностях программы.

4.6. void infoImage(std::map<int, Flag> flags, std::string lastArgument) - функция выводит информацию об изображении.

4.7. void selectionFunction(std::map<int, Flag> flags, std::string lastArgument) – функция определяет, какая команда должна быть выполнена на основе переданных флагов, и вызывает соответствующую функцию для выполнения этой команды. Если ни одна из известных команд не соответствует введенным флагам, она выводит сообщение об ошибке и завершает программу.

5. Makefile

Makefile представляет собой файл без расширения, который используется для сборки всех файлов в один проект Makefile использован для компиляции и создания исполняемого файла cw. Для каждого созданного фалйа с расширением *.c проводится компиляция после которой создаются объектные файлы с тем же названием, но с расширением *.o. Выглядит этот процесс в Makefile следующим образом:

```
name_file.o: name_file.c
@ gcc -std=gnu99 -c name_file.c
```

@ - использован для, того чтобы при сборке не отображалась компиляция каждого файла.

После все эти объектные файлы объединятся в исполняемый файл cw выглядит этот процесс следующим образом:

```
cw: name_file.o
    gcc name_file.o –o cw
    rm *.o
```

rm *.o – удаляет все объектный файлы созданные в ходе сборки.

ЗАКЛЮЧЕНИЕ

Разработанная программа представляет собой мощный инструмент для обработки изображений формата PNG, обеспечивающий гибкое управление через командную строку (CLI). Она включает следующие функции:

1. Рисование прямоугольника: Возможность задавать координаты углов, толщину линий, цвет и наличие заливки, что позволяет пользователям точно настраивать внешний вид прямоугольника.

2. Рисование шестиугольника: Определение через центр и радиус, с возможностью настройки цвета, толщины линий и заливки, что дает пользователям широкие возможности для кастомизации шестиугольника.

3. Копирование области: Позволяет копировать произвольную область изображения в другое место, что особенно полезно для манипуляций с изображениями.

Программа тщательно проверяет соответствие входного файла формату PNG, обеспечивая обработку только корректных данных. Если файл не соответствует формату PNG, программа завершает работу с сообщением об ошибке. Также программа следит за выравниванием данных, дополняя мусорные данные нулями, что гарантирует корректность выходного файла. Для удобства пользователей предусмотрена функция вывода справочной информации, объясняющая использование всех доступных команд и их флагов. Эта информация помогает пользователям эффективно использовать возможности программы.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКО*В*

Авторы: М. М. Заславский, А. А. Лисс, А. В. Гаврилов, С. А. Глазунов, Я. С. Государкин, С. А. Тиняков, В. П. Голубева, К. В. Чайка, В. Е. Допира.

Б17 Базовые сведения к выполнению курсовой работы по дисциплине «Программирование». Второй семестр: учеб.-метод. пособие. СПб.: Изд-во СПбГЭТУ «ЛЭТИ», 2024. 36 с.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include <iostream>
#include <cmath>
#include <png.h>
#include <algorithm>
#include <vector>
#include <map>
#include <set>
#include <getopt.h>
#include <regex>

#define HELP_INDEX 'h'
#define OUTPUT_INDEX 'o'
#define INPUT_INDEX 'i'
#define INFO_INDEX 1000
#define RECT_INDEX 1001
#define LEFT_UP_INDEX 1002
#define RIGHT_DOWN_INDEX 1003
#define THICKNESS_INDEX 1004
#define COLOR_INDEX 1005
#define FILL_INDEX 1006
#define FILL_COLOR_INDEX 1007
#define HEXAGON_INDEX 1008
#define CENTER_INDEX 1009
#define RADIUS_INDEX 1010
#define DEST_LEFT_UP_INDEX 1011
#define COPY_INDEX 1012

struct Flag {
    bool inputed;
    std::string value;
};

struct Rgbacolour {
    int red;
    int green;
    int blue;
    int alpha;
```

```
        Rgbacolour(png_byte r, png_byte g, png_byte b, png_byte a = 255)
{
            red = r;
            green = g;
            blue = b;
            alpha = a;
        }

        Rgbacolour() : red(0), green(0), blue(0), alpha(255) {}
    };

    class ImageFileReader {
    private:
        int width;
        int height;
        png_byte color_type;
        png_byte bit_depth;
        int number_of_passes;
        png_structp png_ptr;
        png_infop info_ptr;
        png_bytep *row_pointers;

        void allocateRowPointers() {
            row_pointers   =   (png_bytep*)malloc(sizeof(png_bytep)   *
height);
            if (!row_pointers) {
                std::cout << "Error ImageRead: Can't allocate memory";
                exit(41);
            }
            for (int y = 0; y < height; y++) {
                row_pointers[y]              =              (png_byte*)
malloc(png_get_rowbytes(png_ptr, info_ptr));
            }
        }

        static void checkFileHeader(FILE* file) {
            png_byte header[8];
            fread(header, sizeof(png_byte), 8, file);
            if (png_sig_cmp(header, 0, 8)){
                std::cout  <<  "Error:  File  isn't  have  png  format"  <<
std::endl;
                fclose(file);
                exit(41);
            }
```

