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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Обработка PNG изображений

Студентка гр. 3344 _____ Коняева М.В.

Преподаватель _____ Глазунов С.А.

Санкт-Петербург

2024

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студентка Коняева М.В.

Группа 3344

Тема работы: Обработка *PNG* изображений

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 окружности. Флаг для выполнения данной операции: `--*circle*`. Окружность определяется:
    - координатами ее центра и радиусом. 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 линии окружности. Флаг `--*thickness*`. На вход принимает число больше 0
- цветом линии окружности. Флаг `--*color*` (цвет задаётся строкой `*rrr.ggg.bbb*`, где *rrr/ggg/bbb* – числа, задающие цветовую компоненту. пример `--*color* 255.0.0` задаёт красный цвет)
- окружность может быть залитой или нет. Флаг `--*fill*`. Работает как бинарное значение: флага нет – *false*, флаг есть – *true*.
- цветом которым залита сама окружность, если пользователем выбрана залитая окружность. Флаг `--*fill_color*` (работает аналогично флагу `--*color*`)

2. Фильтр *rgb*-компонент. Флаг для выполнения данной операции: `--*rgbfilter*`. Этот инструмент должен позволять для всего изображения либо установить в диапазоне от 0 до 255 значение заданной компоненты. Функционал определяется
   - Какую компоненту требуется изменить. Флаг `--*component_name*`. Возможные значения `*red*`, `*green*` и `*blue*`.
   - В какой значение ее требуется изменить. Флаг `--*component_value*`. Принимает значение в виде числа от 0 до 255
3. Разделяет изображение на *N*M* частей. Флаг для выполнения данной операции: `--*split*`. Реализация: провести линии заданной толщины. Функционал определяется:
   - Количество частей по “оси” *Y*. Флаг `--*number_x*`. На вход принимает число больше 1
   - Количество частей по “оси” *X*. Флаг `--*number_y*`. На вход принимает число больше 1
   - Толщина линии. Флаг `--*thickness*`. На вход принимает число больше 0
   - Цвет линии. Флаг `--*color*` (цвет задаётся строкой `*rrr.ggg.bbb*`, где

rrr/ggg/bbb – числа, задающие цветовую компоненту. пример `--*color* 255.0.0` задаёт красный цвет)

4. Рисование квадрата с диагоналями. Флаг для выполнения данной операции: `--*squared_lines*`. Квадрат определяется:
    - Координатами левого верхнего угла. Флаг `--*left_up*`, значение задаётся в формате `*left.up*`, где *left* – координата по *x, up* – координата по y
    - Размером стороны. Флаг `--*side_size*`. На вход принимает число больше 0
    - Толщиной линий. Флаг `--*thickness*`. На вход принимает число больше 0
    - Цветом линий. Флаг `--*color*` (цвет задаётся строкой `*rrr.ggg.bbb*`, где *rrr/ggg/bbb* – числа, задающие цветовую компоненту. пример `--*color* 255.0.0` задаёт красный цвет)
    - Может быть залит или нет (диагонали располагаются “поверх” заливки). Флаг `--*fill*`. Работает как бинарное значение: флага нет – *false*, флаг есть – *true*.
    - Цветом которым он залит, если пользователем выбран залитый. Флаг `--*fill_color*` (работает аналогично флагу `--*color*`)

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

Содержание пояснительной записки:

Разделы пояснительной записки: Содержание, Введение, Заключение, Список использованных источников.

Предполагаемый объем пояснительной записки:

Не менее 50 страниц.

Дата выдачи задания: 18.03.2023

Дата сдачи реферата: 20.05.2023

Дата защиты реферата: 22.05.2023

Студентка _____ Коняева М.В.

Преподаватель _____ Глазунов С.А.

АННОТАЦИЯ

Курсовая работа представляет собой программу, которая обрабатывает *PNG*-изображение. Программа имеет *CLI* (интерфейс командной строки) для ввода параметров обработки *PNG*-файла пользователем.

Для чтения и записи изображения была использована библиотека *libpng*; для обработки изображения использовались функции стандартных библиотек; для анализов аргументов командной строки использовалась библиотека *getopt*. Используемый язык программирования Си.

SUMMARY

The coursework is a program that processes a PNG image. The program has a CLI (command line interface) for entering parameters for processing a PNG file by the user.

The libpng library was used to read and write the image; the functions of standard libraries were used to process the image; the getopt library was used to analyze command-line arguments. The C programming language used.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель данной работы — разработка программы на языке Си для обработки PNG-изображений. Для достижения поставленной цели требуется решить следующие задачи:

- разработать функции чтения и записи *PNG*-файла, реализовать записи в структуру *Png*;
- реализовать интерфейс с помощью библиотеки *getopt*;
- разработать функцию рисования круга на изображении;
- разработать функцию изменения значения заданной компоненты;
- разработать функцию, которая разделяет изображение на заданное количество частей;
- разработать функцию рисования квадрата с диагоналями;
- написать *Makefile* для удобной сборки проекта;
- протестировать разработанную программу.

1. ПОДКЛЮЧАЕМЫЕ БИБЛИОТЕКИ, МАКРООПРЕДЕЛЕНИЯ, СТРУКТУРЫ

Для корректной работы программы подключены стандартные библиотеки языка Си: stdlib.*h, stdio.h, math.h, ctype.h, string.h, regex.h, string.h.*

Также подключена библиотека *png.h* для чтения и записи *PNG*-файла и библиотека *getopt.h* для анализа аргументов командной строки.

Определены две структуры:

- Структура *Png,* хранящая параметры изображения: высоту *height* и ширину *width*, цветовой тип *color_type*, битовую глубину *bit_depth*, количество проходов, необходимых для декодирования изображения в случае использования интерлейсинга number_of_passes, количество каналов *channels*, указатель на *png_struct*, указатель на *png_info*, указатель на сетку пикселей;
- Структура *Configs*, содержащая все аргументы, передаваемые в командную строку.

2. ФУНКЦИИ

2.1. Функции чтения и записи PNG-файла

Функция *read_png_file()* принимает на вход указатель на строку *file_name* – имя *PNG*-файла, который нужно считать, а также указатель на структуру *Png img*; с помощью функций из библиотеки *libpng* данные из *IHDR* считываются и записываются в структуру *image*; также происходит динамическое выделение памяти для сетки пикселей с последующей записью в структуру *img*; если на каком-либо этапе считывания *PNG*-файла возникает ошибка, то выводится сообщение о том, какую именно часть файла не удалось считать, и программа завершается.

Функция *write_png_file()* принимает на вход указатель на строку *file_name* – имя *PNG*-файла, куда требуется записать изображение, а также указатель на структуру *Png img*; с помощью функций из библиотеки *libpng* информация о изображении, а также сетка пикселей записывается в *PNG*-файл. Если на этапе записи *PNG*-файла возникает ошибка, то она корректно обрабатывается: выводится сообщение о том, что именно не удалось записать, и программа завершается.

