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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Обработка изображений

Студент гр. 3344 _____ **Жаворонок Д. Н.**

Преподаватель _____ **Глазунов С.А.**

Санкт-Петербург

2024

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студент Жаворонок Д. Н.

Группа 3344

Тема работы: Обработка изображений.

Исходные данные:

- Программа **обязательно должна иметь CLI** (опционально дополнительное использование GUI).
- Программа должна реализовывать весь следующий функционал по обработке png-файла
- 24 бита на цвет
- 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)
- Каждую подзадачу следует вынести в отдельную функцию, функции сгруппировать в несколько файлов
- Сборка должна осуществляться при помощи make и Makefile или другой системы сборки

Содержание пояснительной записки:

- Содержание
- Введение
- Описание задания
- Описание реализованных функций, структур
- Описание файловой структуры программы
- Описание модульной структуры, сборки программы
- Примеры работы программы
- Примеры ошибок
- Заключение
- Список использованных источников
- Приложение А. Исходный код программы

Предполагаемый объем пояснительной записки:

Не менее 30 страниц.

Дата выдачи задания: 18.03.2024

Дата сдачи реферата: 22.05.2024

Дата защиты реферата: 22.05.2024

Студент _____ Жаворонок Д. Н.

Преподаватель _____ Глазунов С.А.

АННОТАЦИЯ

Данная курсовая работа посвящена разработке программы для обработки изображений в формате PNG с использованием командного интерфейса (CLI). Программа реализует функционал для чтения и обработки PNG-файлов, включая проверку соответствия формату PNG, а также обработку мусорных данных выравнивания.

Основные функции программы включают:

Рисование треугольника: Программа позволяет рисовать треугольник, задавая координаты его вершин, толщину и цвет линий. Треугольник может быть не залит или залит, в таком случае можно выбрать цвет заливки.

Нахождение и перекраска самого большого прямоугольника заданного цвета: Программа находит самый большой прямоугольник заданного цвета и перекрашивает его в другой цвет который так же указывается пользователем.

Создание коллажа: Программа создает коллаж из одного изображения, повторяя его N*M раз, где N и M задаются пользователем.

Рисование отрезка: Программа позволяет рисовать отрезок, задавая координаты начала и конца, цвет и толщину линии.

Программа использует библиотеку stb_image для работы с PNG-файлами и обеспечивает корректную обработку заголовков и данных изображения.

Сборка программы осуществляется с использованием системы сборки make и Makefile. Все функции сгруппированы в отдельные модули для обеспечения удобства и структурированности кода.

Результатом работы программы является обработанное изображение, сохраненное в файл с заданным именем (по умолчанию output.png), а также справка о реализованных функциях и информация о возможных ошибках.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью данной работы является создание программы на языке программирования C++, которая будет обрабатывать PNG-изображение с помощью CLI интерфейса.

Для достижения поставленной цели требуется решить следующие задачи:

1. Изучить формат PNG
2. Изучить методы реализации CLI интерфейса
3. Реализовать функции обработки изображения
4. Реализовать эффективную сборку программы
5. Предусмотреть возможные ошибки и их причины

Возможные методы решения поставленных задач:

1. Использование библиотеки getopt для работы с командной строкой
2. Использование библиотеки stb_image для работы с изображениями
3. Сборка проекта с помощью Makefile
4. Вынесение каждой подзадачи в отдельную функцию

1. ОПИСАНИЕ ЗАДАНИЯ

Программа должна иметь следующую функции по обработке изображений:

Рисование треугольника. Флаг для выполнения данной операции: `--triangle`. Треугольник определяется

Координатами его вершин. Флаг `--points`, значение задаётся в формате `x1.y1.x2.y2.x3.y3` (точки будут (x1; y1), (x2; y2) и (x3; y3)), где x1/x2/x3 – координаты по x, y1/y2/y3 – координаты 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)

Треугольник 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`)

Находит самый большой прямоугольник заданного цвета и перекрашивает его в другой цвет. Флаг для выполнения данной операции: `--biggest_rect`. Функционал определяется:

Цветом, прямоугольник которого надо найти. Флаг `--old_color` (цвет задаётся строкой `rrr.ggg.bbb`, где rrr/ggg/bbb – числа, задающие цветовую компоненту. пример `--old_color 255.0.0` задаёт красный цвет)

Цветом, в который надо его перекрасить. Флаг `--new_color` (работает аналогично флагу `--old_color`)

Создать коллаж размера N*M из одного изображения. Флаг для выполнения данной операции: `--collage`. Коллаж представляет собой это же самое изображение повторяющееся N*M раз.

Количество изображений по “оси” Y. Флаг `--number_y`. На вход принимает число больше 0

Количество изображений по “оси” X. Флаг `--number_x`. На вход принимает число больше 0

Рисование отрезка. Флаг для выполнения данной операции: `--line`. Отрезок определяется:

координатами начала. Флаг `--start`, значение задаётся в формате `x.y`, где x – координата по x, y – координата по y

координатами конца. Флаг `--end` (аналогично флагу `--start`)

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

толщиной. Флаг `--thickness`. На вход принимает число больше 0

Каждую подзадачу следует вынести в отдельную функцию, функции сгруппировать в несколько файлов (например, функции обработки текста в один, функции ввода/вывода в другой). Сборка должна осуществляться при помощи make и Makefile или другой системы сборки

2. ОПИСАНИЕ ПРОГРАММЫ

2.1. Реализованные функции, структуры

Во время разработки программы были реализованы следующие **структуры**:

1. position - используется для хранения 2d координат.
2. Circle - используется для хранения информации о круге, координаты его центра и радиус.
3. triangle - используется для хранения информации о треугольнике, 3 координаты его вершин.
4. option_temp - используется для хранения шаблонной информации для параметров принимаемых программой, указывает длинное имя, требуется ли аргумент и какая информация должна выводиться в справке о ней.
5. arg - используется для хранения информации о количестве переданных аргументов для каждого разрешенного флага и их значениях
6. RGB - используется для хранения информации о цвете пикселя
7. Rectangle - используется для хранения информации о прямоугольнике

Во время разработки программы были реализованы следующие **функции**:

template <class T> bool contains(const T &map, const std::string &key) — шаблонная функция проверяющая наличие ключа типа *std::string* в хэш-мапе.

std::unordered_map<char, option_temp> gen_input_temp() — функция, генерирующая хэш-мапу определяющую все возможные опции программы.

std::vector<option> gen_long_options(const std::unordered_map<char, option_temp> &input_temp) — функция, генерирущая массив опций типа *option*, используемый библиотечной функцией *getopt_long()*