```
    }

    void getPngParam() {
        width = png_get_image_width(png_ptr, info_ptr);
        height = png_get_image_height(png_ptr, info_ptr);
        bit_depth = png_get_bit_depth(png_ptr, info_ptr);
        color_type = png_get_color_type(png_ptr, info_ptr);
        number_of_passes = png_set_interlace_handling(png_ptr);
    }

    void transforImage() {
        if (color_type == PNG_COLOR_TYPE_RGB ||
        color_type == PNG_COLOR_TYPE_GRAY ||
        color_type == PNG_COLOR_TYPE_PALETTE) {
            png_set_filler(png_ptr, 0xFF, PNG_FILLER_AFTER);
            png_set_add_alpha(png_ptr, 0xFF, PNG_FILLER_AFTER);
        }
    }
public:
    ImageFileReader() {
        width = 0;
        height = 0;
        color_type = 0;
        bit_depth = 0;
        number_of_passes = 0;
        png_ptr = nullptr;
        info_ptr = nullptr;
        row_pointers = nullptr;
    }

    ~ImageFileReader() {
        if (row_pointers != nullptr) {
            for (int y = 0; y < height; y++) {
                free(row_pointers[y]);
            }
            free(row_pointers);
        }
        if (png_ptr != nullptr) {
            png_destroy_read_struct(&png_ptr, &info_ptr, nullptr);
        }
    }

    void resolve(const char* filename) {
        FILE* input_png = fopen(filename, "rb");
```

```
            if (!input_png){
                std::cout << "Cannot read file: " << filename <<
std::endl;
                exit(41);
            }
            checkFileHeader(input_png);

            png_ptr = png_create_read_struct(PNG_LIBPNG_VER_STRING,
nullptr, nullptr, nullptr);
            if (!png_ptr) {
                std::cout << "Error in png structure" << std::endl;
                fclose(input_png);
                exit(41);
            }

            info_ptr = png_create_info_struct(png_ptr);
            if (!info_ptr){
                std::cout << "Error in png info-structure" << std::endl;
                png_destroy_read_struct(&png_ptr, (png_infopp)nullptr,
(png_infopp)nullptr);
                fclose(input_png);
                exit(41);
            }

            if (setjmp(png_jmpbuf(png_ptr))){
                png_destroy_read_struct(&png_ptr, &info_ptr,
(png_infopp)nullptr);
                fclose(input_png);
                exit(41);
            }
            png_init_io(png_ptr, input_png);
            png_set_sig_bytes(png_ptr, 8);
            png_read_info(png_ptr, info_ptr);
            getPngParam();
            transforImage();
            png_read_update_info(png_ptr, info_ptr);
            getPngParam();
            number_of_passes = png_set_interlace_handling(png_ptr);
            allocateRowPointers();
            png_read_image(png_ptr, row_pointers);
            fclose(input_png);
        }

        int getWidth() {
```

```
            return width;
        }

        int getHeight() {
            return height;
        }

        int getColortype() {
            return color_type;
        }

        int getBitDepth() {
            return bit_depth;
        }

        void write(const char* filename) {
            FILE *output_png = fopen(filename, "wb");
            png_structp                    png_ptr                    =
png_create_write_struct(PNG_LIBPNG_VER_STRING,     nullptr,     nullptr,
nullptr);
            if (!png_ptr) {
                std::cout << "Error in write png structure" << std::endl;
                fclose(output_png);
                exit(41);
            }

            png_infop info_ptr = png_create_info_struct(png_ptr);
            if (!info_ptr) {
                std::cout  <<  "Error  in  write  info-png  structure"  <<
std::endl;
                fclose(output_png);
                exit(41);
            }
            if (setjmp(png_jmpbuf(png_ptr))) {
                fclose(output_png);
                exit(41);
            }

            png_init_io(png_ptr, output_png);
            if (setjmp(png_jmpbuf(png_ptr))) {
                std::cout << "Error to write a header" << std::endl;
                fclose(output_png);
                exit(41);
            }
```

```
            png_set_IHDR(png_ptr, info_ptr, width, height, bit_depth,
color_type,
                         PNG_INTERLACE_NONE,
PNG_COMPRESSION_TYPE_DEFAULT, PNG_FILTER_TYPE_DEFAULT);
            png_write_info(png_ptr, info_ptr);
            if (setjmp(png_jmpbuf(png_ptr))) {
                fclose(output_png);
                exit(41);
            }
            png_write_image(png_ptr, row_pointers);
            if (setjmp(png_jmpbuf(png_ptr))) {
                fclose(output_png);
                exit(41);
            }
            png_write_end(png_ptr, nullptr);
            png_destroy_write_struct(&png_ptr, &info_ptr);
            fclose(output_png);
        }

        void savePixel(int x, int y, Rgbacolour colour) {
            if (!(x >= 0 && x < width && y >= 0 && y < height)) return;
            png_bytep pixel = &(row_pointers[y][x * 4]);
            pixel[0] = colour.red;
            pixel[1] = colour.green;
            pixel[2] = colour.blue;
            pixel[3] = colour.alpha;
        }

        Rgbacolour getPixel(int x, int y) {
            Rgbacolour colourPixel(0,0,0);
            if (!((x >= 0 && x < width && y >= 0 && y < height))) {
                return colourPixel;
            }
            colourPixel.red = row_pointers[y][x * 4 + 0];
            colourPixel.green = row_pointers[y][x * 4 + 1];
            colourPixel.blue = row_pointers[y][x * 4 + 2];
            colourPixel.alpha = row_pointers[y][x *4  + 3];
            return colourPixel;
        }

        void drawCircleBorder(int x0, int y0, int radius, Rgbacolour
color) {
            int x = 0;
            int y = radius;
```

