

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

Якимова Ю.А.

Преподаватель

Глазунов С.А.

Санкт-Петербург

2024

ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

Студентка Якимова Ю.А.

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. Копирование заданной области. Флаг для выполнения данной операции: `--copy`. Функционал определяется:
 - Координатами левого верхнего угла области-источника. Флаг `--left_up`, значение задаётся в формате `'left.up'`, где `left` – координата по `x`, `up` – координата по `y`

- Координатами правого нижнего угла области-источника. Флаг `--right_down`, значение задаётся в формате `right.down`, где `right` – координата по x, `down` – координата по y
 - Координатами левого верхнего угла области-назначения. Флаг `--dest_left_up`, значение задаётся в формате `left.up`, где `left` – координата по x, `up` – координата по y
2. Заменяет все пиксели одного заданного цвета на другой цвет. Флаг для выполнения данной операции: `--color_replace`. Функционал определяется:
- Цвет, который требуется заменить. 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)
 - Цвет на который требуется заменить. Флаг `--new_color` (работает аналогично флагу `--old_color`)
3. Сделать рамку в виде узора. Флаг для выполнения данной операции: `--ornament`. Рамка определяется:
- Узором. Флаг `--pattern`. Обязательные значения: *rectangle* и *circle*, *semicircles*. Также можно добавить свои узоры (красивый узор можно получить используя фракталы)
 - 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)
 - Шириной. Флаг `--thickness`. На вход принимает число больше 0
 - Количеством. Флаг `--count`. На вход принимает число больше 0
 - При необходимости можно добавить дополнительные флаги для необозначенных узоров
4. Поиск всех залитых прямоугольников заданного цвета. Флаг для выполнения данной операции: `--filled_rects`. Требуется найти все прямоугольники заданного цвета и обвести их линией. Функционал

определяется:

- Цветом искомых прямоугольников. Флаг `--color` (цвет задаётся строкой `rrr.ggg.bbb`, где `rrr/ggg/bbb` – числа, задающие цветовую компоненту. пример `--color 255.0.0` задаёт красный цвет)
- Цветом линии для обводки. Флаг `--border_color` (работает аналогично флагу `--color`)
- Толщиной линий. 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

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: 27.05.2023

Дата защиты реферата: 29.05.2023

Студентка

Якимова Ю.А.

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	7
1. Подключаемые библиотеки, структуры	8
2. Функции	9
2.1 Функции чтения и записи <i>PNG</i> -файла	9
2.2 Дополнительные, вспомогательные функции	9
2.3 Основные функции	11
2.4 Функция <i>main</i>	12
2.5 <i>Makefile</i>	13
Заключение	14
Список использованных источников	15
Приложение А. Результаты тестирования	16
Приложение Б. Исходный код программы	19

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 копирования заданной области;
- разработать функцию замены пикселей заданного цвета;
- разработать функцию, которая создает рамки в виде узора;
- разработать функцию поиска всех залитых прямоугольников заданного цвета;
- написать *Makefile* для удобной сборки проекта;
- протестировать разработанную программу.

1. ПОДКЛЮЧАЕМЫЕ БИБЛИОТЕКИ, СТРУКТУРЫ

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

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

Определено шесть структур:

- 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 *Color*, содержащая все аргументы, требуемые для замены цвета, а также их флаги
- Структура *Copy*, содержащая все аргументы, требуемые для копирования заданной области, а также их флаги
- Структура *Ornament*, содержащая все аргументы, требуемые для создания рамок в виде узора, а также флаги аргументов
- Структура *Filled_Rect*, содержащая все аргументы, требуемые для того, чтобы найти и обвести прямоугольники заданного цвета, а также флаги аргументов
- Структура *Coordinates*, содержащая координаты прямоугольника

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

Функция *fill_rect()* принимает на вход указатель *img* на структуру *Png*, координаты левого верхнего и правого нижнего углов прямоугольника, массив из трех целых чисел, представляющий цвет в формате RGB. Функция заполняет прямоугольник в изображении *PNG* указанным цветом.

Функция *create_border()* создает рамку вокруг прямоугольника в изображении *PNG* указанным цветом и толщиной. Если толщина рамки больше или равна половине ширины или высоты прямоугольника, функция заполняет весь прямоугольник указанным цветом. В противном случае функция заполняет рамку вокруг прямоугольника указанной толщиной. Она заполняет верхнюю, нижнюю, левую и правую стороны прямоугольника с помощью функции *fill_rect()*.

Функция *check_str_format()* проверяет строку и извлекает из нее значения в зависимости от количества токенов (разделенных точками), которое может быть равно 2 (для координат формата *x.y*) или 3 (для цветов формата *rrr.ggg.bbb*). Функция также проверяет корректность значений и возвращает массив целых чисел, содержащий извлеченные значения.