2.2. Дополнительные, вспомогательные функции

Функция *create_default()* принимает на вход указатель *conf* на структуру *Configs* и заполняет поля структуры начальными значениями.

Функция *free_conf()* принимает на вход указатель *conf* на структуру *Configs* и очищает память, которая была выделена под поля структуры.

Функция *free_image_data()* принимает на вход указатель *img* на структуру *Png* и очищает динамическую память, выделенную для хранения сетки пикселей Принцип работы: с помощью цикла *for* проходит по каждой строке пикселей и

освобождает динамически выделенную ранее память; далее освобождает память, выделенную под хранение указателей на строки пикселей.

Функция *print_help()* печатает в поток вывода справку по работе с программой.

Функция *print_png_info()* принимает на вход указатель *img* на структуру *Png* и печатает в поток вывода основную информацию о *PNG*-файле.

Функция *void coords_checker()* принимает на вход строку и поверяет с помощью регулярных выражений корректность строки. Функция выводит ошибку, если в строке не два числа через точку.

Функция *void distance_checker()* принимает на вход строку и поверяет с помощью регулярных выражений корректность строки. Функция выводит ошибку, если в строке неположительное число.

Функция *int *parse_color()* принимает на вход строку. Происходит запись и разделение строки на массив чисел с помощью функции *strtok*. Функция возвращает полученный массив чисел.

Функция *void color_checker(char *color)* принимает на вход строку с информацией о цвете и проверяет ее корректность. Если хотя бы одно из значений выходит из диапазона от 0 до 255, выводится ошибка.

Функция *void names_checker()* принимает на вход две строки, которые содержат имена входного и выходного файлов. Если имена совпадают, то выводится ошибка.

Функция *void no_argschecker()* принимает на вход две строки, которые содержат поданный аргумент и флаг. Если аргумент не пустой и является не следующей командой, то выводится ошибка. Данная функция используется, когда флаг должен быть введен без аргументов.

Функция *is_pixel_part_of_img()* принимает на вход указатель *img* на структуру *Png*, координаты пикселя *x* и *y*. Далее проверяется, является ли данный пиксель частью изображения. Если является, то возвращается значение 1, иначе – 0. С помощью условного оператора *if* соответствующие координаты сравниваются с высотой и шириной изображения, проверяется нахождение пикселя внутри изображения; возвращается результат сравнения.

Функция *set_pixel()* принимает на вход указатель *img* на структуру *Png*, координаты пикселя *x* и *y*, указатель на массив с цветом *color_arr*; изменяет цвет пикселя на переданный в функцию цвет. Функция переходит в сетке пикселей к заданному координатами пикселю и изменяет значения трех каналов (*RGB*) в соответствии с переданным цветом *color_arr*.

Функция *draw_contour_circle()* принимает на вход указатель *img* на структуру *Png*, координаты центра окружности *xc* и *yc*, координаты *x* и *y*, указатель на массив с цветом *color_arr*. Для рисования окружности используется алгоритм Брезенхема. На каждом шаге алгоритма рассматриваются три пикселя, из которых выбирается наиболее подходящий путём сравнения расстояний от центра до выбранного пикселя с радиусом окружности. Затем координаты наиболее подходящего пикселя передаются во вспомогательную функцию *draw_in_octants*, которая отвечает за перекрашивание восьми симметричных пикселей в каждой из восьми частей окружности. Таким образом рисуется контур окружности.

Функция *draw_in_octants()* принимает на вход указатель *img* на структуру *Png*, координаты центра окружности *xc* и *yc*, координаты *x* и *y*, указатель на массив с цветом *color_arr*. Данная функция является вспомогательной для рисования окружности. В функции вызывается ранее описанная функция *set_pixel()*, которая изменяет цвет каждого симметричного пикселя в каждой из восьми частей окружности.

Функция *fill_circle()* принимает на вход указатель *img* на структуру *Png*, координаты центра окружности *xc* и *yc*, радиус *r* и указатель на массив с цветом *color_arr*; заливает заданную окружность цветом *color_arr*. Данная функция является вспомогательной для рисования залитого круга. С помощью цикла *for* в цикле *for* проходит по каждому пикселю в квадрате, в котором вписана окружность, а с помощью уравнения окружности $(x - xc)^2 + (y - yc)^2 \leq r^2$ проверяет, находится ли текущий пиксель внутри окружности, и если находится, то к текущему пикселю применяется функция *set_pixel* для изменения его цвета.

Функция *void draw_thick_circle()* принимает на вход указатель *img* на структуру *Png*, координаты центра окружности *xc* и *yc*, радиус *r*, толщину *th* и указатель на массив с цветом *color_arr*. Данная функция является вспомогательной для рисования круга. С помощью цикла *for* в цикле *for* проходит по каждому пикселю в квадратах, в которые вписаны внутренний и внешний контуры. Проверяет, находится ли текущий пиксель внутри нужной области, и если находится, то к текущему пикселю применяется функция *set_pixel* для изменения его цвета.

Функция *draw_thick_line()* принимает на вход указатель *img* на структуру *Png*, координаты двух точек *x1* и *y1*, *x2* и *y2*, толщину *thick*, указатель на массив с цветом *color_arr*; рисует линию по заданным параметрам. Для рисования линии используется алгоритм Брезенхема. Сначала вычисляются приращения, абсолютные значения и направления рисования по осям X и Y. Затем в цикле

while происходит движение от одной точки к другой путём обновления значения ошибки, сравнения этого значения с приращением по осям и добавления к координатам точки значений направления. На каждом шаге цикла вычисляются координаты пикселей, составляющих линию, и эти координаты передаются в функцию *fill_circle()* для непосредственного рисования линии нужной толщины на изображении. Аналогичным образом работает функция *draw_line().* Однако координаты передаются в функцию *set_pixel()* для рисования линии толщиной 1.

2.3. Основные функции

Функция *void draw_circle()* принимает на вход указатель *img* на структуру *Png*, указатель на массив координат центра окружности *center*, радиус *r*, толщину *thick*, указатель на массив с цветом *th_color*, флаг заливки *flag_fill*, указатель на массив с цветом заливки *fill_color*. Если флаг заливки не равен нулю, вызывается функция *fill_circle()*. Для того чтобы нарисовать толщину, рисуется внешняя и внутренняя обводка с помощью функции *draw_contour_circle()*. Далее пространство между ними заполняется цветом толщины контура функцией *draw_thick_circle().*

Функция *switch_filter()* принимает на вход указатель *img* на структуру *Png,* строку, содержащую название компоненты *comp_name* и значение компоненты *comp_value*. Сравнивает с помощью функции *strstr* строки *red, green* или *blue* с заданным названием компоненты и присваивает ей значение.