```
            int delta = 3 - 2 * y;
            while (x <= y) {
                savePixel(x0 + x, y0 + y, color);
                savePixel(x0 + x, y0 - y, color);
                savePixel(x0 - x, y0 + y, color);
                savePixel(x0- x, y0 - y, color);
                savePixel(x0 + y, y0 + x, color);
                savePixel(x0 + y, y0 - x, color);
                savePixel(x0 - y, y0 + x, color);
                savePixel(x0 - y, y0 - x, color);
                delta += delta < 0 ? 4 * x + 6 : 4 * (x - y--) + 10;
                ++x;
            }
        }

        void drawCircleArea(int x0, int y0, int radius, Rgbacolour color)
{
            for (int y = std::max(0, y0 - radius); y <= std::min(height
- 1, y0 + radius); y++) {
                for (int x = std::max(0, x0 - radius); x <= std::min(width
- 1, x0 + radius); x++) {
                    if ((x - x0) * (x - x0)  + (y - y0) * (y - y0)  <=
radius * radius) {
                        savePixel(x, y, color);
                    }
                }
            }
        }

        void drawLine(int beginX, int beginY, int endX, int endY, int
thickness, Rgbacolour color) {
            int absDeltaX = abs(endX - beginX);
            int absDeltaY = abs(endY - beginY);
            int signedDeltaX = beginX < endX ? 1: -1;
            int signedDeltaY = beginY < endY ? 1: -1;
            int err = absDeltaX - absDeltaY;
            int currentX = beginX;
            int currentY = beginY;
            while (currentX != endX || currentY != endY) {
                drawCircleArea(currentX,  currentY,  thickness  /  2,
color);
                drawCircleBorder(currentX,  currentY,  thickness  /  2,
color);
                int doubleError = 2 * err;
```

```
                if (doubleError > -absDeltaY) {
                    err -= absDeltaY;
                    currentX += signedDeltaX;
                }
                if (doubleError < absDeltaX) {
                    err += absDeltaX;
                    currentY += signedDeltaY;
                }
            }
            drawCircleBorder(endX, endY, thickness / 2, color);
            drawCircleArea(endX, endY, thickness /2, color);
        }

        void swapCoordinates(int& leftX, int& leftY, int& rightX, int&
rightY) {
            int max_x = std::max(leftX, rightX);
            int min_x = std::min(leftX, rightX);
            int max_y = std::max(leftY, rightY);
            int min_y = std::min(leftY, rightY);
            leftX = min_x;
            leftY = min_y;
            rightX = max_x;
            rightY = max_y;
        }

        void drawBorderRect(int leftX, int leftY, int rightX, int rightY,
Rgbacolour color, int thickness, bool fill, Rgbacolour fill_color){
            swapCoordinates(leftX,leftY,rightX,rightY);
            if  (fill)  drawAreaRect(leftX,  leftY,  rightX,  rightY,
fill_color);
            drawLine(leftX, leftY, rightX, leftY, thickness, color);
            drawLine(leftX, leftY, leftX, rightY, thickness, color);
            drawLine(leftX, rightY, rightX, rightY, thickness, color);
            drawLine(rightX, leftY, rightX, rightY, thickness, color);
        }

        void drawAreaRect(int leftX, int leftY, int rightX, int rightY,
Rgbacolour fill_color) {
            swapCoordinates(leftX,leftY,rightX,rightY);
            for (int y = leftY + 1; y < rightY;y++){
                for (int x = leftX + 1; x < rightX;x++){
                    savePixel(x, y, fill_color);
                }
            }
```

```
        }

        std::vector<std::pair<int, int>> getCorners(int x0, int y0, int
radius) {
            std::vector<std::pair<int, int>> corners;
            for (int i = 0; i < 6;i++){
                corners.push_back({x0 + radius * std::cos(i * (M_PI * 2
/ 6)), y0 + radius * std::sin(i * (M_PI * 2 / 6))});
            }
            return corners;
        }

        void drawHexagon(int x0, int y0, int radius, Rgbacolour color,
int thickness, bool fill, Rgbacolour fill_color) {
            std::vector<std::pair<int, int>> corners = getCorners(x0,
y0, radius);
            if (fill) {
                for (int y = std::max(0, y0 - radius); y <=
std::min(height - 1, y0 + radius);y++) {
                    for (int x = std::max(0, x0 - radius); x <=
std::min(width - 1, x0 + radius);x++) {
                        if (coordinateInHexagon(x, y, corners)) {
                            savePixel(x, y, fill_color);
                        }
                    }
                }
            }
            for (int i = 0; i < 6; i++) {
                drawLine(corners[i].first, corners[i].second, corners[(i
+ 1) % 6].first, corners[(i + 1) % 6].second, thickness, color);
            }
        }

        int coordinateInHexagon(int x, int y, const
std::vector<std::pair<int, int>>& corners) {
            int intersectionCount = 0;
            for (int i = 0; i < 6; i++) {
                int next = (i + 1) % 6;
                if ((corners[i].second > y && corners[next].second <= y)
|| (corners[next].second > y && corners[i].second <= y)) {
                    if ((corners[next].second - corners[i].second) < 0)
{
                        if ((x * (corners[next].second -
corners[i].second)) < ((corners[next].second - corners[i].second) *
```