*std::vector<int> parse_input(std::string input, bool (*check)(int), int required_num_of_params)* — функция, обрабатывающая переданные на вход

данные каждой отдельной опции, выбрасывает ошибку в случае неправильности введенных данных

*std::unordered_map<std::string, arg> input(int argc, char **argv)* — функция, обрабатывающая переданные на вход данные программы, генерирует хэш-мапу для удобного доступа к парам опция — переданное значение из любой точки программы.

bool check_no_intersecting_flags(const std::vector<std::string> &corresponding_options, const std::unordered_map<std::string, arg> &input_data) — функция, проверяющая, что при вызове какой-то определенной функции ей были переданы все необходимые аргументы и не было передано ничего лишнего.

std::unordered_map<std::string, std::vector<std::string>> gen_corresponding_options() — функция, генерирующая хэш-мапу для быстрого и удобного дотупа и определения взаимосвязанных разрешенных опций.

std::string get_function_to_exec(std::unordered_map<std::string, arg> input_data) — функция, на основе входных данных определяющая, какую функцию требуется вызвать.

void print_help() — функция, генерирующая и выводящая справку о программе.

bool contains(const std::vector<std::string> &container, const std::string &key) — функция, проверяющая наличие строки в векторе строк

bool check_rgb_val(int val) — функция, проверяющая диапазон значений RGB.

bool check_N_val(int val) — функция, проверяющая что значение является натуральным числом.

bool no_check(int) — заглушка на случай, если не требуется никаких проверок.

void throw_exception(std::string error, int exitCode) — функция, выводящая сообщение об ошибке и завершающая выполнение программы с определенным кодом выхода.

*RGB *make_collage(RGB *pixels, int w, int h, int N, int M, int &cw, int &ch)* — функция, генерирующая коллаж из заданного изображения повторяя его N на M раз.

*bool is_surrounded(RGB *image, int w, int h, const Rectangle &rect, const RGB &oldColor)* — функция, проверяющая, что вокруг Rectangle существует «рамка» из пикселей цвета, отличающихся от исходного.

*void find_and_recolor_biggest_rect(RGB *image, int w, int h, const RGB &oldColor, const RGB &newColor)* — функция, находящая и перекрашивающая наибольший прямоугольник заданного цвета в другой цвет.

*void draw_pixel(RGB *image, int width, int height, position pos, RGB color)* — функция, закрашивающая заданный пиксель изображения в определенный цвет.

*void draw_circle_line(RGB *image, int width, int height, Circle c, RGB color)* — функция рисования окружности без заполения.

*void draw_circle_filled(RGB *image, int width, int height, Circle c, RGB color)* — функция рисования круга с заполнением определнного цвета.

Rectangle get_line_bounding_box(position a, position b, int thickness) — функция, генерирующая ограничительную рамку для повернутой линии определенной толщины.

bool rectangles_intersect(const Rectangle &r1, const Rectangle &r2) — функция, проверяющая коллизии двух прямоугольников в пространстве.

*void draw_line(RGB *image, int width, int height, position a, position b, int thickness, RGB color)* — функция, рисующая линию определенной ширины по заданным координатам.

*void draw_flat_horizontal_line(RGB *image, int width, int height, position a, position b, RGB color)* — функция, рисующая тонкую горизонтальзую линию по заданным координатам.

*void fill_bottom_flat_triangle(RGB *image, int width, int height, triangle tri, RGB fill_color)* — функция, рисующая треугольник с плоским основанием снизу.

*void fill_top_flat_triangle(RGB *image, int width, int height, triangle tri, RGB fill_color)* — функция, рисующая треугольник с плоской гранью сверху.

bool position_y_sorter(position const &lhs, position const &rhs) — предикат используемый для сортировки координат точек треугольника по возрастанию.

*void draw_triangle(RGB *image, int width, int height, triangle tri, int thickness, RGB color, bool fill, RGB fill_color)* — функция, рисующая заполненный или не заполненный треугольник.

2.2. Файловая структура программы

Во время разработки программа была разбита на следующие файлы:

- Makefile — файл, необходимый для компиляции и сборки проекта.
- biggest_rect.h — заголовочный файл с объявлениями функций, необходимыми для реализации поиска и перекрашивания наибольшего прямоугольника заданного цвета.
- biggest_rect.cpp — файл исходного кода с определениями функций, необходимыми для реализации поиска и перекрашивания наибольшего прямоугольника заданного цвета.
- draw_line.h — заголовочный файл с объявлениями функций, необходимыми для рисования линий.
- draw_line.cpp — файл исходного кода с определениями функций, необходимыми для рисования линий.
- draw_triangle.h — заголовочный файл с объявлениями функций, необходимыми для рисования треугольников.
- draw_triangle.cpp — файл исходного кода с определениями функций, необходимыми для рисования треугольников.
- make_collage.h — заголовочный файл с объявлениями функций, необходимыми для создания коллажа из данного изображения.
- make_collage.cpp — файл исходного кода с определениями функций, необходимыми для создания коллажа из данного изображения.

- input.h — заголовочный файл с объявлениями функций, необходимыми для обработки пользовательского ввода.
- input.cpp — файл исходного кода с определениями функций, необходимыми для обработки пользовательского ввода.
- rgb.h — заголовочный файл с объявлениями функций, необходимыми для обработки информации о цвете.
- rgb.cpp — файл исходного кода с определениями функций, необходимыми для обработки информации о цвете.
- throw_exception.h — заголовочный файл с объявлениями функций, необходимыми для выбрасывания ошибок.
- throw_exception.cpp — файл исходного кода с определениями функций, необходимыми для выбрасывания ошибок.
- utils.h — заголовочный файл с объявлениями функций, используемых некоторыми другими функциями в программе.
- utils.cpp — файл исходного кода с определениями функций, используемых некоторыми другими функциями в программе.
- stb_image.h — файл библиотеки stb_image, используется для доступа к функции для считывания png файла.
- stb_image_write.h — файл библиотеки stb_image_write, используется для доступа к функции для записи png файла.
- main.cpp — главный файл с исходным кодом.
- Makefile — файл с описанием процесса сборки проекта.