Функция *divide_image()* принимает на вход количество частей по оси Y, количество частей по оси X, толщину линии *thick,* указатель на массив с цветом *color_arr.* Заранее вычисляется ширина и высота полос. Для этого размеры изображения делятся на поданные значения. Далее с помощью двух циклов *for* и функции *draw_lin*e рисуются линии нужной толщины.

Функция *draw_square()* принимает на вход указатель *img* на структуру *Png*, указатель на массив координат левого верхнего угла квадрата *center*, размер стороны *side*, толщину *thick*, указатель на массив с цветом *color_arr*, флаг *fill* и указатель на массив с цветом заливки *fill_color*. Если флаг заливки равен не нулю, то с помощью двух циклов *for* для нужных координат вызывается функция *set_pixel*, которая закрашивает пиксели внутри области. Далее рисуются контуры квадрата и его диагонали с помощью ранее описанной функции *draw_thick_line()*.

2.4. Функция *main*

В функции *main()* реализовано управление программой с использованием аргументов командной строки. Используя функции из библиотеки *getopt*, программа считывает параметры для обработки PNG-изображения, соответствующие указанным флагам, проверяет их на корректность. Если введён некорректный параметр, неверный флаг или недостаточное количество аргументов, программа выводит сообщение об ошибке и завершает выполнение. Правильные параметры сохраняются в структуру *conf Configs*. Инструкции по использованию программы, доступные флаги и передаваемые значения можно увидеть, если не передать аргументы или передать флаг *–help (-h)*. Затем происходит чтение PNG-файла и обработка изображения согласно переданным параметрам. Обработанное изображение сохраняется в новый PNG-файл, и программа завершает работу.

2.5. Makefile

Файлы делятся на два типа: файлы с расширением .c, которые содержат исходный код и заголовочные файлы с расширение .h, которые содержат нужные библиотеки и сигнатуры функций. Файлы с содержанием исходного кода*: functions.c, input.c, checker.c.* У каждого из них есть одноименный заголовочный файл. Файл *main.c* содержит главную функцию *main*, которая управляет

выполнением программы, вызывая другие её функции. Заголовочный файл *library.h* содержит объявление структур и библиотек.

Для сборки проекта используется *Makefile*. В make-файле прописана цель *all*, и её функция *gcc *.o -lpng -o cw -std=c99*, которая занимается линковкой объектных модулей, в результате будет получен исполняемый модуль cw. Также прописаны другие цели для компиляции и получения объектных модулей. При компиляции объектных файлов используется флаг *-std=c99* (стандарт языка C99). Последняя цель *clean* позволяет очищать директиву от всех объектных модулей.

Результаты тестирования см. в приложении А.
Разработанный программный код см. в приложении Б.

ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы была успешно создана программа, которая управляется с помощью аргументов командной строки. Программа осуществляет считывание PNG-изображения и выполняет его обработку на основе переданных параметров. Виды обработки изображения: рисование окружности и квадрата с диагоналями, изменение компоненты цвета, разделение изображения на части. В ходе выполнения задания были улучшены навыки работы с функциями библиотек *libpng* и *getopt*. Полученные результаты подтверждают успешное достижение поставленной цели.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Керниган Б., Ритчи Д., Язык программирования Си.: Издательство Москва, Вильямс, 2015 г. 304 с.
2. Онлайн-библиотека // Википедия. URL: https://ru.wikipedia.org/wiki/Алгоритм_Брезенхэма#:~:text=Алгоритм%20Брезенхе́ма%20(англ.,разработан%20Джеком%20Элтоном%20Брезенхэмом%20(англ. (дата обращения 17.05.2023).
3. Веб-сайт системы вопросов и ответов // stackoverflow. URL: https://en.cppreference.com (дата обращения 16.05.2023).
4. Мануал по работе с библиотекой libpng // libpng.org. URL: http://www.libpng.org/pub/png/libpng-1.2.5-manual.html (дата обращения 15.05.2023).
5. Электронный учебник по программированию на языках Си и C++ // cppstudio. URL: http://cppstudio.com/ (дата обращения 17.05.2023).

ПРИЛОЖЕНИЕ А
РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ

1. Вывод справки (./cw):

```
marina@marina-ASUS-TUF-Dash-F15-FX516PC-FX516PC:~/Рабочий стол/make$ ./cw
Course work for option 5.17, created by Marina Konyaeva.
Course work for option 5.17, created by Marina Konyaeva.

Вспомогательные функции:

            -h, -help - справка, которую вы видите сейчас

            -info - подробная информация об изображении

            -i, -input - задаёт имя входного изображения

            -o, -output - задаёт имя выходного изображения

Функции по обработке изображений:

            --circle - рисование окружности
                --center - координаты центра
                ---radius - радиус окружности
                --thickness - толщина обводки
                --color - цвет обводки
                --fill - флаг заливки
                --fill_color - цвет заливки
            --rgbfilter - фильтр rgb-компонент
                --component_name - имя изменяемоей компоненты
                --component_value - новое значение изменяемой компоненты
            --split - разделяет изображение на N*M частей
                --number_x - количество частей по оси Y
                --number_y - количество частей по оси X
                --thickness - толщина линии
                --color - цвет линий
            --squared_lines - рисование квадрата с диагоналями
                --left_up - координаты левого верхнего угла
                --side_size - размер стороны
                --thickness - толщина линий
                --color - цвет линий
                --fill - флаг заливки
                --fill_color - цвет заливки
```

Рисунок 1.1

2. Обработка изображения: рисование залитой окружности (./cw --input tes.png --output output.png --color 255.255.0 --thickness 50 --radius 150 --center 550.700 --circle --fill --fill_color 255.0.255):

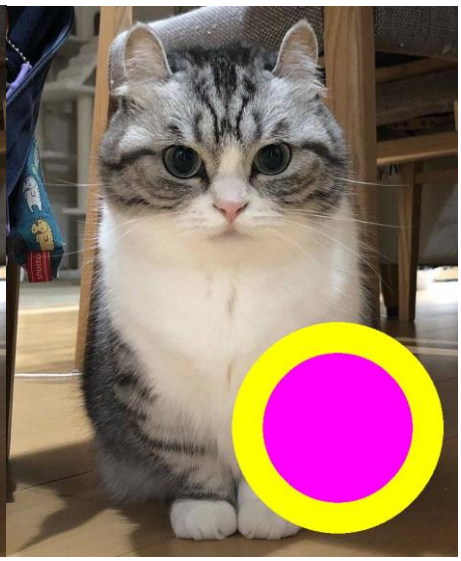

Рисунок 2.1 (до обработки) *Рисунок 2.2 (после обработки)*

3. Обработка изображения: смена rgb-компоненты(./cw -i img.png -o result.png –rgbfilter –component_name red –component_value 210):