```
corners[i].first + (y - corners[i].second) * (corners[next].first -
corners[i].first))) {
                            intersectionCount++;
                        }
                }
                else {
                    if      ((x      *      (corners[next].second      -
corners[i].second))  >  ((corners[next].second  -  corners[i].second)  *
corners[i].first + (y - corners[i].second) * (corners[next].first -
corners[i].first))) {
                        intersectionCount++;
                    }
                }
            }
            }
            return (intersectionCount % 2);
        }

         void copyArea(int leftX, int leftY, int rightX, int rightY, int
destX, int destY) {
            swapCoordinates(leftX, leftY, rightX, rightY);
            int copyDimensionX = rightX - leftX + 1;
            int copyDimensionY = rightY - leftY + 1;
            int maxCopyWidth = std::min(width, copyDimensionX);
            int maxCopyHeight = std::min(height, copyDimensionY);
            std::vector<std::vector<Rgbacolour>>  colors(maxCopyHeight,
std::vector<Rgbacolour>(maxCopyWidth, Rgbacolour(0, 0, 0)));
            for (int y = 0; y < maxCopyHeight; y++) {
                for (int x = 0; x < maxCopyWidth; x++) {
                    int sourceX = x + leftX;
                    int sourceY = y + leftY;
                    if (sourceX >= 0 && sourceX < width && sourceY >= 0
&& sourceY < height) {
                        colors[y][x] = getPixel(sourceX, sourceY);
                    }
                }
            }
            for (int y = 0; y < maxCopyHeight; y++) {
                for (int x = 0; x < maxCopyWidth; x++) {
                    int destinationX = x + destX;
                    int destinationY = y + destY;
                    if (destinationX >= 0 && destinationX < width &&
destinationY >= 0 && destinationY < height) {
```

```
                        savePixel(destinationX,           destinationY,
colors[y][x]);
                    }
                }
            }
            colors.clear();
        }
    };

    void description(){
        std::cout << "Course work for option 4.21, created by Vladislav
Trofimov." << std::endl;
    }

    std::pair<int,        int>        processCoordinatesCli(std::string
string_coordinates) {
        std::pair<int, int> coordinates;
        std::cmatch groups;
        std::regex pattern("(-?\\d+)\\.(-?\\d+)");
        if    (!std::regex_match(string_coordinates.c_str(),    groups,
pattern)) {
            std::cout << "CLI Error: Coordinates not matched x.y" <<
std::endl;
            exit(40);
        }
        coordinates = {std::stoi(groups[1]) , std::stoi(groups[2])};
        return coordinates;
    }

    int processNumberCli(std::string string_number) {
        std::cmatch groups;
        std::regex pattern("(-?\\d+)");
        if (!std::regex_match(string_number.c_str(), groups, pattern)) {
            std::cout << "CLI Error: Number not matched with format" <<
std::endl;
            exit(40);
        }
        int number = std::stoi(groups[1]);
        if (number <= 0) {
            std::cout << "CLI Error: Number not processed" << std::endl;
            exit(40);
        }
        return number;
    }
```

```
    void processColorCli(Rgbacolour& color,std::string string_color) {
        std::cmatch groups;
        std::regex pattern("(\\d+)\\.(\\d+)\\.(\\d+)");
        if (!std::regex_match(string_color.c_str(), groups, pattern)) {
            std::cout << "CLI Error: Colour not matched with format
rrr.ggg.bbb" << std::endl;
            exit(40);
        }
        color.red = std::stoi(groups[1]);
        color.green = std::stoi(groups[2]);
        color.blue = std::stoi(groups[3]);
        color.alpha = 255;
        if ((0 > color.red || color.red > 255) || (0 > color.green ||
color.green > 255) || (0 > color.blue || color.blue > 255)){
            std::cout << "CLI Error: Color value does not match the
expected format" << std::endl;
            exit(40);
        }
    }

    std::set<int>    getUnusableFlags(std::map<int,    Flag>    flags,
std::set<int> necessaryFlags, std::set<int> functionalFlags) {
        std::set<int> result;
        for (auto i : flags){
            if (necessaryFlags.find(i.first) == necessaryFlags.end() &&
functionalFlags.find(i.first) == functionalFlags.end()){
                result.insert(i.first);
            }
        }
        return result;
    }

    bool   checkValidFlags(std::map<int,   Flag>   flags,   std::set<int>
necessaryFlags, std::set<int> unusableFlags) {
        int all_necessaryFlags = 0;
        for (auto i : necessaryFlags){
            if(flags[i].inputed) {
                all_necessaryFlags++;
            }
        }

        int all_unusableFlags = 0;
        for (auto i : unusableFlags) {
```

```
            if(!flags[i].inputed) {
                all_unusableFlags++;
            }
        }

        return (all_necessaryFlags == (int)necessaryFlags.size() &&
all_unusableFlags == (int)unusableFlags.size());
    }

    void proccessFileOutname(std::map<int, Flag> flags, std::string&
inputFilename, std::string& outputFilename, std::string lastArgument) {
        if (flags[INPUT_INDEX].inputed) {
            inputFilename = flags[INPUT_INDEX].value;
        } else {
            inputFilename = lastArgument;
        }
        if (flags[OUTPUT_INDEX].inputed){
            outputFilename = flags[OUTPUT_INDEX].value;
        }
    }