Функция *create_semicircle()* рисует полукруг на основе полученных координат его центра, радиуса, а также толщины и цвета линии. Функция вычисляет минимальные и максимальные координаты *x* и *y* для полукруга с учетом его толщины (*r2*). Далее функция выполняет двойной цикл по всем пикселям в пределах полукруга. Для каждого пикселя она вычисляет расстояние от точки (*x*, *y*) до центра (*x0*, *y0*). Если расстояние находится между радиусами *r1* и *r2*, то пиксель окрашивается в цвет *rgb*.

Функция *compare_colors()* сравнивает два цвета формата RGB.

Функция *free_border_data()* создает двумерный массив и для каждого пикселя PNG изображения записывает количество окружающих его пикселей с цветом, отличным от заданного.

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

2.3. Основные функции

Функция *copy_img()* принимает на вход указатель *img* на структуру *Png*, строки, содержащие координаты левого верхнего и правого нижнего угла прямоугольной области в исходном изображении в формате "x,y", а также строку, содержащую координаты левого верхнего угла области, в которую будут скопированы пиксели. Функция преобразует входные строки в координаты, используя вспомогательную функцию *check_str_format*. Она проверяет, что координаты образуют действительный прямоугольник, вычисляет его ширину и высоту. Функция выделяет память для временного массива *rect*, который будет хранить пиксели из прямоугольной области, после чего пиксели из прямоугольной области копируются в массив *rect*. Функция вычисляет ширину и высоту области, в которую будут копироваться пиксели, с учетом возможных ограничений в виде границы изображения. Пиксели из массива *rect* копируются в целевую область.

Функция *replace_color()* принимает на вход указатель *img* на структуру *Png*, а также две строки, содержащие цвета. Функция заменяет цвет пикселей в изображении в формате PNG с одного цвета (*old_color*) на другой цвет (*new_color*).

Функция *create_rect_ornam()* создает орнамент в виде прямоугольных рамок на изображении путем рисования ряда прямоугольников с заданным цветом, толщиной и количеством (рамка состоит из 4 прямоугольников).

Функция *create_circle_ornam()* создает круговое украшение на изображении, заполняя область за пределами круга заданным цветом. Функция вычисляет радиус круга как меньшую из половины ширины и половины высоты изображения. Центр круга устанавливается как середина изображения. Для каждого пикселя в изображении функция вычисляет расстояние от пикселя до центра круга. Если пиксель находится за пределами круга (расстояние больше радиуса), функция заполняет пиксель заданным цветом.

Функция *create_semicir_ornam()* создает орнамент в виде полукругов по границам изображения. Функция вычисляет радиусы полукругов для горизонтальных и вертикальных сторон. Далее для каждой стороны определяет центр первого полукруга при нечетном количестве полукругов (центр находится в середине стороны) или центры первых полукругов для нечетного количества (центры смещены от середины стороны на сумму радиуса и половины толщины линии). Последующие полукруги рисуются по принципу смещения от центральных в сторону границ изображения.

Функция *trace_rects()* находит и обводит прямоугольники заданного цвета.

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 введен

некорректный параметр, неверный флаг или недостаточное количество аргументов, программа выводит сообщение об ошибке и завершает выполнение. Правильные параметры сохраняются в одну из структур. 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 расширением *.h*, которые содержат нужные библиотеки и сигнатуры функций. Файлы с содержанием исходного кода: *functions.c*, *input.c*. У каждого из них есть одноименный заголовочный файл. Файл *main.c* содержит главную функцию *main*, которая управляет выполнением программы, вызывая другие её функции. Заголовочные файлы *library.h* и *structures.h* содержат 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: изменение цвета, копирование области изображения, создание орнамента, нахождение и обводка прямоугольников заданного цвета. 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%20Брезенхэма%20\(англ.,разработан%20Джеком%20Элтоном%20Брезенхэм%20\(англ.\(дата обращения 17.05.2023\).](https://ru.wikipedia.org/wiki/Алгоритм_Брезенхэма#:~:text=Алгоритм%20Брезенхэма%20(англ.,разработан%20Джеком%20Элтоном%20Брезенхэм%20(англ.(дата%20обращения%2017.05.2023).)
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 С++ // cppstudio. URL: <http://cppstudio.com/> (дата обращения 17.05.2023).