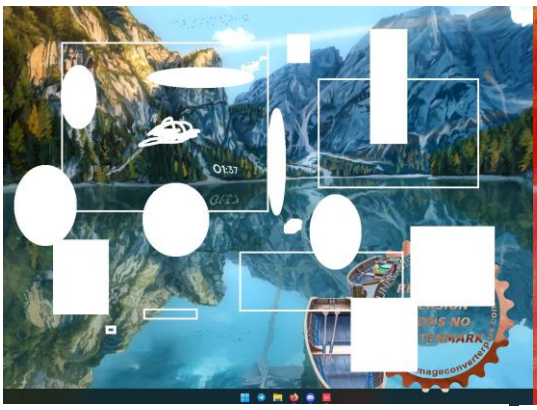

Рисунок 3.1 (до обработки)

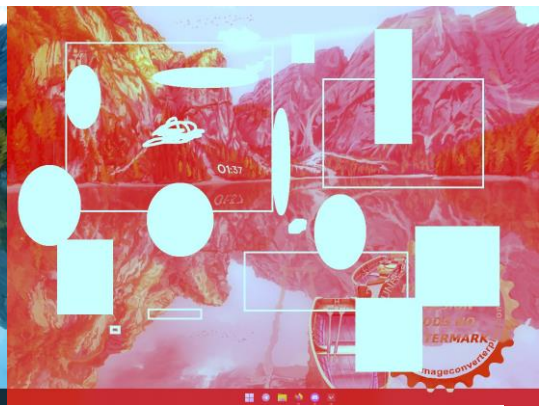

Рисунок 3.2 (после обработки)

4. Обработка изображения: разделение изображения на N*M частей (./cw --input e.png --output output.png --color 0.0.255 --thickness 5 --number_x 6 --number_y 4 --split):

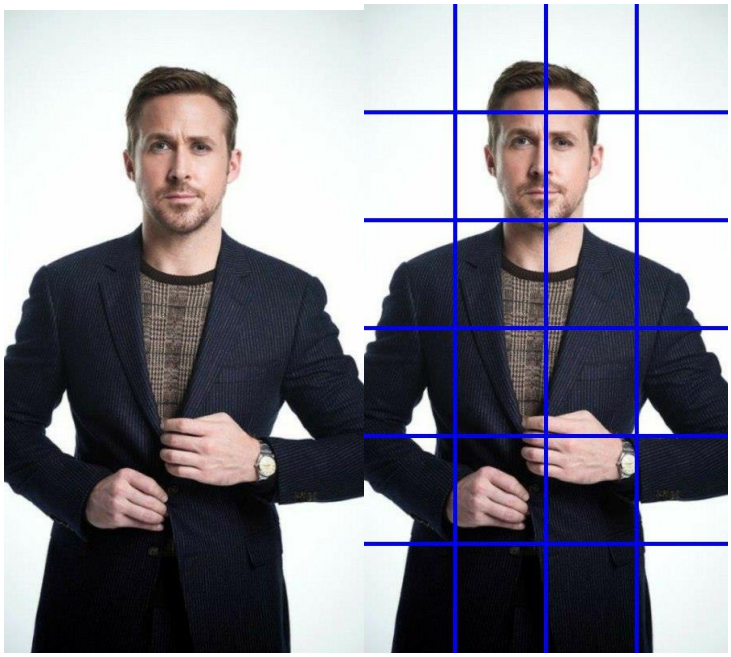

Рисунок 4.1 (до обработки) *Рисунок 4.2 (после обработки)*

5. Обработка изображения: рисование квадрата с диагоналями (./cw collage -o result.png -i img.png --collage --height 3 --width 4):

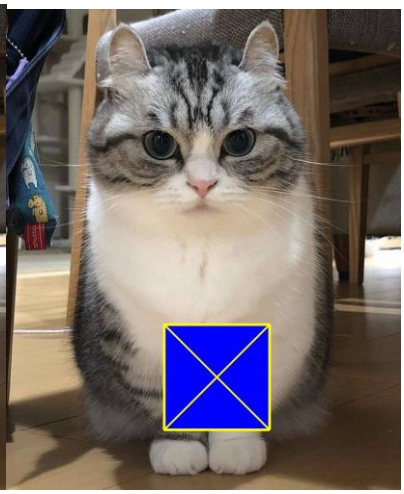

Рисунок 5.1 (до обработки) *Рисунок 5.2 (после обработки)*

6. Обработка некорректных флагов и аргументов:

```
Course work for option 5.17, created by Marina Konyaeva.
--squared_lines doesn't have arguments
marina@marina-ASUS-TUF-Dash-F15-FX516PC-FX516PC:~/Рабочий стол/make$ ./cw --input input.png --output input.png --color 255.255.0 --thickness 5 --
side_size 200 --fill --fill_color 0.0.250 --squared_lines  --leftup 300.300
Course work for option 5.17, created by Marina Konyaeva.
./cw: unrecognized option '--leftup'
Arguments entered incorrectly.
marina@marina-ASUS-TUF-Dash-F15-FX516PC-FX516PC:~/Рабочий стол/make$
```

Рисунок 6.1

```
Course work for option 5.17, created by Marina Konyaeva.
Error in names_checker: Input and output files are the same
marina@marina-ASUS-TUF-Dash-F15-FX516PC-FX516PC:~/Рабочий стол/make$ ./cw --input input.png --output output.png --color 255.255.0 --thickness 5 -
-side_size 200  --squared_lines
Course work for option 5.17, created by Marina Konyaeva.
Not enough arguments have been entered to draw a square.
marina@marina-ASUS-TUF-Dash-F15-FX516PC-FX516PC:~/Рабочий стол/make$
```

Рисунок 6.2

7. Обработка некорректных параметров:

```
marina@marina-ASUS-TUF-Dash-F15-FX516PC-FX516PC:~/Рабочий стол/make$ ./cw --input input.png --output output.png --color 255.255.0 --thickness 5 -
-side_size 200 --fill --fill_color 0.0.300 --squared_lines --left_up 300.300
Course work for option 5.17, created by Marina Konyaeva.
The entered color is incorrect.
```

Рисунок 7.1

```
marina@marina-ASUS-TUF-Dash-F15-FX516PC-FX516PC:~/Рабочий стол/make$ ./cw --input input.png --output output.png --color 255.255.0 --thickness -1
--side_size 200 --fill --fill_color 0.0.250 --squared_lines --left_up 300.300
Course work for option 5.17, created by Marina Konyaeva.
Error in distance_checker: digit is incorrect
```