    void drawRectangle(std::map<int, Flag> flags, std::string
lastArgument) {
        std::pair<int, int> left_up =
processCoordinatesCli(flags[LEFT_UP_INDEX].value);
        std::pair<int, int> right_down =
processCoordinatesCli(flags[RIGHT_DOWN_INDEX].value);
        Rgbacolour fill_color(0,0,0);
        Rgbacolour color(0,0,0);
        int thickness = processNumberCli(flags[THICKNESS_INDEX].value);
        processColorCli(color, flags[COLOR_INDEX].value);
        std::string inputFilename;
        std::string outputFilename = "out.png";
        proccessFileOutname(flags, inputFilename, outputFilename,
lastArgument);
        bool fill = flags[FILL_INDEX].inputed;
        if (fill)
            processColorCli(fill_color, flags[FILL_COLOR_INDEX].value);
        ImageFileReader image;
        image.resolve(inputFilename.c_str());
        image.drawBorderRect(left_up.first, left_up.second,
right_down.first, right_down.second, color, thickness, fill, fill_color);
        image.write(outputFilename.c_str());
    }
```

```
    void drawHexagon(std::map<int, Flag> flags, std::string
lastArgument) {
        std::pair<int, int> center =
processCoordinatesCli(flags[CENTER_INDEX].value);
        int radius = processNumberCli(flags[RADIUS_INDEX].value);
        int thickness = processNumberCli(flags[THICKNESS_INDEX].value);
        Rgbacolour color(0,0,0);
        Rgbacolour fill_color(0,0,0);
        processColorCli(color, flags[COLOR_INDEX].value);
        std::string inputFilename;
        std::string outputFilename = "out.png";
        proccessFileOutname(flags,inputFilename, outputFilename,
lastArgument);
        bool fill = flags[FILL_INDEX].inputed;
        if (fill)
            processColorCli(fill_color, flags[FILL_COLOR_INDEX].value);
        ImageFileReader image;
        image.resolve(inputFilename.c_str());
        image.drawHexagon(center.first, center.second, radius, color,
thickness, fill, fill_color);
        image.write(outputFilename.c_str());
    }

    void copyArea(std::map<int, Flag> flags, std::string lastArgument) {
        std::pair<int, int> left_up =
processCoordinatesCli(flags[LEFT_UP_INDEX].value);
        std::pair<int, int> right_down =
processCoordinatesCli(flags[RIGHT_DOWN_INDEX].value);
        std::pair<int, int> dest_left_up =
processCoordinatesCli(flags[DEST_LEFT_UP_INDEX].value);
        std::string inputFilename;
        std::string outputFilename = "out.png";
        proccessFileOutname(flags, inputFilename, outputFilename,
lastArgument);
        ImageFileReader image;
        image.resolve(inputFilename.c_str());
        image.copyArea(left_up.first,left_up.second,right_down.first,
right_down.second, dest_left_up.first, dest_left_up.second);
        image.write(outputFilename.c_str());
    }

    void helpUser(){
```

```
        std::cout << "Программа должна реализовывать весь следующий функционал по обработке png-файла." << std::endl
                  << "Программа должна иметь следующую функции по обработке изображений: " << std::endl
                  << "(1) Рисование прямоугольника. Флаг для выполнения данной операции: `--rect`. Он определяется:" << std::endl
                  << "Координатами левого верхнего угла. Флаг `--left_up`, значение задаётся в формате `left.up`, где left – координата по x,"
                  << " up – координата по y. Координатами правого нижнего угла.  Флаг `--right_down`, значение задаётся в формате `right.down`,"
                  << " где right – координата по x, down – координата по y." << std::endl
                  << "Толщиной линий: флаг `--thickness`. На вход принимает число больше 0." << std::endl
                  << "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)." << std::endl
                  << "Прямоугольник 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`)." << std::endl
                  << "(2) Рисование правильного шестиугольника. Флаг для выполнения данной операции: `--hexagon`. Шестиугольник определяется:"
                  << std::endl << "координатами его центра и радиусом в который он вписан. Флаги `--center` и `--radius`. Значение флаг `--center`"
                  << " задаётся в формате `x.y`, где x – координата по оси x, y – координата по оси y." << std::endl
                  << "Флаг `--radius` На вход принимает число больше 0." << std::endl
                  << "Толщиной линий. Флаг `--thickness`. На вход принимает число больше 0 цветом линий." << std::endl
                  << "Флаг `--color` (цвет задаётся строкой `rrr.ggg.bbb`, где rrr/ggg/bbb – числа,задающие цветовую компоненту."
                  << " пример `--color 255.0.0` задаёт красный цвет)." << std::endl
                  << "Шестиугольник может быть залит или нет. Флаг `--fill`. Работает как бинарное значение: флага нет – false , флаг есть – true."
                  << " цветом которым залит шестиугольник, если пользователем выбран залитый. Флаг `--fill_color` (работает аналогично флагу `--color`)."
                  << std::endl << "(3) Копирование заданной области. Флаг для выполнения данной операции: `--copy`. Функционал определяется:"
```