ПРИЛОЖЕНИЕ А

РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ

1. Вывод справки:

```
● peacemaker@DESKTOP-KMB4N4U:/mnt/c/from-ubuntu/local/pr-2024-3344/Yakimova_Yuliya_cw/src$ ./cw --help
Course work for option 5.16, created by Yakimova Yuliya.

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:

    --copy - копирование заданной области (координаты задаются в формате `x.y`)
        --left_up - координаты левого верхнего угла области-источника
        --right_down - координаты правого нижнего угла области-источника
        --dest_left_up - координаты левого верхнего угла области-назначения
    --color_replace - заменяет все пиксели одного заданного цвета на другой цвет
        --old_color - цвет, который требуется заменить (задаётся строкой `rrr.ggg.bbb`)
        --new_color - цвет, на который требуется заменить (задаётся строкой `rrr.ggg.bbb`)
    --ornament - сделать рамку в виде узора
        --pattern - узор (возможные значения: rectangle, circle, semicircles)
        --color - цвет (задаётся строкой `rrr.ggg.bbb`)
        --thickness - ширина
        --count - количество
    --filled_rects - поиск всех залитых прямоугольников заданного цвета
        --color - цвет искомых прямоугольников (задаётся строкой `rrr.ggg.bbb`)
        --border_color - цвет линии для обводки
        --thickness - толщина линии для обводки
```

2. Обработка изображения: копирование области изображения (./cw --copy --left_up 200.100 --right_down 500.250 --dest_left_up 300.300 ornament_image.png):



3. Обработка изображения: замена цвета (./cw --color_replace --old_color 0.0.0 --new_color 255.0.0 ornament_image.png):



4. Обработка изображения: поиск прямоугольников (./cw --filled_rects --color 0.0.0 --border_color 0.255.0 --thickness 5 ornament_image.png):



5. Обработка изображения: рисование орнамента



ПРИЛОЖЕНИЕ Б

ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл: library.h

```
#pragma once

#include <getopt.h>
#include <stdlib.h>
#include <png.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
```

Файл: structures.h

```
#pragma once

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;    //число проходов для полной обработки
изображения
    png_bytep *row_pointers;
};

struct Color{
    char* old_color;
    char* new_color;

    int old_color_f;
    int new_color_f;
};

struct Copy{
    char* left_up;
    char* right_down;
    char* dest_left_up;

    int left_up_f;
    int right_down_f;
    int dest_left_up_f;
};

struct Ornament{
    char* pattern;
    char* color;
    char* thickness;
    char* count;
```

```

    int pattern_f;
    int color_f;
    int thickness_f;
    int count_f;
};

struct Filled_Rect {
    char* color;
    char* border_color;
    char* thickness;

    int color_f;
    int border_color_f;
    int thickness_f;
};

struct Coordinates {
    int x0, y0, x1, y1;
};

```

Файл: input.h

```

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

void read_png_file(char *file_name, struct Png *image);
void write_png_file(char *file_name, struct Png *image);
void free_image_data(struct Png *image);

```

Файл: input.c

```

#include "input.h"

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

    int x, y;
    char header[8]; //8 - максимальная длина, которую можно проверить

    //открыть файл (fopen возвращает NULL, если файл не может быть
открыт)
    FILE *fp = fopen(file_name, "rb"); //rb - открыть двоичный файл
для чтения
    if (!fp){
        printf("Cannot read file: %s\n", file_name);
        exit(1);
    }

    fread(header, 1, 8, fp); //size_t fread(void *buf, size_t size,
size_t count, FILE *stream)
    //проверить, что файл PNG
    if (png_sig_cmp(header, 0, 8)){ //start - второй аргумент,
num_to_check - третий
        printf("%s is not recognized as a PNG\n", file_name);
        exit(1);
    }
}

```

```

//инициализация структуры PNG
image->png_ptr = png_create_read_struct(PNG_LIBPNG_VER_STRING,
NULL, NULL, NULL);
if (!image->png_ptr){
    printf("Error in PNG structure\n");
    exit(1);
}

image->info_ptr = png_create_info_struct(image->png_ptr);
if (!image->info_ptr){
    printf("Error in PNG info-structure\n");
    exit(1);
}

if (setjmp(png_jmpbuf(image->png_ptr))){
    printf("Error during init_io\n");
    exit(1);
}

//заполнение структуры PNG

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

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

image->width          =      png_get_image_width(image->png_ptr,
image->info_ptr);
image->height         =      png_get_image_height(image->png_ptr,
image->info_ptr);
image->color_type      =      png_get_color_type(image->png_ptr,
image->info_ptr);
image->bit_depth       =      png_get_bit_depth(image->png_ptr,
image->info_ptr);

image->number_of_passes
png_set_interlace_handling(image->png_ptr);
image->channels         =      png_get_channels(image->png_ptr,
image->info_ptr);

if (image->color_type == PNG_COLOR_TYPE_GRAY) {
    printf("The program does not support working with the
PNG_COLOR_TYPE_GRAY color type.\n");
    exit(1);
} else if (image->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(1);
} else if (image->color_type == PNG_COLOR_TYPE_PALETTE) {
    printf("The program does not support working with the
PNG_COLOR_TYPE_PALETTE color type.\n");
    exit(1);
}

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

//чтение файла