Рисунок 7.2

```
marina@marina-ASUS-TUF-Dash-F15-FX516PC-FX516PC:~/Рабочий стол/make$ ./cw --input input.png --output input.png --color 255.255.0 --thickness 5 --
side_size 200 --fill --fill_color 0.0.250 --squared_lines --left_up 300.300
Course work for option 5.17, created by Marina Konyaeva.
Error in names_checker: Input and output files are the same
```

Рисунок 7.3

```
marina@marina-ASUS-TUF-Dash-F15-FX516PC-FX516PC:~/Рабочий стол/make$ ./cw --input input.png --output input.png --color 255.255.0 --thickness 5 --
side_size 200 --fill --fill_color 0.0.250 --squared_lines djfe  --left_up 300.300
Course work for option 5.17, created by Marina Konyaeva.
--squared_lines doesn't have arguments
```

Рисунок 7.4

ПРИЛОЖЕНИЕ Б

ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл: library.h

```
#pragma once
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <ctype.h>
#include <string.h>
#include <png.h>
#include <getopt.h>
#include <regex.h>
#include <string.h>

struct Png{
    int width, height;
    png_byte color_type;
    png_byte bit_depth;
    png_byte channels;

    png_structp png_ptr;
    png_infop info_ptr;
    int number_of_passes;
    png_bytep *row_pointers;
};

struct Configs{
     int x_y[2];
    int radius;
    int thickness;

    int* color;
    int fill;
    int* fill_color;

    char* component_name;
    int component_value;

    int number_x;
    int number_y;

    int side_size;
};
```

Файл: input.h

```
#pragma once
#include "library.h"


void read_png_file(char *file_name, struct Png *img);
void write_png_file(char *file_name, struct Png *img);
void print_png_info(struct Png *img);
void print_help();
void free_image_data(struct Png *img);
```

```
void create_default(struct Configs* conf);
void free_conf(struct Configs* conf);
```

Файл: input.c

```
#include "input.h"

void read_png_file(char *file_name, struct Png *img) {

    char header[8];    // 8 is the maximum size that can be checked
    /* open file and test for it being a png */

    FILE *fp = fopen(file_name, "rb");
    if (!fp){
        printf("Error in read_png_file function: file could not be
opened.\n");
        exit(40);
    }

    fread(header, 1, 8, fp);
    if (png_sig_cmp(header, 0, 8)){
        printf("Error  in  read_png_file  function:  file  is  not
recognized as a PNG.\n");
        exit(41);
    }

    /* initialize stuff */
    img->png_ptr   =   png_create_read_struct(PNG_LIBPNG_VER_STRING,
NULL, NULL, NULL);

    if (!img->png_ptr){
        printf("Error       in       read_png_file       function:
png_create_read_struct failed.\n");
        exit(42);
    }

    img->info_ptr = png_create_info_struct(img->png_ptr);
    if (!img->info_ptr){
        printf("Error       in       read_png_file       function:
png_create_info_struct failed.\n");
        exit(40);
    }
    // что это зачем оно нада как это работает
    if (setjmp(png_jmpbuf(img->png_ptr))){
        fclose(fp);
        printf("Error  in  read_png_file  function:  error  during
init_io.\n");
        exit(43);
    }

    png_init_io(img->png_ptr, fp);
    png_set_sig_bytes(img->png_ptr, 8);

    png_read_info(img->png_ptr, img->info_ptr);

    img->width = png_get_image_width(img->png_ptr, img->info_ptr);
    img->height = png_get_image_height(img->png_ptr, img->info_ptr);
```

```
        img->color_type       =       png_get_color_type(img->png_ptr,
img->info_ptr);
        img->bit_depth = png_get_bit_depth(img->png_ptr, img->info_ptr);
        //есть channels а есть numofpasses
        img->number_of_passes                                          =
png_set_interlace_handling(img->png_ptr);
        img->channels = png_get_channels(img->png_ptr, img->info_ptr);

        if (img->color_type == PNG_COLOR_TYPE_GRAY) {
            printf("The program does not support working with the
PNG_COLOR_TYPE_GRAY color type.\n");
            exit(44);
        } else if (img->color_type == PNG_COLOR_TYPE_GRAY_ALPHA) {
            printf("The program does not support working with the
PNG_COLOR_TYPE_GRAY_ALPHA color type.\n");
            exit(45);
        } else if (img->color_type == PNG_COLOR_TYPE_PALETTE) {
            printf("The program does not support working with the
PNG_COLOR_TYPE_PALETTE color type.\n");
            exit(46);
        }

        png_read_update_info(img->png_ptr, img->info_ptr);

        /* read file */
        if (setjmp(png_jmpbuf(img->png_ptr))){
            printf("Error in read_png_file function: error during
read_img.\n");
            exit(47);
        }

        img->row_pointers = (png_bytep *) malloc(sizeof(png_bytep) *
img->height);
        for (int y = 0; y < img->height; y++)
            img->row_pointers[y]        =        (png_byte        *)
malloc(png_get_rowbytes(img->png_ptr, img->info_ptr));

        png_read_image(img->png_ptr, img->row_pointers);

        fclose(fp);
    }


    void write_png_file(char *file_name, struct Png *img) {

        /* create file */
        FILE *fp = fopen(file_name, "wb");
        if (!fp){
            printf("Error in write_png_file function: file could not be
opened.\n");
            exit(40);
        }

        /* initialize stuff */
        img->png_ptr  =  png_create_write_struct(PNG_LIBPNG_VER_STRING,
NULL, NULL, NULL);
```

```
        if (!img->png_ptr){
            printf("Error       in       write_png_file       function:
png_create_write_struct failed.\n");
            exit(41);
        }

        img->info_ptr = png_create_info_struct(img->png_ptr);
        if (!img->info_ptr){
            printf("Error       in       write_png_file       function:
png_create_info_struct failed.\n");
            exit(42);
        }

        if (setjmp(png_jmpbuf(img->png_ptr))){
            printf("Error  in  write_png_file  function:  error  during
init_io.\n");
            exit(43);
        }

        png_init_io(img->png_ptr, fp);


        /* write header */
        if (setjmp(png_jmpbuf(img->png_ptr))){
            fclose(fp);
            printf("Error  in  write_png_file  function:  error  during
writing header.\n");
            exit(44);
        }

        png_set_IHDR(img->png_ptr,      img->info_ptr,      img->width,
img->height,
                     img->bit_depth,                   img->color_type,
PNG_INTERLACE_NONE,
                     PNG_COMPRESSION_TYPE_BASE, PNG_FILTER_TYPE_BASE);

        png_write_info(img->png_ptr, img->info_ptr);


        /* write bytes */
        if (setjmp(png_jmpbuf(img->png_ptr))){
            fclose(fp);
            printf("Error  in  write_png_file  function:  error  during
writing bytes.\n");
            exit(45);
        }

        png_write_image(img->png_ptr, img->row_pointers);


        /* end write */
        if (setjmp(png_jmpbuf(img->png_ptr))){
            fclose(fp);
            printf("Error  in  write_png_file  function:  error  during
writing end of file.\n");
            exit(46);
        }
```

```
        png_write_end(img->png_ptr, NULL);

        /* cleanup heap allocation */
        for (int y = 0; y < img->height; y++) //можно вынести в отдельную
функцию
            free(img->row_pointers[y]);
        free(img->row_pointers);

        fclose(fp);
    }

    void print_png_info(struct Png *img) {
        printf("img Width: %d\n", img->width);
        printf("img Height: %d\n", img->height);
        printf("img Bit Depth: %d\n", img->bit_depth);
        printf("img Number of passes: %d\n", img->number_of_passes);
        printf("img Channels: %d\n", img->channels);
        if (img->color_type == PNG_COLOR_TYPE_RGB) {
            printf("img Colour Type: RGB\n");
        } else {
            printf("img Colour Type: RGB_A\n");
        }
    }

    void print_help(){
        printf("Course work for option 17, created by Marina
Konyaeva.\n");
        printf("-input, -i: specifies the name of the input image\n");
        printf("-output, -o: specifies the name of the output image.\n");
        printf("-info: prints image information \n");
        printf("-help, -h: displays help.\n");
    }

    void free_image_data(struct Png *img) {
        for (int y = 0; y < img->height; y++) {
            free(img->row_pointers[y]);
        }
        free(img->row_pointers);
    }

    void create_default(struct Configs* conf){
        conf->x_y[0] = -1;
        conf->x_y[1] = -1;
        conf->radius = -1;
        conf->thickness = -1;
        conf->fill = 0;
        conf->color = malloc(sizeof(int) * 3);
        conf->fill_color = malloc(sizeof(int) * 3);
        for (int i = 0; i < 3; i++){
            conf->color[i] = 0;
            conf->fill_color[i] = 0;
        }
        conf->component_value = -1;
        conf->number_x = -1;
        conf->number_y = -1;
        conf->side_size = -1;
```

```
}

void free_conf(struct Configs* conf){
    free(conf->color);
    free(conf->fill_color);
}
```