```
                << std::endl << "Координатами левого верхнего угла области-
источника. Флаг `--left_up`, значение задаётся в формате `left.up`, где
left – координата"
                << " по x, up – координата по y " << std::endl <<
"Координатами правого нижнего угла области-источника. Флаг `--right_down`,
значение"
                << " задаётся в формате `right.down`, где right –
координата по x, down – координата по y" << std::endl
                << "Координатами левого верхнего угла области-назначения.
Флаг `--dest_left_up`, значение задаётся в формате `left.up`,"
                << " где left – координата по x, up – координата по y." <<
std::endl;
     }

     void infoImage(std::map<int, Flag> flags, std::string lastArgument){
         ImageFileReader image;
         std::string inputFilename;
         std::string outputFilename;
         proccessFileOutname(flags,inputFilename,        outputFilename,
lastArgument);
         image.resolve(inputFilename.c_str());
         std:: cout << "Высота изображения " << image.getHeight() <<
std::endl
                 <<"Ширина изображения " << image.getWidth() << std::endl
                 <<"Глубина изображения " << image.getBitDepth() << " в
байтах" << std::endl
                 <<"Тип цвета изображения " << image.getColortype() <<
std::endl;
     }

     bool flagsRectangle(std::map<int, Flag> flags) {
         std::set<int>                  requiredFlags                  =
{RECT_INDEX,LEFT_UP_INDEX,RIGHT_DOWN_INDEX,             THICKNESS_INDEX,
COLOR_INDEX};
         std::set<int>  functionalFlags  =  {INPUT_INDEX,  OUTPUT_INDEX,
FILL_INDEX, FILL_COLOR_INDEX};
         std::set<int>    unusableFlags    =    getUnusableFlags(flags,
requiredFlags, functionalFlags);
         return  checkValidFlags(flags,  requiredFlags,  unusableFlags)
&& !(flags[FILL_INDEX].inputed && !flags[FILL_COLOR_INDEX].inputed);
     }

     bool flagsHexagon(std::map<int, Flag> flags) {
```

```
        std::set<int> necessaryFlags = {HEXAGON_INDEX, CENTER_INDEX,
RADIUS_INDEX, THICKNESS_INDEX, COLOR_INDEX};
        std::set<int> functionalFlags = {INPUT_INDEX, OUTPUT_INDEX,
FILL_INDEX, FILL_COLOR_INDEX};
        std::set<int>    unusableFlags    =    getUnusableFlags(flags,
necessaryFlags, functionalFlags);
        return checkValidFlags(flags, necessaryFlags, unusableFlags)
&& !(flags[FILL_INDEX].inputed && !(flags[FILL_COLOR_INDEX].inputed));
    }

    bool flagsCopy(std::map<int, Flag> flags) {
        std::set<int> necessaryFlags = {COPY_INDEX, LEFT_UP_INDEX,
RIGHT_DOWN_INDEX, DEST_LEFT_UP_INDEX};
        std::set<int> functionalFlags = {INPUT_INDEX, OUTPUT_INDEX};
        std::set<int>    unusableFlags    =    getUnusableFlags(flags,
necessaryFlags, functionalFlags);
        return checkValidFlags(flags, necessaryFlags, unusableFlags);
    }

    bool flagsHelp(std::map<int, Flag> flags) {
        std::set<int> necessaryFlags = {};
        std::set<int> functionalFlags = {HELP_INDEX};
        std::set<int>    unusableFlags    =    getUnusableFlags(flags,
necessaryFlags, functionalFlags);
        return checkValidFlags(flags, necessaryFlags, unusableFlags);
    }

    bool flagsInfo(std::map<int, Flag> flags) {
        std::set<int> necessaryFlags = {INFO_INDEX};
        std::set<int> functionalFlags = {INPUT_INDEX};
        std::set<int>    unusableFlags    =    getUnusableFlags(flags,
necessaryFlags, functionalFlags);
        return checkValidFlags(flags, necessaryFlags, unusableFlags);
    }
    void selectionFunction(std::map<int, Flag> flags, std::string
lastArgument){
        if (flagsRectangle(flags)){
            drawRectangle(flags, lastArgument);
        }
        else if (flagsHexagon(flags)){
            drawHexagon(flags, lastArgument);
        }
        else if (flagsCopy(flags)) {
            copyArea(flags, lastArgument);
```

```
        }
        else if (flagsHelp(flags)){
            helpUser();
        }
        else if (flagsInfo(flags)){
            infoImage(flags, lastArgument);
        }
        else {
            std::cout << "Error CLI: Unknown function" << std::endl;
            exit(40);
        }
    }

    void processCLI(int argc, char** argv) {
        std::map<int, Flag> flags = {
            {INPUT_INDEX, {0, ""}},
            {OUTPUT_INDEX, {0, ""}},
            {HELP_INDEX, {0, ""}},
            {INFO_INDEX, {0, ""}},
            {RECT_INDEX, {0, ""}},
            {LEFT_UP_INDEX, {0, ""}},
            {RIGHT_DOWN_INDEX, {0, ""}},
            {THICKNESS_INDEX, {0, ""}},
            {COLOR_INDEX, {0, ""}},
            {FILL_INDEX, {0, ""}},
            {FILL_COLOR_INDEX, {0, ""}},
            {HEXAGON_INDEX, {0, ""}},
            {CENTER_INDEX, {0, ""}},
            {RADIUS_INDEX, {0, ""}},
            {DEST_LEFT_UP_INDEX, {0, ""}},
            {COPY_INDEX, {0, ""}}
        };
        const char* short_options = "hi:o:";
        const option long_options[] = {
            {"input", required_argument, nullptr, INPUT_INDEX},
            {"output", required_argument, nullptr, OUTPUT_INDEX},
            {"help", no_argument, nullptr, HELP_INDEX},
            {"info", no_argument, nullptr, INFO_INDEX},
            {"rect", no_argument, nullptr, RECT_INDEX},
            {"left_up", required_argument, nullptr, LEFT_UP_INDEX},
            {"right_down",          required_argument,          nullptr,
RIGHT_DOWN_INDEX},
            {"thickness", required_argument, nullptr, THICKNESS_INDEX},
            {"color", required_argument, nullptr, COLOR_INDEX},
```