2.3. Модульная структура, сборка

Для сборки проекта используется Makefile:

Компилятор — g++; флаги компиляции — -std=c++11 -Wno-deprecated

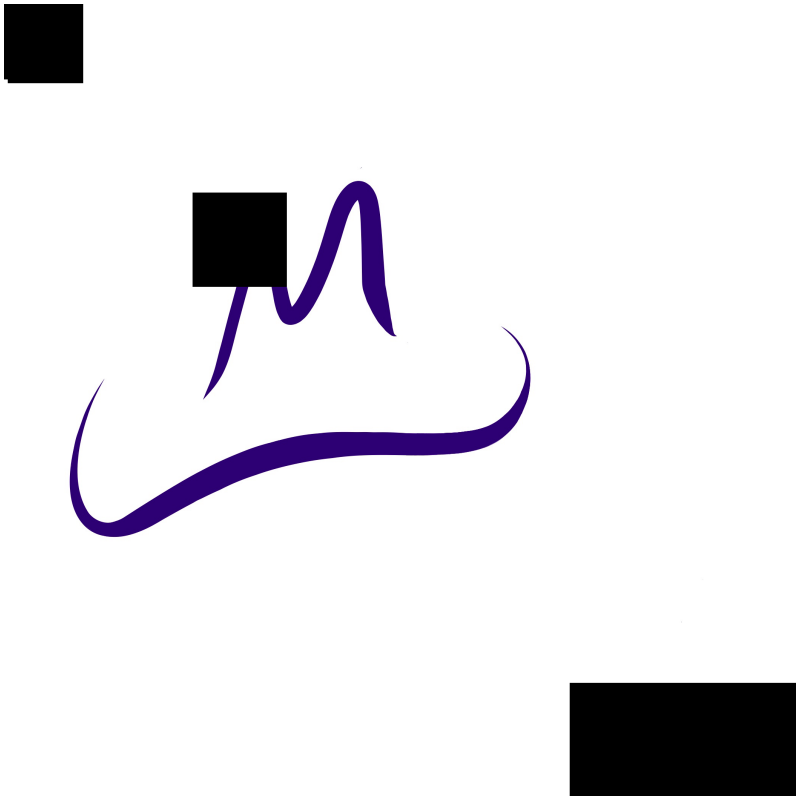
- cw – главная цель сборки, генерирует исполняемый файл, требует все нижеперечисленные объектные файлы для сборки и линковки.
- rgb.o – объектный файл, требующий src/rgb.cpp и include/rgb.h для компиляции.
- throw_exception.o – объектный файл, требующий src/throw_exception.cpp и include/throw_exception.h для компиляции.
- utils.o – объектный файл, требующий src/utils.cpp, include/utils.h и include/throw_exception.h для компиляции.
- input.o – объектный файл, требующий src/input.cpp, include/input.h, include/rgb.h, include/utils.h и include/throw_exception.h для компиляции.
- draw_line.o – объектный файл, требующий src/draw_line.cpp, include/draw_line.h и include/rgb.h для компиляции.
- biggest_rect.o – объектный файл, требующий src/biggest_rect.cpp, include/biggest_rect.h, include/rgb.h и include/draw_line.h для компиляции.
- make_collage.o – объектный файл, требующий src/make_collage.cpp, include/make_collage.h и include/rgb.h для компиляции.
- draw_triangle.o – объектный файл, требующий src/draw_triangle.cpp, include/draw_triangle.h и include/draw_line.h для компиляции.
- main.o – объектный файл, требующий main.cpp, include/rgb.h, include/throw_exception.h, include/utils.h, include/input.h, include/draw_line.h, include/biggest_rect.h, include/make_collage.h и include/draw_triangle.h для компиляции.
- clean – очистка всех объектных файлов и исполняемого файла cw.

3. ПРИМЕРЫ РАБОТЫ ПРОГРАММЫ

Вывод справки о программе:

```
Course work for option 5.14, created by Zhavoronok Danila.
--help (Displays this message.)

--line (Draw line flag.)
        --start (Start point coordinates. Format: x.y)
        --end (End point coordinates. Format: x.y)
        --thickness (Border line thickness. Format: n > 0)
        --color (Color of the line. Format: <rrr.ggg.bbb>, where each component is in range [0, 255])
        --input (Path to provided image. Format: filepath)
        --output (Path to output image. Format: filepath)

--collage (Make collage flag.)
        --number_x (Number of times to repeat the image on the X axis. Format: n > 0)
        --number_y (Number of times to repeat the image on the Y axis. Format: n > 0)
        --input (Path to provided image. Format: filepath)
        --output (Path to output image. Format: filepath)

--biggest_rect (Find the biggest rectangle of a specified color and recolor it flag.)
        --old_color (Color to search the biggest rect. Format: identic to --color)
        --new_color (Color to recolor the biggest rect to. Format: identic to --color)
        --input (Path to provided image. Format: filepath)
        --output (Path to output image. Format: filepath)

--triangle (Draw triangle flag.)
        --points (Triangle vertices coordinates. Format: <x1.y1.x2.y2.x3.y3>)
        --thickness (Border line thickness. Format: n > 0)
        --color (Color of the line. Format: <rrr.ggg.bbb>, where each component is in range [0, 255])
        --fill (Fill triangle background flag.)
        --fill_color (Color of the backgroud. Format: identic to --color)
        --input (Path to provided image. Format: filepath)
        --output (Path to output image. Format: filepath)
```

Исходная картинка:



Рисование линии:

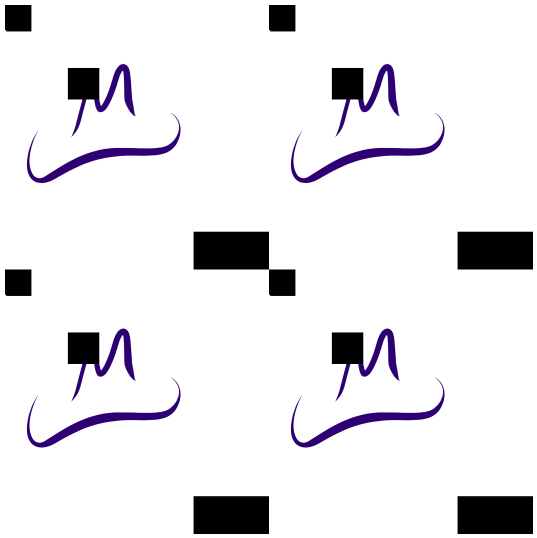
Входные данные: —end 630.-467 --color 215.7.184 --input ./moevm.png --thickness 1400 --start 2560.-542 —line

Вывод программы:

Создание коллажа:

Входные данные: --input ./moevm.png --number_x 2 --number_y 2 —collage

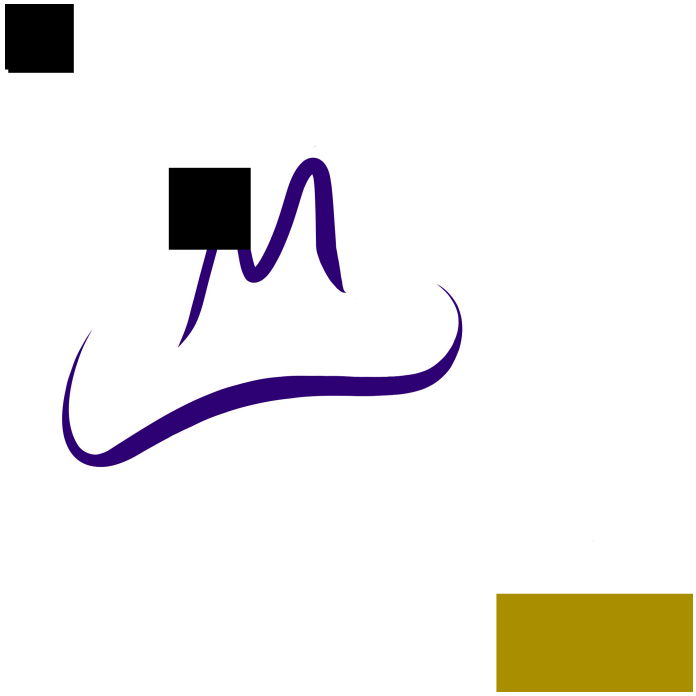
Вывод программы:



Закрашивание наибольшего прямоугольника выбранного цвета другим цветом:

Входные данные: --input ./moevm.png --old_color 0.0.0 --new_color 169.142.0 —biggest_rect

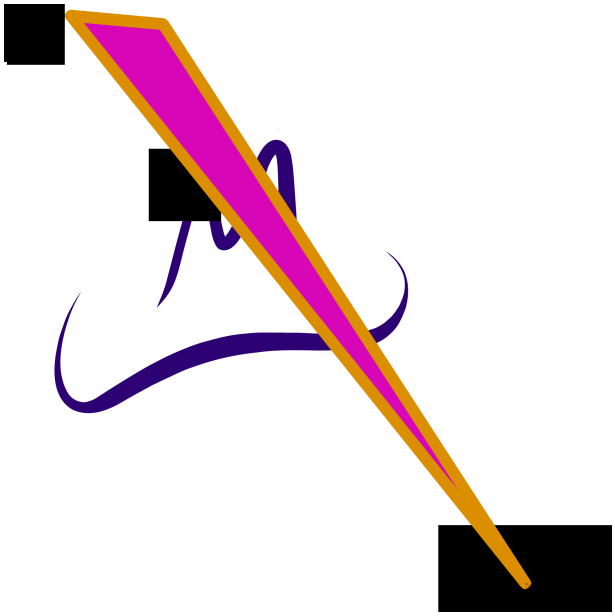
Вывод программы:



Рисование треугольника;

Входные данные: --input ./moevm.png --points 230.40.549.69.1800.2000 --thickness 40 --color 219.142.0 --fill --fill_color 215.7.184 —triangle

Вывод программы:



ЗАКЛЮЧЕНИЕ

Была успешно реализована программа, обрабатывающая PNG изображения согласно инструкциям в виде флагов и аргументов передаваемых пользователем через CLI. Программа выполняет поставленные задачи по считыванию, обработке и записи PNG изображений.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. The GNU C Library Reference Manual. GETOPT. URL: https://www.gnu.org/software/libc/manual/html_node/Getopt.html (20.04.2024)

2. Бьёрн Страуструп: A Tour of C++ (2nd Edition) (2018) США: Addison-Wesley, 2018 г.

3. Базовые сведения к выполнению курсовой работы по дисциплине «программирование». второй семестр: учеб.-метод. Пособие сост. А. А. Лисс, С. А. Глазунов, М. М. Заславский, К. В. Чайка и др. СПб.: Изд-во СПбГЭТУ "ЛЭТИ", 2024. 36 с.

4. Язык программирования C++ упражнения и лекции 5-ое издание. Стивен Прата. Из. «Вильямс» 2007г

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Makefile

```
CC =g++
CFLAGS = -std=c++11 -Wno-deprecated

all: cw

cw: main.o rgb.o throw_exception.o utils.o input.o draw_line.o
biggest_rect.o make_collage.o draw_triangle.o
    ${CC} $^ -o $@ ${CFLAGS}

rgb.o: src/rgb.cpp include/rgb.h
    ${CC} -c $< ${CFLAGS}

throw_exception.o: src/throw_exception.cpp include/throw_exception.h
    ${CC} -c $< ${CFLAGS}

utils.o: src/utils.cpp include/utils.h include/throw_exception.h
    ${CC} -c $< ${CFLAGS}

input.o: src/input.cpp include/input.h include/rgb.h include/utils.h
include/throw_exception.h
    ${CC} -c $< ${CFLAGS}

draw_line.o: src/draw_line.cpp include/draw_line.h include/rgb.h
    ${CC} -c $< ${CFLAGS}

biggest_rect.o: src/biggest_rect.cpp include/biggest_rect.h include/
rgb.h include/draw_line.h
    ${CC} -c $< ${CFLAGS}

make_collage.o: src/make_collage.cpp include/make_collage.h include/
rgb.h
    ${CC} -c $< ${CFLAGS}

draw_triangle.o: src/draw_triangle.cpp include/draw_triangle.h
include/draw_line.h
    ${CC} -c $< ${CFLAGS}

main.o: main.cpp include/rgb.h include/throw_exception.h include/
utils.h include/input.h include/draw_line.h include/biggest_rect.h
include/make_collage.h include/draw_triangle.h
    ${CC} -c $< ${CFLAGS}

clean:
    rm *.o cw
```

rgb.h

```
#pragma once
#include <iostream>
#include <vector>

struct RGB
{
```

```
    uint8_t r = 0;
    uint8_t g = 0;
    uint8_t b = 0;

    RGB(uint8_t r, uint8_t g, uint8_t b);
    RGB(const std::vector<int> &vec);
    RGB() = default;
    ~RGB() = default;

    bool operator==(const RGB &rgb);
    bool operator!=(const RGB &rgb);
    friend std::ostream &operator<<(std::ostream &os, const RGB &rgb);
};
```

rgb.cpp

```
#include "../include/rgb.h"

RGB::RGB(uint8_t r, uint8_t g, uint8_t b) : r(r), g(g), b(b){};
RGB::RGB(const std::vector<int> &vec)
{
    if (vec.size() != 3)
        return;
    *this = RGB(vec[0], vec[1], vec[2]);
}

bool RGB::operator==(const RGB &color)
{
      return this->r == color.r && this->g == color.g && this->b ==
color.b;
}

bool RGB::operator!=(const RGB &color)
{
    return !(*this == color);
}

std::ostream &
operator<<(std::ostream &os, const RGB &color)
{
      os << unsigned(color.r) << '.' << unsigned(color.g) << '.' <<
unsigned(color.b);
    return os;
}
```