Файл: functions.h

```
#pragma once
#include "library.h"

void ending(struct Text t);
void moreten(struct Text *t);
int cmp(const void* sent1,const void* sent2);
void greenword(struct Text t);
int check(wchar_t* sent,wchar_t* token);
void arrofwords(struct Text *t);
void search_seps(struct Text t);
void freeText1(struct Text t);
void freeText2(struct Text t);
```

Файл: functions.c

```
#include "functions.h"
#include "input.h"


int is_pixel_part_of_img(struct Png *img, int x, int y){
    if (y >= 0 && y < img->height && x >= 0 && x < img->width){
        return 1;
    }
    return 0;
}

void set_pixel(struct Png *img, int x, int  y, int* color_arr){
    if (is_pixel_part_of_img(img, x, y) == 0){
        return;
    }
    img->row_pointers[y][x * img->channels + 0] = color_arr[0];
    img->row_pointers[y][x * img->channels + 1] = color_arr[1];
    img->row_pointers[y][x * img->channels + 2] = color_arr[2];
}

void fill_circle(struct Png *img, int xc, int yc, int r, int
*color_arr) {
    int minX = xc - r;
    int minY = yc - r;
    int maxX = xc + r;
    int maxY = yc + r;
    for (int x = minX; x <= maxX; x++) {
        for (int y = minY; y <= maxY; y++) {
            if ((x - xc)*(x - xc) + (y - yc)*(y - yc) <= r * r) {
                set_pixel(img, x, y, color_arr);
            }
        }
    }
}
```

```
     void draw_thick_circle(struct Png *img, int xc, int yc, int r, int
th, int *color_arr) {
          int newth = th;
          if (r < th){
              newth = r;
          }
          int minX = xc - r - th;
          int minY = yc - r - th;
          int maxX = xc + r + th;
          int maxY = yc + r + th;
          for (int x = minX; x <= maxX; x++) {
              for (int y = minY; y <= maxY; y++) {
                  if ((x - xc)*(x - xc) + (y - yc)*(y - yc) <= (r + th -
0.5)*(r + th - 0.5) && (x - xc)*(x - xc) + (y - yc)*(y - yc) >= (r - newth
- 0.75)*(r - newth - 0.75)) {
                      set_pixel(img, x, y, color_arr);
                  }
              }
          }
     }

     void draw_in_octants(struct Png *img, int xc, int yc, int x, int y,
int *color_arr) {
          set_pixel(img, xc+x, yc+y, color_arr);
          set_pixel(img, xc-x, yc+y, color_arr);
          set_pixel(img, xc+x, yc-y, color_arr);
          set_pixel(img, xc-x, yc-y, color_arr);
          set_pixel(img, xc+y, yc+x, color_arr);
          set_pixel(img, xc-y, yc+x, color_arr);
          set_pixel(img, xc+y, yc-x, color_arr);
          set_pixel(img, xc-y, yc-x, color_arr);
     }

     void draw_contour_circle(struct Png *img, int xc, int yc, int r, int
*color_arr) {
          int x = 0;
          int y =  r;
          int d = 3 - 2 * r;
          draw_in_octants(img, xc, yc, x, y, color_arr);
          while (y >= x) {
              x++;
              if (d > 0) {
                  y--;
                  d = d + 4 * (x - y) + 10;
              } else {
                  d = d + 4 * x + 6;
              }
              draw_in_octants(img, xc, yc, x, y, color_arr);
          }
     }

     void draw_line(struct Png *img, int x1, int y1, int x2, int y2,  int
*color_arr){
          const int deltaX = abs(x2 - x1);
          const int deltaY = abs(y2 - y1);
          const int signX = x1 < x2 ? 1 : -1;
          const int signY = y1 < y2 ? 1 : -1;
```

```
        int error = deltaX - deltaY;

        while (x1 != x2 || y1 != y2) {
            // Draw pixel at current position
            set_pixel(img, x1, y1, color_arr);
            int error2 = error * 2;
            if (error2 > -deltaY) {
                error -= deltaY;
                x1 += signX;
            }
            if (error2 < deltaX) {
                error += deltaX;
                y1 += signY;
            }
        }
        set_pixel(img, x1, y1, color_arr);
    }

    void draw_thick_line(struct Png *img, int x1, int y1, int x2, int y2,
int thick, int *color_arr){
        const int deltaX = abs(x2 - x1);
        const int deltaY = abs(y2 - y1);
        const int signX = x1 < x2 ? 1 : -1;
        const int signY = y1 < y2 ? 1 : -1;
        int error = deltaX - deltaY;


        while (x1 != x2 || y1 != y2) {
            // Draw pixel at current position
            fill_circle(img, x1, y1, thick, color_arr);
            int error2 = error * 2;
            if (error2 > -deltaY) {
                error -= deltaY;
                x1 += signX;
            }
            if (error2 < deltaX) {
                error += deltaX;
                y1 += signY;
            }
        }
    }

    void draw_circle(struct Png* img, int* center, int r, int thick, int*
th_color, int flag_fill, int* fill_color){
        thick = thick/2;
        int newthick = thick;
        int x = center[0];
        int y = center[1];

        if (flag_fill){
            fill_circle(img, x, y, r, fill_color);
        }
        if (r < thick){
            newthick = r;
        }

        draw_contour_circle(img, x, y, r + thick, th_color);
```

```
        draw_contour_circle(img, x, y, r - newthick, th_color);
        draw_thick_circle(img, x, y, r, thick, th_color);

    }

    void switch_filter(struct Png* img, char* comp_name, int comp_value){
        if (comp_value < 0 || comp_value > 255){
            printf("Error in switch_filter: incorrect
component_value.\n");
            exit(40);
        }

        if (strcmp(comp_name, "red") == 0){
            for (int x = 0; x < img->width; x++){
                for (int y = 0; y < img->height; y++){
                    img->row_pointers[y][x * img->channels + 0] =
comp_value;
                }
            }
        }
        else if (strcmp(comp_name, "green") == 0){
            for (int x = 0; x < img->width; x++){
                for (int y = 0; y < img->height; y++){
                    img->row_pointers[y][x * img->channels + 1] =
comp_value;
                }
            }
        }
        else if (strcmp(comp_name, "blue") == 0){
            for (int x = 0; x < img->width; x++){
                for (int y = 0; y < img->height; y++){
                    img->row_pointers[y][x * img->channels + 2] =
comp_value;
                }
            }
        }else{
            printf("Error in switch_filter function: Component name is
incorrect\n");
            exit(40);
        }
    }

    void divide_image(struct Png* img, int N, int M, int thick, int*
color_arr){
        if (N < 0 || M < 0){
            printf("Error in divide_image: number_x or number_y is
incorrect\n");
            exit(41);
        }

        int seg_width = img->width / M;
        int seg_height = img->height / N;

        // Draw vertical lines
        for (int i = 1; i < M; ++i) {
            for (int k = 0; k < thick; k++){
                int x = i * seg_width - thick / 2 + k;
```

```
                draw_line(img, x, 0, x, img->height, color_arr);
            }
        }

        // Draw horizontal lines
        for (int i = 1; i < N; ++i) {
            for (int k = 0; k < thick; k++){
                int y = i * seg_height - thick / 2 + k;
                draw_line(img, 0, y, img->width, y, color_arr);
            }
        }
    }

    void draw_square(struct Png* img, int* center, int side, int thick,
int* color_arr, int fill, int* fill_color){
        thick = thick/2 ;
        int x = center[0];
        int y = center[1];

        if (fill){
            for (int i = 0; i < img->width; i++){
                for (int j = 0; j < img->height; j++){
                    if (i > x && i < x + side + 1 && j > y && j < y +
side + 1){
                        set_pixel(img, i, j, fill_color);
                    }
                }
            }
        }

        draw_thick_line(img, x, y, x + side, y, thick, color_arr);
        draw_thick_line(img, x, y + side, x + side, y + side, thick,
color_arr);
        draw_thick_line(img, x, y, x, y + side, thick, color_arr);
        draw_thick_line(img, x + side, y, x + side, y + side, thick,
color_arr);


        draw_thick_line(img, x, y, x + side, y + side, thick, color_arr);
        draw_thick_line(img, x + side, y, x, y + side, thick, color_arr);

    }
```