```

```

        if (setjmp(png_jmpbuf(image->png_ptr))) {
            printf("Error during read_image\n");
            return;
        }

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

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

        fclose(fp);
    }

void write_png_file(char *file_name, struct Png *image) {
    int x,y;

    /* создание файла */
    FILE *fp = fopen(file_name, "wb"); //wb - создать двоичный файл
для записи
    if (!fp) {
        printf("%s could not be opened\n", file_name);
        exit(1);
    }

    /* инициализация структуры */
    image->png_ptr = png_create_write_struct(PNG_LIBPNG_VER_STRING,
NULL, NULL, NULL);

    if (!image->png_ptr) {
        printf("png_create_write_struct failed\n");
        exit(1);
    }

    image->info_ptr = png_create_info_struct(image->png_ptr);
    if (!image->info_ptr) {
        printf("png_create_info_struct failed\n");
        exit(1);
    }

    if (setjmp(png_jmpbuf(image->png_ptr))) {
        printf("Error during init_io\n");
        exit(1);
    }

    png_init_io(image->png_ptr, fp);

    /* запись заголовка */
    if (setjmp(png_jmpbuf(image->png_ptr))) {
        printf("Error during writing header\n");
        exit(1);
    }

```

```

        //set image header information in info_ptr
        png_set_IHDR(image->png_ptr, image->info_ptr, image->width,
image->height,
                        image->bit_depth, image->color_type,
PNG_INTERLACE_NONE,
                        PNG_COMPRESSION_TYPE_BASE, PNG_FILTER_TYPE_BASE);

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

        /* запись байтов */
        if (setjmp(png_jmpbuf(image->png_ptr))) {
            printf("Error during writing bytes\n");
            exit(1);
        }

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

        /* конец записи */
        if (setjmp(png_jmpbuf(image->png_ptr))) {
            printf("Error during the end of writing\n");
            exit(1);
        }

        png_write_end(image->png_ptr, NULL);

        /* очистка */

        free_image_data(image);

        fclose(fp);
    }

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

```

Файл: functions.h

```

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

void no_argschecker(char* arg, char* name);
void print_png_info(struct Png *image);
void fill_rect(struct Png *image, int x1, int y1, int x2, int y2,
int* rgb);
void create_border(struct Png *image, int x1, int y1, int x2, int y2,
int* rgb, int thickness);
int* check_str_format(char* str, int tok_num, struct Png *image);
void replace_color(struct Png *image, char* new_color, char*
old_color);

```



```

void copy_img(struct Png *image, char* left_up, char* right_down,
char* dest_left_up);
void create_rect_ornam(struct Png *image, char* color, char*
thickness_str, char* count_str);
void create_circle_ornam(struct Png *image, char* color);
void create_semicircle (struct Png *image, int x0, int y0, int r1,
int thickness, int* rgb);
void create_semicir_ornam(struct Png *image, char* color, char*
thickness_str, char* count_str);
int compare_colors(png_byte* ptr, int* color);
int** free_border_data(struct Png *image, int* color);
void trace_rects(struct Png *image, char* color_ch, char*
border_color_ch, char* thickness_str);
void print_help();

```

Файл: functions.c

```

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

void no_argschecker(char* arg, char* name){
    if(arg != NULL) {
        if(!strstr(arg, "--")){
            printf("%s doesn't have any arguments\n", name);
            exit(40);
        }
    }
}

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

void fill_rect(struct Png *image, int x1, int y1, int x2, int y2,
int* rgb) {
    if (x1 > x2 || y1 > y2 || x2 > (image->width - 1) || y2 >
(image->height - 1)) return;
    if (x1 < 0 || x2 < 0 || y1 < 0 || y2 < 0) return;

    png_byte channels = image->channels;

    for (int y = y1; y <= y2; y++) {
        png_byte *row = image->row_pointers[y];
        for (int x = x1; x <= x2; x++) {
            png_byte *ptr = &(row[x * channels]);
            ptr[0] = rgb[0];