```
            {"fill", no_argument, nullptr, FILL_INDEX},
            {"fill_color",          required_argument,          nullptr,
FILL_COLOR_INDEX},
            {"hexagon", no_argument, nullptr, HEXAGON_INDEX},
            {"center", required_argument, nullptr, CENTER_INDEX},
            {"radius", required_argument, nullptr, RADIUS_INDEX},
            {"dest_left_up",         required_argument,         nullptr,
DEST_LEFT_UP_INDEX},
            {"copy", no_argument, nullptr, COPY_INDEX},
            {nullptr, 0, nullptr, 0}
        };
        int option;
        while   ((option   =   getopt_long(argc,   argv,   short_options,
long_options, NULL)) != -1) {
            if (flags.find(option) != flags.end()){
                flags[option].inputed = true;
                if (optarg) {
                    flags[option].value = optarg;
                }
            }
            else {
                std::cout << "Error CLI: wrong flag" << std::endl;
                exit(40);
            }
        }
        std::string lastArgument = argv[argc - 1];
        selectionFunction(flags, lastArgument);
    }

    int main(int argc, char** argv) {
        description();
        processCLI(argc, argv);
        return 0;
    }
```

Название файла: Makefile

```
CC = g++
CFLAGS = -c -Wall
all: cw
cw: main.o
     $(CC) main.o -o cw -lpng
main.o: main.cpp
     $(CC) $(CFLAGS) main.cpp
clean:
     @ rm -rf *.o
```

ПРИЛОЖЕНИЕ В
ДЕМОНСТРАЦИЯ РАБОТЫ ПРОГРАММЫ

--help: ./cw