Файл: checker.h

```
#pragma once
#include "library.h"

void coords_checker(char* coords);
void distance_checker(char * distance);
int* parse_color(char *color);
void color_checker(char *color);
void names_checker(char* inputname, char* outputname);
void no_argschecker(char* arg,char *name);
```

Файл: checker.c

```
#include "checker.h"

void coords_checker(char* coords){
    regex_t regex;
    int reti = regcomp(&regex, "^\\-?[0-9]+\\.\\-?[0-9]+$",
REG_EXTENDED);
    if (reti) {
        printf("Error in coords_checker function: Could not compile
regex.\n");
        exit(41);
    }
    reti = regexec(&regex, coords, 0, NULL, 0);
    if (reti) {
        printf("Error in coords_checker function: Coords are not
correct.\n");
        exit(41);
    }

}

void distance_checker(char * distance){
    regex_t regex;
    int reti = regcomp(&regex, "^\\-?[0-9]+$", REG_EXTENDED);
    if (reti) {
        printf("Error in distance_checker function: Could not compile
regex.\n");
        exit(41);
    };
    reti = regexec(&regex, distance, 0, NULL, 0);
    if (reti) {
        printf("Error in distance_checker function: distance is not
correct\n");
        exit(41);
    }
    int digit = atoi(distance);
    if (digit < 1){
        printf("Error in distance_checker: digit is incorrect\n");
        exit(41);
    }
}

int *parse_color(char *color) {
    int *result = malloc(3 * sizeof(int));
    char *copy_color = malloc(sizeof(char) * 100);
    strcpy(copy_color, color);
    char *token = strtok(copy_color, ".");
    for(int i = 0; i < 3; i++){
        result[i] = atoi(token);
        token = strtok(NULL, ".");
    }
    free(copy_color);
    return result;
}

void color_checker(char *color) {
    regex_t regex;
```

```
        int reti = regcomp(&regex, "^[0-9]+\\.[0-9]+\\.[0-9]+$",
REG_EXTENDED);
        if (reti) {
            printf("Error in color_checker function: color is not
correct\n");
            exit(41);
        }

        reti = regexec(&regex, color, 0, NULL, 0);
        if (reti){
            printf("The entered color is incorrect.\n");
            exit(41);
        }

        int *color_rgb = parse_color(color);
        if (color_rgb[0] > 255 || color_rgb[1] > 255 || color_rgb[2] >
255 || color_rgb[0] < 0 || color_rgb[1] < 0 ||color_rgb[2] < 0) {
            free(color_rgb);
            printf("The entered color is incorrect.\n");
            exit(41);
        }
        free(color_rgb);
    }

    void names_checker(char* inputname, char* outputname){
        if (strcmp(inputname, outputname)==0){
            printf("Error in names_checker: Input and output files are
the same\n");
            exit(41);
        }
    }

    void no_argschecker(char* arg,char *name){
        if(arg != NULL){
            if(!strstr(arg,"--")){
                printf("%s doesn't have arguments\n", name);
                exit(42);
            }
        }
    }
```