throw_exception.h

```
#pragma once
#include <iostream>
#include <string>

void throw_exception(std::string error, int exitCode);
```

throw_exception.cpp

```
#include "../include/throw_exception.h"

void throw_exception(std::string error, int exitCode)
{
```

```
    std::cerr << error << '\n';
    exit(exitCode);
}
```

utils.h

```
#pragma once
#include <string>
#include <vector>
#include <algorithm>
#include <iostream>
#include <unordered_map>
#include "../include/throw_exception.h"

bool contains(const std::vector<std::string> &container, const
std::string &key);

std::ostream &operator<<(std::ostream &os, const
std::vector<std::string> &val);

bool check_rgb_val(int val);
bool check_N_val(int val);
bool no_check(int);
```

utils.cpp

```
#include "../include/utils.h"

std::ostream &
operator<<(std::ostream &os, const std::vector<std::string> &val)
{
    for (auto &it : val)
        os << '<' << it << '>' << ' ';

    return os;
}

bool contains(const std::vector<std::string> &container, const
std::string &key)
{
        return std::find(container.begin(), container.end(), key) !=
container.end();
}

bool check_rgb_val(int val)
{
    if (val >= 0 && val <= 255)
        return true;

    throw_exception("RGB value is not in range.", 43);
    return false;
}

bool check_N_val(int val)
{
    if (val >= 0)
        return true;

    throw_exception("Value is less than zero.", 44);
    return false;
```

```
}

bool no_check(int)
{
    return true;
}
```

input.h

```
#pragma once
#include "../include/utils.h"
#include "../include/throw_exception.h"
#include <sstream>
#include <getopt.h>
#include <vector>
#include <iostream>
#include <string>
#include <unordered_map>

template <class T>
bool contains(const T &map, const std::string &key);

struct option_temp
{
    std::string long_name;
    int requires_arg;
    std::string help_info;
};

std::unordered_map<char, option_temp> gen_input_temp();
std::vector<option> gen_long_options(const std::unordered_map<char,
option_temp> &input_temp);

struct arg
{
    std::vector<std::string> values;
    int count = 0;
};
std::ostream &operator<<(std::ostream &os, const arg &val);

std::unordered_map<std::string, arg> input(int argc, char **argv);

std::vector<int> parse_input(std::string input, bool (*check)(int),
int required_num_of_params);

bool check_no_intersecting_flags(const std::vector<std::string>
&corresponding_options, const std::unordered_map<std::string, arg>
&input_data);
std::unordered_map<std::string, std::vector<std::string>>
gen_corresponding_options();
std::string get_function_to_exec(std::unordered_map<std::string, arg>
input_data);

void print_help();
```

input.cpp

```
#include "../include/input.h"
#include <iostream>
#include <getopt.h>
```

```
#include <string>

std::ostream &
operator<<(std::ostream &os, const arg &val)
{
    os << val.values << val.count;

    return os;
}

template <class T>
bool contains(const T &map, const std::string &key)
{
    return map.find(key) != map.end();
}

std::unordered_map<char, option_temp> gen_input_temp()
{
    std::unordered_map<char, option_temp> input_temp;

        input_temp['t'] = {"triangle", no_argument, "Draw triangle
flag."};
    input_temp['p'] = {"points", required_argument, "Triangle vertices
coordinates. Format: <x1.y1.x2.y2.x3.y3>"};
      input_temp['k'] = {"thickness", required_argument, "Border line
thickness. Format: n > 0"};
    input_temp['c'] = {"color", required_argument, "Color of the line.
Format: <rrr.ggg.bbb>, where each component is in range [0, 255]"};
     input_temp['f'] = {"fill", no_argument, "Fill triangle background
flag."};
     input_temp['i'] = {"fill_color", required_argument, "Color of the
backgroud. Format: identic to --color"};

     input_temp['r'] = {"biggest_rect", no_argument, "Find the biggest
rectangle of a specified color and recolor it flag."};
        input_temp['o'] = {"old_color", required_argument, "Color to
search the biggest rect. Format: identic to --color"};
        input_temp['n'] = {"new_color", required_argument, "Color to
recolor the biggest rect to. Format: identic to --color"};

    input_temp['g'] = {"collage", no_argument, "Make collage flag."};
     input_temp['y'] = {"number_y", required_argument, "Number of times
to repeat the image on the Y axis. Format: n > 0"};
    input_temp['x'] = {"number_x", required_argument, "Number of times
to repeat the image on the X axis. Format: n > 0"};

    input_temp['l'] = {"line", no_argument, "Draw line flag."};
        input_temp['s'] = {"start", required_argument, "Start point
coordinates. Format: x.y"};
         input_temp['e'] = {"end", required_argument, "End point
coordinates. Format: x.y"};

     input_temp['I'] = {"input", required_argument, "Path to provided
image. Format: filepath"};
      input_temp['O'] = {"output", required_argument, "Path to output
image. Format: filepath"};
    input_temp['H'] = {"help", no_argument, "Displays this message."};

    return input_temp;
}
```

```
std::vector<option>  gen_long_options(const  std::unordered_map<char,
option_temp> &input_temp)
{
    std::vector<option> long_options;

    for (const auto &it : input_temp)
    {
        const auto &key = it.first;
        const auto &val = it.second;

                    long_options.push_back({val.long_name.c_str(),
val.requires_arg, 0, key});
    }
    long_options.push_back({0, 0, 0, 0});

    return long_options;
}

std::vector<int>  parse_input(std::string  input,  bool  (*check)(int),
int required_num_of_params)
{
    std::istringstream ss(input);
    std::string token;
    std::vector<int> array;
    while (std::getline(ss, token, '.'))
    {
        std::istringstream sstemp(token);
        std::string temp_str;
        int temp = 0;
        try
        {
            sstemp >> temp_str;
            temp = stoi(temp_str);
        }
        catch (const std::exception &e)
        {
            throw_exception("Use int values.", 42);
        }
        check(temp);
        array.push_back(temp);
    }

    if (array.size() != required_num_of_params)
                throw_exception("Num  of  params  isn't  equal  to  " +
std::to_string(required_num_of_params) + ".", 48);
    return array;
}

std::unordered_map<std::string, arg> input(int argc, char **argv)
{
    auto input_temp = gen_input_temp();
    auto long_options = gen_long_options(input_temp);

    std::unordered_map<std::string, arg> input_data;

    int c;
    while (1)
    {
        int option_index = 0;
```

```
                            c = getopt_long(argc, argv,
"tp:k:c:fi:r:o:n:g:y:x:l:s:e:I:O:", long_options.data(),
&option_index);

        if (c == -1)
            break;

        auto option_data = input_temp[c];

        if (!contains(input_data, option_data.long_name))
            input_data[option_data.long_name] = arg();

        input_data[option_data.long_name].count++;
        if (option_data.requires_arg)

input_data[option_data.long_name].values.push_back(optarg);
    }

    return input_data;
}

bool check_no_intersecting_flags(const std::vector<std::string>
&corresponding_options, const std::unordered_map<std::string, arg>
&input_data)
{
    std::vector<std::string> entered_params;
    for (const auto &it : input_data)
    {
        const auto &key = it.first;
        if (!contains(corresponding_options, key))
                    throw_exception("Unnecessary arguments detected,
exiting.", 41);

        entered_params.push_back(key);
    }

    for (auto &key : corresponding_options)
    {
         bool is_required = key != "fill" && key != "fill_color" && key
!= "output"; // THE ONLY OPTIONAL ARGUMENTS

        if (is_required && !contains(entered_params, key))
                            throw_exception("Not all required " +
corresponding_options[0] + " arguments were provided.", 42);
    }

    return true;
}

std::unordered_map<std::string, std::vector<std::string>>
gen_corresponding_options()
{
          std::unordered_map<std::string, std::vector<std::string>>
corresponding_options;
          corresponding_options["triangle"] = {"triangle", "points",
"thickness", "color", "fill", "fill_color", "input", "output"};
          corresponding_options["biggest_rect"] = {"biggest_rect",
"old_color", "new_color", "input", "output"};
          corresponding_options["collage"] = {"collage", "number_x",
"number_y", "input", "output"};
```

```
          corresponding_options["line"] = {"line", "start", "end",
"thickness", "color", "input", "output"};
    corresponding_options["help"] = {"help"};

    return corresponding_options;
}

std::string get_function_to_exec(std::unordered_map<std::string, arg>
input_data)
{
    auto corresponding_options = gen_corresponding_options();
    std::string function_to_exec;
    for (const auto &it : corresponding_options)
    {
        const auto &key = it.first;
        const auto &val = it.second;
                    if (contains(input_data, key) &&
check_no_intersecting_flags(val, input_data))
        {
            function_to_exec = key;
        }
    }

    return function_to_exec;
}

void print_help()
{
    auto input_temp = gen_input_temp();
          std::unordered_map<std::string, option_temp>
input_temp_string_hash;
    for (const auto &it : input_temp)
    {
        const auto &opt = it.second;
        input_temp_string_hash[opt.long_name] = opt;
    }

     std::cout << "Course work for option 5.14, created by Zhavoronok
Danila.\n";
    auto corresponding_options = gen_corresponding_options();
    for (const auto &it : corresponding_options)
    {
        const auto &options = it.second;
        for (const auto &opt_name : options)
        {
            const auto &opt = input_temp_string_hash[opt_name];
            if (opt.long_name != it.first)
                std::cout << '\t';
                      std::cout << "--" << opt.long_name << " (" <<
opt.help_info << ")\n";
        }
        std::cout << '\n';
    }
}
```