```

```

        ptr[1] = rgb[1];
        ptr[2] = rgb[2];
        if(channels == 4) ptr[3] = 255;
    }
}

void create_border(struct Png *image, int x1, int y1, int x2, int y2,
int* rgb, int thickness) {
    if (x1 > x2 || y1 > y2 || x2 > (image->width - 1) || y2 >
(image->height - 1)) return;
    if (x1 < 0 || x2 < 0 || y1 < 0 || y2 < 0) return;
    if (thickness < 0) return;

    int width = x2 - x1 + 1;
    int height = y2 - y1 + 1;

    if ((thickness >= (width / 2)) || (thickness >= (height / 2))) {
        fill_rect(image, x1, y1, x2, y2, rgb);
        return;
    }

    fill_rect(image, x1, y1, x2, y1 + thickness - 1, rgb);
    fill_rect(image, x1, y2 - thickness + 1, x2, y2, rgb);
    fill_rect(image, x1, y1, x1 + thickness - 1, y2, rgb);
    fill_rect(image, x2 - thickness + 1, y1, x2, y2, rgb);
}

//проверяет формат x.y или rrr.ggg.bbb
int* check_str_format(char* str, int tok_num, struct Png *image) {
    char** char_res = malloc(sizeof(char*) * tok_num);
    int* res = malloc(sizeof(int) * tok_num);

    for (int j = 0; j < tok_num; j++) {
        char_res[j] = malloc(sizeof(char) * strlen(str) + 10);
    }

    char sep[1] = ".";
    char* istr;
    istr = strtok(str, sep);

    int i=0;
    while (istr != NULL)
    {
        strcpy(char_res[i], istr);
        if((tok_num == 3) && (strcmp(char_res[i], "0") != 0) &&
(!atoi(char_res[i]))) {
            printf("The entered color name must be `rrr.ggg.bbb`
format.");
            exit(40);
        }

        if((tok_num == 2) && (strcmp(char_res[i], "0") != 0) &&
(!atoi(char_res[i]))) {
            printf("The entered coordinates must be `x.y` format.");

```

```

        exit(40);
    }

    istr = strtok (NULL,sep);
    i++;
    if(i == tok_num) break;
}

if ((i < tok_num) && (tok_num == 3)) {
    printf("The entered color name must be `rrr.ggg.bbb`
format.");
    exit(40);
}

if ((i < tok_num) && (tok_num == 2)) {
    printf("The entered coordinates must be `x.y` format.");
    exit(40);
}

for (int j=0; j<i; j++) {
    if ((tok_num == 3) && (atoi(char_res[j]) < 0 ||
atoi(char_res[j]) > 255)) {
        printf("Error in check_str_format: incorrect color
value.\n");
        exit(40);
    }

    if((tok_num == 2) && (atoi(char_res[j]) < 0 ||
atoi(char_res[0]) > image->width || atoi(char_res[1]) > image->height)) {
        printf("Error in check_str_format: incorrect coordinates
value.\n");
        exit(40);
    }

    res[j] = atoi(char_res[j]);
}

for (int k=0; k < tok_num; k++) {
    free(char_res[k]);
}
free(char_res);

return res;
}

void replace_color(struct Png *image, char* new_color, char*
old_color) {
    int* new_rgb = malloc(sizeof(int) * 3);
    new_rgb = check_str_format(new_color, 3, image);
    int* old_rgb = malloc(sizeof(int) * 3);
    old_rgb = check_str_format(old_color, 3, image);

    int f = 0;
    for (int x = 0; x < image->width; x++) {
        for (int y = 0; y < image->height; y++) {

```

```

        png_byte *ptr = &(image->row_pointers[y][x *
image->channels]);
        for (int i = 0; i < image->channels; i++) {
            if (ptr[i] == old_rgb[i]) f++;
            else break;
        }
        if (f >= 3) {
            for (int j = 0; j < image->channels; j++) {
                ptr[j] = new_rgb[j];
            }
        }
        f = 0;
    }
}

void copy_img(struct Png *image, char* left_up, char* right_down,
char* dest_left_up) {
    int* lu_coords = malloc(sizeof(int) * 2);
    lu_coords = check_str_format(left_up, 2, image);
    int* rd_coords = malloc(sizeof(int) * 2);
    rd_coords = check_str_format(right_down, 2, image);
    int* dlu_coords = malloc(sizeof(int) * 2);
    dlu_coords = check_str_format(dest_left_up, 2, image);

    int x1 = lu_coords[0];
    int y1 = lu_coords[1];
    int x2 = rd_coords[0];
    int y2 = rd_coords[1];
    int x3 = dlu_coords[0];
    int y3 = dlu_coords[1];

    if (x1 > x2 || y1 > y2) {
        printf("The entered coordinates do not form a rectangle.\n");
        exit(40);
    }

    int rect_width = x2 - x1 + 1;
    int rect_height = y2 - y1 + 1;
    png_byte channels = image->channels;

    png_bytep* rect = malloc(sizeof(png_byte*) * rect_height + 10);
    for (int i=0; i < rect_height; i++) {
        rect[i] = malloc(sizeof(png_byte) * rect_width * channels +
10);
        for (int j = 0; j < rect_width * channels; j++) {
            rect[i][j] = image->row_pointers[i + y1][j + x1 *
channels];
        }
    }

    int rect_w_fin;
    if (rect_width > image->width - x3) rect_w_fin = image->width -
x3;
    else rect_w_fin = rect_width;
    int rect_h_fin;