Файл: main.c

```
    #include "input.h"
    #include "functions.h"
    #include "checker.h"

    int main(int argc, char **argv) {
        printf("Course work for option 5.17, created by Marina
Konyaeva.\n");
        if (argc <= 1) {
            print_help();
            return 0;
        }

        struct Png img;
        struct Configs conf;
        create_default(&conf);
```

```
        int opt;
        int long_opt_index = 0;
        int key = -1;
        int write_fl = 0;
        char *optstring = "i:o:h";
        struct option long_options[] = {
                {"circle", no_argument, NULL, 'c'},
                {"rgbfilter", no_argument, NULL, 'r'},
                {"split", no_argument, NULL, 's'},
                {"squared_lines", no_argument, NULL, 'l'},

                {"help", no_argument, NULL, 'h'},
                {"info", required_argument, NULL, 'f'},
                {"input", required_argument, NULL, 'i'},
                {"output", required_argument, NULL, 'o'},

                {"center", required_argument, NULL, 'A'},
                {"radius", required_argument, NULL, 'B'},
                {"thickness", required_argument, NULL, 'Z'},
                {"color", required_argument, NULL, 'D'},
                {"fill", no_argument, NULL, 'E'},
                {"fill_color", required_argument, NULL, 'Q'},

                {"component_name", required_argument, NULL, 'W'},
                {"component_value", required_argument, NULL, 'Y'},

                {"number_x", required_argument, NULL, 'R'},
                {"number_y", required_argument, NULL, 'T'},

                {"left_up", required_argument, NULL, 'A'},
                {"side_size", required_argument, NULL, 'P'},
                {NULL, 0, NULL, 0}
        };
        opt   =   getopt_long(argc,   argv,   optstring,   long_options,
&long_opt_index);

        char *pch;
        int input_flag = 0;
        int output_flag = 0;
        char input_file[255];
        char output_file[255];

        int circle_param_c = 0;
        int filter_param_c = 0;
        int split_param_c = 0;
        int square_param_c = 0;

        while (opt != -1) {
            switch (opt) {
                case 'h':
                    no_argschecker(argv[optind],"--help");
                    print_help();
                    return 0;
                case 'c':
                    no_argschecker(argv[optind],"--circle");
                    key = 'c';
                    break;
```

```
case 'r':
    no_argschecker(argv[optind],"--rgbfilter");
    key = 'r';
    break;
case 's':
    no_argschecker(argv[optind],"--split");
    key = 's';
    break;
case 'l':
    no_argschecker(argv[optind],"--squared_lines");
    key = 'l';
    break;
case 'i':
    input_flag = 1;
    strcpy(input_file, optarg);
    break;
case 'o':
    output_flag = 1;
    strcpy(output_file, optarg);
    break;
case 'f':
    read_png_file(optarg, &img);
    print_png_info(&img);
    free_image_data(&img);
    return 0;
    break;
case 'A':
    coords_checker(optarg);

    pch = strtok (optarg,".");
    conf.x_y[0] = atoi(pch);
    pch = strtok (NULL, " ");
    conf.x_y[1] = atoi(pch);

    circle_param_c += 1;
    square_param_c += 1;
    break;
case 'B':
    distance_checker(optarg);
    conf.radius = atoi(optarg);
    circle_param_c += 1;
    break;
case 'Z':
    distance_checker(optarg);
    conf.thickness = atoi(optarg);
    circle_param_c += 1;
    split_param_c += 1;
    square_param_c += 1;
    break;
case 'D':
    color_checker(optarg);
    conf.color = parse_color(optarg);
    circle_param_c += 1;
    split_param_c += 1;
    square_param_c += 1;
    break;
case 'E':
```

```
                    conf.fill = 1;
                    circle_param_c += 1;
                    square_param_c += 1;
                    break;
                case 'Q':
                    color_checker(optarg);
                    conf.fill_color = parse_color(optarg);
                    circle_param_c += 1;
                    square_param_c += 1;
                    break;
                case 'W':
                    conf.component_name = optarg;
                    filter_param_c += 1;
                    break;
                case 'Y':
                    conf.component_value = atoi(optarg);
                    filter_param_c += 1;
                    break;
                case 'R':
                    distance_checker(optarg);
                    conf.number_x = atoi(optarg);
                    split_param_c += 1;
                    break;
                case 'T':
                    distance_checker(optarg);
                    conf.number_y = atoi(optarg);
                    split_param_c += 1;
                    break;
                case 'P':
                    distance_checker(optarg);
                    conf.side_size = atoi(optarg);
                    square_param_c += 1;
                    break;
                case '?':
                    printf("Arguments entered incorrectly.\n");
                    return 0;
                default:
                    break;
            };
            opt = getopt_long(argc, argv, optstring, long_options,
&long_opt_index);
        }

        if (!input_flag){
            input_flag = 1;
            strcpy(input_file, argv[argc - 1]);
        }
        if (!output_flag){
            output_flag = 1;
            strcpy(output_file, "out.png");
        }

        names_checker(input_file, output_file);
        read_png_file(input_file, &img);

        switch (key) {
            case 'c':
```

```
                if (circle_param_c < 4) {
                    printf("Not enough arguments have been entered to
draw a circle.\n");
                    return 0;
                }
                draw_circle(&img, conf.x_y, conf.radius, conf.thickness,
conf.color, conf.fill, conf.fill_color);
                write_png_file(output_file, &img);
                break;
            case 'r':
                if (filter_param_c < 2){
                    printf("Not enough arguments have been entered to
switch filter.\n");
                    return 0;
                }
                switch_filter(&img,                conf.component_name,
conf.component_value);
                write_png_file(output_file, &img);
                break;
            case 's':
                if (split_param_c < 4){
                    printf("Not enough arguments have been entered to
split image.\n");
                    return 0;
                }
                divide_image(&img,    conf.number_x,    conf.number_y,
conf.thickness, conf.color);
                write_png_file(output_file, &img);
                break;
            case 'l':
                if (square_param_c < 4){
                    printf("Not enough arguments have been entered to
draw a square.\n");
                    return 0;
                }
                draw_square(&img,       conf.x_y,       conf.side_size,
conf.thickness, conf.color, conf.fill, conf.fill_color);
                write_png_file(output_file, &img);
                break;
            default:
                printf("Arguments entered incorrectly.\n");
                return 0;
        };

        free_conf(&conf);
        return 0;
    }
```

Файл: Makefile

```
all: main.o input.o functions.o
     gcc *.o -lpng -o cw -std=c99
main.o: main.c input.o functions.o checker.o *.h
     gcc -c -lpng main.c -std=c99
input.o: input.c
     gcc -c -lpng input.c -std=c99
functions.o: functions.c
     gcc -c -lpng functions.c -std=c99
```

```
checker.o: checker.c
	gcc -c -lpng checker.c -std=c99
clean:
	rm *.o cw
```