draw_line.h

```
#pragma once
#include "../include/rgb.h"
#include <vector>
```

```
struct position
{
    int x = 0;
    int y = 0;

    position(int x, int y);
    position(const std::vector<int> &vec);
};

std::ostream &operator<<(std::ostream &os, const position &point);
struct Circle
{
    position pos;
    int r = 0;

    Circle(position pos, int r);
};

void draw_pixel(RGB *image, int width, int height, position pos, RGB
color);

void fill_circle(RGB *image, int width, int height, Circle c, RGB
color);

void draw_line(RGB *image, int width, int height, position a, position
b, int thickness, RGB color);
```

draw_line.cpp

```
#include "../include/draw_line.h"

position::position(int x, int y) : x(x), y(y) {}
position::position(const std::vector<int> &vec)
{
    if (vec.size() != 2)
        return;
    *this = position(vec[0], vec[1]);
}

Circle::Circle(position pos, int r) : pos(pos), r(r) {}

std::ostream &operator<<(std::ostream &os, const position &point)
{
    std::cout << '(' << point.x << ',' << point.y << ')';

    return os;
}

void draw_pixel(RGB *image, int width, int height, position pos, RGB
color)
{
    if (pos.x < 0 || pos.x >= width)
        return;
    if (pos.y < 0 || pos.y >= height)
        return;
    // std::cout << pos << '\n';
    image[pos.y * width + pos.x] = color;
}
```

```
void draw_circle_line(RGB *image, int width, int height, Circle c, RGB
color)
{
    int x = 0;
    int y = c.r;
    int d = 3 - 2 * y;
    while (y >= x)
    {
          draw_pixel(image, width, height, {c.pos.x + x, c.pos.y + y},
color);
          draw_pixel(image, width, height, {c.pos.x - x, c.pos.y + y},
color);
          draw_pixel(image, width, height, {c.pos.x + x, c.pos.y - y},
color);
          draw_pixel(image, width, height, {c.pos.x - x, c.pos.y - y},
color);
          draw_pixel(image, width, height, {c.pos.x + y, c.pos.y + x},
color);
          draw_pixel(image, width, height, {c.pos.x - y, c.pos.y + x},
color);
          draw_pixel(image, width, height, {c.pos.x + y, c.pos.y - x},
color);
          draw_pixel(image, width, height, {c.pos.x - y, c.pos.y - x},
color);
        x++;
        if (d < 0)
        {
            d += (4 * x) + 6;
        }
        else
        {
            d += 4 * (x - y) + 10;
            y--;
        }
    }
}

void draw_circle_filled(RGB *image, int width, int height, Circle c,
RGB color)
{
    draw_circle_line(image, width, height, c, color);

    for (int y = -c.r; y <= c.r; y++)
    {
        for (int x = -c.r; x <= c.r; x++)
        {
            if (x * x + y * y <= c.r * c.r)
                    draw_pixel(image, width, height, {c.pos.x + x,
c.pos.y}, color);
        }
    }
}
struct Rectangle
{
    int x_min;
    int y_min;
    int x_max;
    int y_max;
};
```

```
Rectangle get_line_bounding_box(position a, position b, int thickness)
{
    int half_thickness = thickness / 2;

    int x_min = std::min(a.x, b.x) - half_thickness;
    int y_min = std::min(a.y, b.y) - half_thickness;
    int x_max = std::max(a.x, b.x) + half_thickness;
    int y_max = std::max(a.y, b.y) + half_thickness;

    return {x_min, y_min, x_max, y_max};
}

bool rectangles_intersect(const Rectangle &r1, const Rectangle &r2)
{
     return !(r1.x_min > r2.x_max || r1.x_max < r2.x_min || r1.y_min >
r2.y_max || r1.y_max < r2.y_min);
}

void draw_line(RGB *image, int width, int height, position a, position
b, int thickness, RGB color)
{
    Rectangle image_rect = {0, 0, width - 1, height - 1};
    Rectangle line_rect = get_line_bounding_box(a, b, thickness);

    if (!rectangles_intersect(line_rect, image_rect))
        return;

    int skip = 200;

    int dx = abs(b.x - a.x);
    int dy = abs(b.y - a.y);
    int sx = (a.x < b.x ? 1 : -1) * (thickness / skip + 1);
    int sy = (a.y < b.y ? 1 : -1) * (thickness / skip + 1);
    int err = dx - dy;

      while (abs(a.x - b.x) > thickness / skip && abs(a.y - b.y) >
thickness / skip)
    {
          draw_circle_filled(image, width, height, {a, thickness / 2},
color);
        if (2 * err > -dy)
        {
            err -= dy;
            a.x += sx;
        }
        if (2 * err < dx)
        {
            err += dx;
            a.y += sy;
        }

        line_rect = get_line_bounding_box(a, b, thickness);
        if (!rectangles_intersect(line_rect, image_rect))
            return;
    }
}
```