```

```

        if (rect_height > image->height - y3) rect_h_fin = image->height
- y3;
        else rect_h_fin = rect_height;

        for (int x = 0; x < rect_w_fin * channels; x++) {
            for (int y = 0; y < rect_h_fin; y++) {
                image->row_pointers[y + y3][x + x3 * channels] =
rect[y][x];
            }
        }
    }

    void create_rect_ornam(struct Png *image, char* color, char*
thickness_str, char* count_str) {
        int* rgb = malloc(sizeof(int) * 3);
        rgb = check_str_format(color, 3, image);

        int thickness = atoi(thickness_str);
        if (thickness <= 0) {
            printf("--thickness argument must be > 0.\n");
            exit(40);
        }
        int count = atoi(count_str);
        if (count <= 0) {
            printf("--count argument must be > 0.\n");
            exit(40);
        }

        int width = image->width;
        int height = image->height;

        if ((thickness >= (width / 2)) || (thickness >= (height / 2))) {
            fill_rect(image, 0, 0, width - 1, height - 1, rgb);
            return;
        }
        while (((count * 2 - 1) * thickness > (width / 2)) || ((count *
2 - 1) * thickness > (height / 2))) {
            count--;
        }

        int x1 = 0;
        int y1 = 0;
        int x2 = width - 1;
        int y2 = height - 1;
        while (count != 0) {
            create_border(image, x1, y1, x2, y2, rgb, thickness);
            x1 = x1 + thickness * 2;
            y1 = y1 + thickness * 2;
            x2 = x2 - thickness * 2;
            y2 = y2 - thickness * 2;
            count--;
        }
    }

    void create_circle_ornam(struct Png *image, char* color) {
        int* rgb = malloc(sizeof(int) * 3);

```

```

    rgb = check_str_format(color, 3, image);

    int rad;
    int x0 = (int)(image->width / 2) + 1;
    int y0 = (int)(image->height / 2) + 1;
    if (image->width < image->height) {
        rad = (int)(image->width / 2);
    } else {
        rad = (int)(image->height / 2);
    }

    for (int y = 0; y < image->height; y++) {
        png_byte *row = image->row_pointers[y];
        for (int x = 0; x < image->width; x++) {
            if ((pow(x - x0, 2) + pow(y - y0, 2)) > pow(rad, 2)) {
                png_byte *ptr = &(row[x * image->channels]);
                ptr[0] = rgb[0];
                ptr[1] = rgb[1];
                ptr[2] = rgb[2];
                if (image->channels == 4) ptr[3] = 255;
            }
        }
    }
}

void create_semicircle (struct Png *image, int x0, int y0, int r1,
int thickness, int* rgb) {
    if (thickness <= 0 || x0 < 0 || y0 < 0 || x0 >= image->width ||
y0 >= image->height) return;

    int x_min, x_max, y_min, y_max;
    int r2 = r1 + thickness;

    if (x0 - r2 + 1 >= 0) x_min = x0 - r2 + 1;
    else x_min = 0;
    if (x0 + r2 - 1 < image->width) x_max = x0 + r2 - 1;
    else x_max = image->width - 1;
    if (y0 - r2 + 1 >= 0) y_min = y0 - r2 + 1;
    else y_min = 0;
    if (y0 + r2 - 1 < image->height) y_max = y0 + r2 - 1;
    else y_max = image->height - 1;

    for (int y = y_min; y <= y_max; y++) {
        png_byte *row = image->row_pointers[y];
        for (int x = x_min; x <= x_max; x++) {
            if (((pow(x - x0, 2) + pow(y - y0, 2)) > pow(r1 + 2, 2))
&& ((pow(x - x0, 2) + pow(y - y0, 2)) <= pow(r2 + 2, 2))) {
                png_byte *ptr = &(row[x * image->channels]);
                ptr[0] = rgb[0];
                ptr[1] = rgb[1];
                ptr[2] = rgb[2];
                if (image->channels == 4) ptr[3] = 255;
            }
        }
    }
}