```
Course work for option 4.21, created by Vladislav Trofimov.
Программа должна реализовывать весь следующий функционал по обработке png-файла.
Программа должна иметь следующую функции по обработке изображений:
(1) Рисование прямоугольника. Флаг для выполнения данной операции: `--rect`. Он определяется:
Координатами левого верхнего угла. Флаг `--left_up`, значение задаётся в формате `left.up`, где left – координата по x, up – координата по y. Координатами правог
о нижнего угла.  Флаг `--right_down`, значение задаётся в формате `right.down`, где right – координата по x, down – координата по y.
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).
Прямоугольник может быть залит или нет. Флаг `--fill`. Работает как бинарное значение: флага нет – false , флаг есть – true. цветом которым он залит, если пользо
вателем выбран залитый. Флаг `--fill_color` (работает аналогично флагу `--color`).
(2) Рисование правильного шестиугольника. Флаг для выполнения данной операции: `--hexagon`. Шестиугольник определяется:
координатами его центра и радиусом в который он вписан. Флаги `--center` и `--radius`. Значение флаг `--center` задаётся в формате `x.y`, где x – координата по о
си x, y – координата по оси y.
Флаг `--radius` На вход принимает число больше 0.
Толщиной линий. Флаг `--thickness`. На вход принимает число больше 0 цветом линий.
Флаг `--color` (цвет задаётся строкой `rrr.ggg.bbb`, где rrr/ggg/bbb – числа,задающие цветовую компоненту. пример `--color 255.0.0` задаёт красный цвет).
Шестиугольник может быть залит или нет. Флаг `--fill`. Работает как бинарное значение: флага нет – false , флаг есть – true. цветом которым залит шестиугольник,
если пользователем выбран залитый. Флаг `--fill_color` (работает аналогично флагу `--color`).
(3) Копирование заданной области. Флаг для выполнения данной операции: `--copy`. Функционал определяется:
Координатами левого верхнего угла области-источника. Флаг `--left_up`, значение задаётся в формате `left.up`, где left – координата по x, up – координата по y
Координатами правого нижнего угла области-источника. Флаг `--right_down`, значение задаётся в формате `right.down`, где right – координата по x, down – координат
а по y
Координатами левого верхнего угла области-назначения. Флаг `--dest_left_up`, значение задаётся в формате `left.up`, где left – координата по x, up – координата п
о y.
```

Рисунок 1– Вывод справки о возможностях программы.

Рисунок 2 – Входное изображение для всех тестов.

--rect: ./cw --output out.png --input test.png --rect --right_down 487.170 --left_up 51.23 --thickness 10 --fill --fill_color 186.236.44 --color 255.0.0

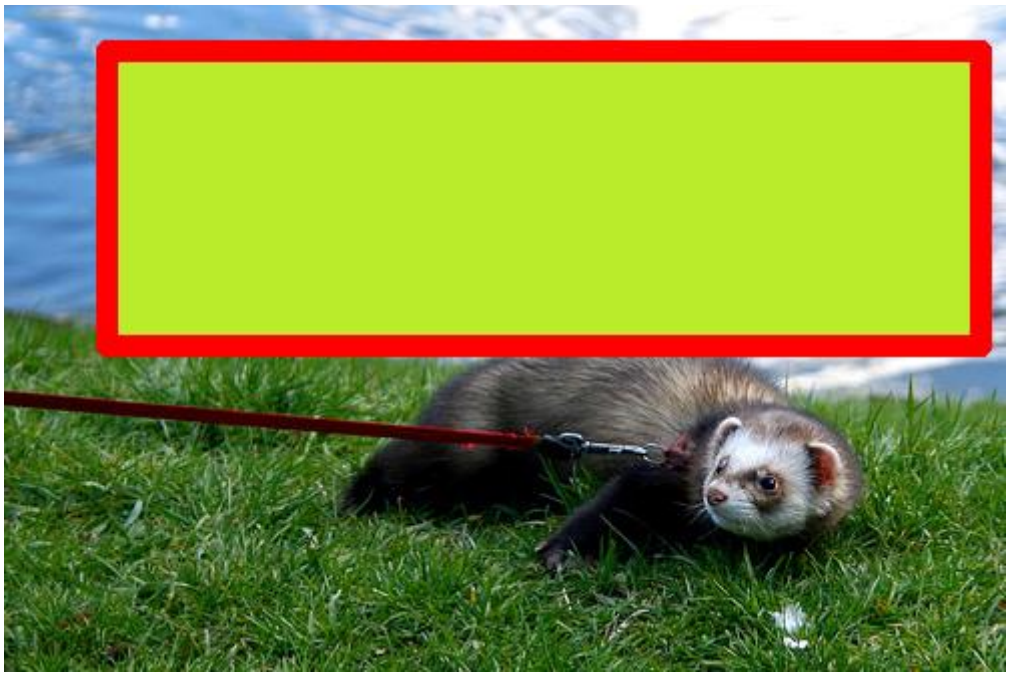

Рисунок 3 – Результат работы рисования прямоугольника.

--hexagon: ./cw --output out.png --input test.png --hexagon --thickness 10 --fill --fill_color 255.255.255 --color 0.0.255 --center 250.250 --radius 50

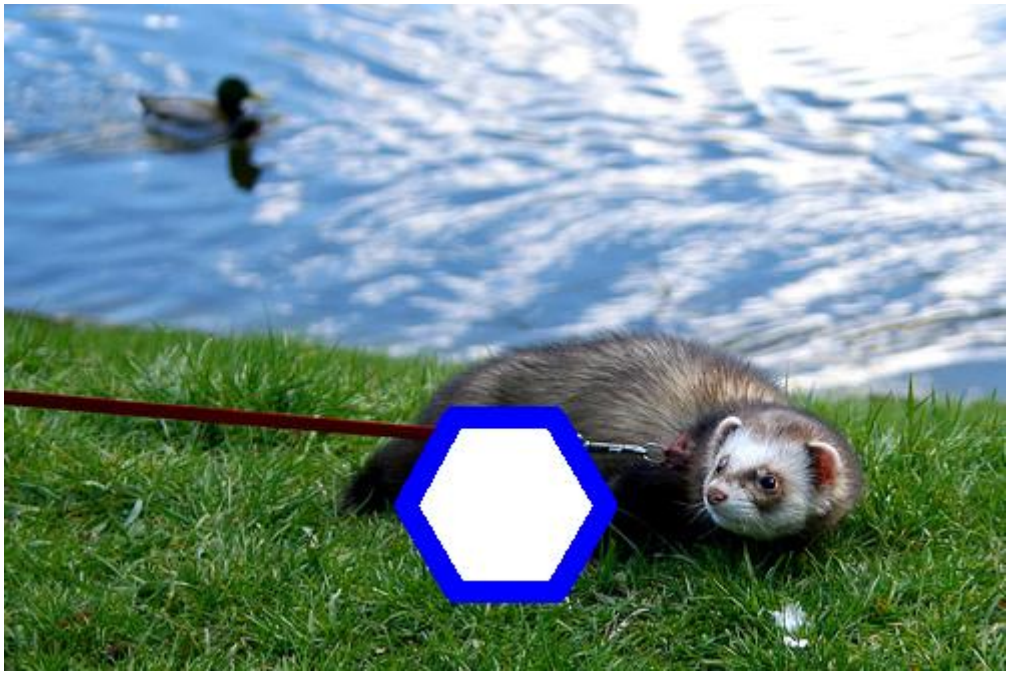

Рисунок 4 – Результат работы рисования шестиугольника.

--copy: ./cw --output out.png --input test.png --copy --dest_left_up 228.291 --right_down 487.170 --left_up 51.23

Рисунок 5 – Результат работы копирования заданной области.

--info: ./cw –input test.png –info

```
tirbah@WIN-FEKLBDRGVA7:/mnt/d/Desktop/Projects/C++$ ./cw --input test.png --info
Course work for option 4.21, created by Vladislav Trofimov.
Высота изображения 333
Ширина изображения 500
Глубина изображения 8 в байтах
Тип цвета изображения 6
```

Рисунок 6 – Результат работы вывода информации о изображении.

Неверная команда: ./cw –ornament

```
tirbah@WIN-FEKLBDRGVA7:/mnt/d/Desktop/Projects/C++$ ./cw --ornament
Course work for option 4.21, created by Vladislav Trofimov.
./cw: unrecognized option '--ornament'
Error CLI: wrong flag
```

Рисунок 7 – Результат работы при вводе неверной команды.