biggest_rect.h

```
#pragma once
#include "../include/rgb.h"
#include "../include/draw_line.h"

void find_and_recolor_biggest_rect(RGB *image, int w, int h, const RGB
&oldColor, const RGB &newColor);
```

biggest_rect.cpp

```
#include "../include/biggest_rect.h"

struct Rectangle
{
    position pos;
    int width, height;
    int area()
    {
        return width * height;
    }
};

bool is_surrounded(RGB *image, int w, int h, const Rectangle &rect,
const RGB &oldColor)
{
    int rows = h;
    int cols = w;

    // Check top and bottom borders
    for (int j = rect.pos.x; j < rect.pos.x + rect.width; ++j)
    {
         if (rect.pos.y > 0 && image[(rect.pos.y - 1) * w + j] == old-
Color)
            return false;
           if (rect.pos.y + rect.height < rows && image[(rect.pos.y +
rect.height) * w + j] == oldColor)
            return false;
    }

    // Check left and right borders
    for (int i = rect.pos.y; i < rect.pos.y + rect.height; ++i)
    {
          if (rect.pos.x > 0 && image[i * w + rect.pos.x - 1] == old-
Color)
            return false;
         if (rect.pos.x + rect.width < cols && image[i * w + rect.pos.x
+ rect.width] == oldColor)
            return false;
    }

    return true;
}

void find_and_recolor_biggest_rect(RGB *image, int w, int h, const RGB
&oldColor, const RGB &newColor)
{
    Rectangle biggest_rect = {{0, 0}, 0, 0};

    std::vector<std::vector<int>> height(h, std::vector<int>(w, 0));
```

```
    std::vector<std::vector<int>> width(h, std::vector<int>(w, 0));

    for (int i = 0; i < h; ++i)
    {
        for (int j = 0; j < w; ++j)
        {
            if (image[i * w + j] == oldColor)
            {
                height[i][j] = (i == 0) ? 1 : height[i - 1][j] + 1;
                width[i][j] = (j == 0) ? 1 : width[i][j - 1] + 1;

                int minWidth = width[i][j];
                for (int k = 0; k < height[i][j]; ++k)
                {
                    minWidth = std::min(minWidth, width[i - k][j]);
                    int area = (k + 1) * minWidth;
                    if (area > biggest_rect.area())
                    {
                         Rectangle potential_rect = {{j - minWidth + 1,
i - k}, minWidth, k + 1};
                         if (is_surrounded(image, w, h, potential_rect,
oldColor))
                        {
                            biggest_rect = potential_rect;
                        }
                    }
                }
            }
        }
    }

    if (biggest_rect.area() > 0)
    {
            for (int i = biggest_rect.pos.y; i < biggest_rect.pos.y +
biggest_rect.height; ++i)
        {
              for (int j = biggest_rect.pos.x; j < biggest_rect.pos.x +
biggest_rect.width; ++j)
            {
                image[i * w + j] = newColor;
            }
        }
    }
}
```

make_collage.h

```
#pragma once
#include "../include/rgb.h"

RGB *make_collage(RGB *pixels, int width, int height, int N, int M,
int &cw, int &ch);
```

make_collage.cpp

```
#include "../include/make_collage.h"

RGB *make_collage(RGB *pixels, int w, int h, int N, int M, int &cw,
int &ch)
{
```

```
    cw = w * N;
    ch = h * M;

    RGB *collage = new RGB[cw * ch];
    for (int i = 0; i < ch; i++)
    {
        for (int j = 0; j < cw; j++)
        {
            int pixelsI = i % h;
            int pixelsJ = j % w;

            collage[i * cw + j] = pixels[pixelsI * w + pixelsJ];
        }
    }

    return collage;
}
```

draw_triangle.h

```
#pragma once
#include <vector>
#include "draw_line.h"

struct triangle
{
    std::vector<position> points;
    triangle(const std::vector<int> &vec);
    triangle(const std::vector<position> &vec);
    triangle(position a, position b, position c);
};

void draw_triangle(RGB *image, int width, int height, triangle tri,
int thickness, RGB color, bool fill, RGB fill_color);
```