```

```

void create_semicir_ornam(struct Png *image, char* color, char*
thickness_str, char* count_str) {
    int* rgb = malloc(sizeof(int) * 3);
    rgb = check_str_format(color, 3, image);

    int thickness = atoi(thickness_str);
    if (thickness <= 0) {
        printf("--thickness argument must be > 0.\n");
        exit(40);
    }
    int count = atoi(count_str);
    if (count <= 0) {
        printf("--count argument must be > 0.\n");
        exit(40);
    }
    if ((image->width < (count+1) * thickness) || (image->height <
(count+1) * thickness)) {
        printf("Some arguments are incorrect.\n");
        exit(40);
    }
    thickness++;

    int d_x, r_x, d_y, r_y;
    if ((image->width - (count+1) * thickness) % count != 0) {
        d_x = (int)((image->width - (count+1) * thickness) / count)
+ 1;
    } else d_x = (int)((image->width - (count+1) * thickness) /
count);
    r_x = (int)(d_x / 2);
    if(d_x % 2 == 1) r_x++;

    if ((image->height - (count+1) * thickness) % count != 0) {
        d_y = (int)((image->height - (count+1) * thickness) / count)
+ 1;
    } else d_y = (int)((image->height - (count+1) * thickness) /
count);
    r_y = (int)(d_y / 2);
    if(d_y % 2 == 1) r_y++;

    int x_center = (int)(image->width / 2);
    int y_center = 0;
    if(count % 2 == 1) {
        //сверху
        create_semicircle(image, x_center, y_center, r_x, thickness,
rgb);
        for (int i = 1; i <= (count-1) / 2; i++) {
            create_semicircle(image, x_center + i * (2 * r_x +
thickness), y_center, r_x, thickness, rgb);
            create_semicircle(image, x_center - i * (2 * r_x +
thickness), y_center, r_x, thickness, rgb);
        }

        //снизу
        y_center = image->height - 1;

```

```

        create_semicircle(image, x_center, y_center, r_x, thickness,
rgb);
        for (int i = 1; i <= (count-1) / 2; i++) {
            create_semicircle(image, x_center + i * (2 * r_x +
thickness), y_center, r_x, thickness, rgb);
            create_semicircle(image, x_center - i * (2 * r_x +
thickness), y_center, r_x, thickness, rgb);
        }

        //слева
        y_center = (int)(image->height / 2);
        x_center = 0;
        create_semicircle(image, x_center, y_center, r_y, thickness,
rgb);
        for (int i = 1; i <= (count-1) / 2; i++) {
            create_semicircle(image, x_center, y_center + i * (2 *
r_y + thickness), r_y, thickness, rgb);
            create_semicircle(image, x_center, y_center - i * (2 *
r_y + thickness), r_y, thickness, rgb);
        }

        //справа
        x_center = image->width - 1;
        create_semicircle(image, x_center, y_center, r_y, thickness,
rgb);
        for (int i = 1; i <= (count-1) / 2; i++) {
            create_semicircle(image, x_center, y_center + i * (2 *
r_y + thickness), r_y, thickness, rgb);
            create_semicircle(image, x_center, y_center - i * (2 *
r_y + thickness), r_y, thickness, rgb);
        }
    }
    if (count % 2 == 0) {
        //сверху
        int del_x = (int)(r_x + thickness / 2);
        if (thickness % 2 != 0) del_x++;
        for (int i = 0; i <= count / 2; i++) {
            create_semicircle(image, x_center + del_x + i * (2 * r_x
+ thickness), y_center, r_x, thickness, rgb);
            create_semicircle(image, x_center - del_x - i * (2 * r_x
+ thickness), y_center, r_x, thickness, rgb);
        }

        //снизу
        y_center = image->height - 1;
        for (int i = 0; i <= count / 2; i++) {
            create_semicircle(image, x_center + del_x + i * (2 * r_x
+ thickness), y_center, r_x, thickness, rgb);
            create_semicircle(image, x_center - del_x - i * (2 * r_x
+ thickness), y_center, r_x, thickness, rgb);
        }

        //слева
        int del_y = (int)(r_y + thickness / 2);
        if (thickness % 2 != 0) del_y++;
        y_center = (int)(image->height / 2);
        x_center = 0;