draw_triangle.cpp

```
#include "../include/draw_triangle.h"
#include <algorithm>

triangle::triangle(position a, position b, position c)
{
    this->points.push_back(a);
    this->points.push_back(b);
    this->points.push_back(c);
}

triangle::triangle(const std::vector<int> &vec)
{
    if (vec.size() != 6)
        return;

    for (int i = 0; i < vec.size(); i += 2)
    {
        this->points.push_back({vec[i], vec[i + 1]});
    }
}

triangle::triangle(const std::vector<position> &vec)
{
```

```
    if (vec.size() != 3)
        return;

    for (int i = 0; i < vec.size(); i++)
    {
        this->points.push_back(vec[i]);
    }
}

std::ostream &operator<<(std::ostream &os, const std::vector<position>
&points)
{
    for (const auto &it : points)
    {
        std::cout << it << ' ';
    }
    std::cout << '\n';

    return os;
}

void draw_flat_horizontal_line(RGB *image, int width, int height,
position a, position b, RGB color)
{
    if (a.x > b.x)
        std::swap(a, b);
    for (int i = a.x; i <= b.x; i++)
    {
        draw_pixel(image, width, height, {i, a.y}, color);
    }
}

void fill_bottom_flat_triangle(RGB *image, int width, int height,
triangle tri, RGB fill_color)
{
    float curx1 = tri.points[0].x;
    float curx2 = tri.points[0].x;

    for (int y = tri.points[0].y; y <= tri.points[1].y; y++)
    {
          draw_flat_horizontal_line(image, width, height, {int(curx1),
y}, {int(curx2), y}, fill_color);
                curx1 += float(tri.points[1].x - tri.points[0].x) /
(tri.points[1].y - tri.points[0].y); // Add slope val
                curx2 += float(tri.points[2].x - tri.points[0].x) /
(tri.points[2].y - tri.points[0].y); // Add slope val
    }
}

void fill_top_flat_triangle(RGB *image, int width, int height,
triangle tri, RGB fill_color)
{
    float curx1 = tri.points[2].x;
    float curx2 = tri.points[2].x;

    for (int y = tri.points[2].y; y > tri.points[0].y; y--)
    {
          draw_flat_horizontal_line(image, width, height, {int(curx1),
y}, {int(curx2), y}, fill_color);
```

```
                curx1 -= float(tri.points[2].x - tri.points[0].x) /
float(tri.points[2].y - tri.points[0].y); // Add slope val
                curx2 -= float(tri.points[2].x - tri.points[1].x) /
float(tri.points[2].y - tri.points[0].y); // Add slope val
    }
}

bool position_y_sorter(position const &lhs, position const &rhs)
{
    return lhs.y < rhs.y;
}

void draw_triangle(RGB *image, int width, int height, triangle tri,
int thickness, RGB color, bool fill, RGB fill_color)
{
    if (fill)
    {
                    std::sort(tri.points.begin(), tri.points.end(),
&position_y_sorter);
        if (tri.points[1].y == tri.points[2].y)
        {
                  fill_bottom_flat_triangle(image, width, height, tri,
fill_color);
        }
        else if (tri.points[0].y == tri.points[1].y)
        {
                    fill_top_flat_triangle(image, width, height, tri,
fill_color);
        }
        else
        {
               position side_point = {(int)(tri.points[0].x + ((float)
(tri.points[1].y - tri.points[0].y) /
                                                                (float)
(tri.points[2].y - tri.points[0].y)) *

(tri.points[2].x - tri.points[0].x)),
                                    tri.points[1].y};
                      fill_bottom_flat_triangle(image, width, height,
{tri.points[0], tri.points[1], side_point}, fill_color);
                         fill_top_flat_triangle(image, width, height,
{tri.points[1], side_point, tri.points[2]}, fill_color);
        }
    }

    for (int i = 0; i < 3; i++)
    {
        position a = tri.points[i];
        position b = tri.points[(i + 1) % 3];
        draw_line(image, width, height, a, b, thickness, color);
    }
}
```

main.cpp

```
#define STB_IMAGE_IMPLEMENTATION
#define STB_IMAGE_WRITE_IMPLEMENTATION
#include "lib/stb_image.h"
#include "lib/stb_image_write.h"
```

```
#include "include/input.h"
#include "include/utils.h"
#include "include/rgb.h"
#include "include/biggest_rect.h"
#include "include/make_collage.h"
#include "include/draw_line.h"
#include "include/draw_triangle.h"

#include <stdlib.h>
#include <string>

int main(int argc, char **argv)
{
    auto input_data = input(argc, argv);
    auto function_to_exec = get_function_to_exec(input_data);
    // std::cout << function_to_exec << '\n';

    if (function_to_exec == "help")
    {
        print_help();
        return 0;
    }

    std::string inputPath = input_data["input"].values[0];
    std::string outputPath = "output.png";
    if (contains(input_data, "output"))
    {
        outputPath = input_data["output"].values[0];
    }
    int width = 0, height = 0;
     RGB *image = reinterpret_cast<RGB *>(stbi_load(inputPath.c_str(),
&width, &height, NULL, 3));
    if (width == 0 || height == 0)
    {
         throw_exception("File provided as input is not a PNG image.",
45);
    }

    if (function_to_exec == "triangle")
    {
           triangle tri = parse_input(input_data["points"].values[0],
no_check, 6);
                                const auto &thickness =
parse_input(input_data["thickness"].values[0], check_N_val, 1);
              RGB color = parse_input(input_data["color"].values[0],
check_rgb_val, 3);
        bool fill = contains(input_data, "fill");
                  RGB fill_color = parse_input(contains(input_data,
"fill_color") ? input_data["fill_color"].values[0] : "0.0.0",
check_rgb_val, 3);

         draw_triangle(image, width, height, tri, thickness[0], color,
fill, fill_color);

           stbi_write_png(outputPath.c_str(), width, height, 3, image,
width * 3);
    }
    else if (function_to_exec == "biggest_rect")
    {
```

```
          const RGB oc = parse_input(input_data["old_color"].values[0],
check_rgb_val, 3);
          const RGB nc = parse_input(input_data["new_color"].values[0],
check_rgb_val, 3);

        find_and_recolor_biggest_rect(image, width, height, oc, nc);

           stbi_write_png(outputPath.c_str(), width, height, 3, image,
width * 3);
    }
    else if (function_to_exec == "collage")
    {
          const auto &N = parse_input(input_data["number_x"].values[0],
check_N_val, 1);
          const auto &M = parse_input(input_data["number_y"].values[0],
check_N_val, 1);
        int cw, ch;
         RGB *collage = make_collage(image, width, height, N[0], M[0],
cw, ch);
           stbi_write_png(outputPath.c_str(), cw, ch, 3, collage, cw *
3);
    }
    else if (function_to_exec == "line")
    {
              RGB color = parse_input(input_data["color"].values[0],
check_rgb_val, 3);
           position start = parse_input(input_data["start"].values[0],
no_check, 2);
              position end = parse_input(input_data["end"].values[0],
no_check, 2);

                                  const auto &thickness =
parse_input(input_data["thickness"].values[0], check_N_val, 1);

            draw_line(image, width, height, start, end, thickness[0],
color);

           stbi_write_png(outputPath.c_str(), width, height, 3, image,
width * 3);
    }

    return 0;
}
```