```



```

        for (int i = 0; i <= count / 2; i++) {
            create_semicircle(image, x_center, y_center + del_y + i
* (2 * r_y + thickness), r_y, thickness, rgb);
            create_semicircle(image, x_center, y_center - del_y - i
* (2 * r_y + thickness), r_y, thickness, rgb);
        }

        //справа
        x_center = image->width - 1;
        for (int i = 0; i <= count / 2; i++) {
            create_semicircle(image, x_center, y_center + del_y + i
* (2 * r_y + thickness), r_y, thickness, rgb);
            create_semicircle(image, x_center, y_center - del_y - i
* (2 * r_y + thickness), r_y, thickness, rgb);
        }
    }

    int compare_colors(png_byte* ptr, int* color) {
        if ((ptr[0] == color[0]) && (ptr[1] == color[1]) && (ptr[2] ==
color[2])) return 1;
        return 0;
    }

    int** free_border_data(struct Png *image, int* color) {
        int** free_border_data_array = (int**) malloc(sizeof(int*) *
image->height + 10);
        for (int y = 0; y < image->height; y++) {
            free_border_data_array[y] = (int*) malloc(sizeof(int) *
image->width + 10);
        }

        png_bytep* row_pointers = image->row_pointers;
        int channels = image->channels;

        for (int y = 0; y < image->height; y++)
        {
            png_byte *row = row_pointers[y];
            for (int x = 0; x < image->width; x++)
            {
                png_byte *ptr = &(row[x * channels]);
                if (compare_colors(ptr, color) == 1)
                {
                    if ((y + 1 == image->height || y - 1 < 0) && (x
+ 1 == image->width || x - 1 < 0))
                        free_border_data_array[y][x] = 5;
                    else if (y + 1 == image->height || y - 1 < 0 ||
x + 1 == image->width || x - 1 < 0)
                        free_border_data_array[y][x] = 3;
                    if (y + 1 < image->height &&
compare_colors(&(row_pointers[y + 1][x * channels]), color) == 0)
                        free_border_data_array[y][x] += 1;
                    if (y - 1 >= 0 &&
compare_colors(&(row_pointers[y - 1][x * channels]), color) == 0)
                        free_border_data_array[y][x] += 1;
                    if (x + 1 < image->width &&
compare_colors(&(row[(x + 1) * channels]), color) == 0)

```

```

        free_border_data_array[y][x] += 1;
        if (x - 1 >= 0 && compare_colors(&(row[(x - 1)
* channels])), color) == 0)
            free_border_data_array[y][x] += 1;
            if (y + 1 < image->height && x + 1 <
image->width && compare_colors(&(row_pointers[y + 1][(x+1) * channels])),
color) == 0)
                free_border_data_array[y][x] += 1;
                if (y + 1 < image->height && x - 1 >= 0 &&
compare_colors(&(row_pointers[y + 1][(x-1) * channels])), color) == 0)
                    free_border_data_array[y][x] += 1;
                    if (y - 1 >= 0 && x + 1 < image->width &&
compare_colors(&(row_pointers[y - 1][(x+1) * channels])), color) == 0)
                        free_border_data_array[y][x] += 1;
                        if (y - 1 >= 0 && x - 1 >= 0 &&
compare_colors(&(row_pointers[y - 1][(x-1) * channels])), color) == 0)
                            free_border_data_array[y][x] += 1;
                    }
                }
            }
        }
    }
    return free_border_data_array;
}

void trace_rects(struct Png *image, char* color_ch, char*
border_color_ch, char* thickness_str) {
    int* color = malloc(sizeof(int) * 3);
    color = check_str_format(color_ch, 3, image);
    int* border_color = malloc(sizeof(int) * 3);
    border_color = check_str_format(border_color_ch, 3, image);

    int thickness = atoi(thickness_str);
    if (thickness <= 0) {
        printf("--thickness argument must be > 0.\n");
        exit(40);
    }

    int** free_border_data_array = free_border_data(image, color);

    struct Coordinates** coords = malloc(sizeof(struct Coordinates*)
* image->width * image->height);
    int coords_count = 0;
    png_bytep* row_pointers = image->row_pointers;
    int channels = image->channels;

    for (int y = 0; y < image->height; y++) {
        png_byte *row = row_pointers[y];
        for (int x = 0; x < image->width; x++) {
            png_byte *ptr = &(row[x * channels]);
            if (compare_colors(ptr, color) == 0) continue;
            struct Coordinates *c = malloc(sizeof(struct
Coordinates));
            if (free_border_data_array[y][x] == 8) {
                c->x0 = x;
                c->x1 = x;
                c->y0 = y;
                c->y1 = y;
                coords[coords_count++] = c;
            }
        }
    }
}

```

```

        continue;
    }
    if (free_border_data_array[y][x] == 7) {
        int flag = 0;
        for (int x1 = x + 1; x1 < image->width; x1++)
        {
            if (free_border_data_array[y][x1] != 6) {
                if
                (free_border_data_array[y][x1] != 7) break;
                c->x1 = x1;
                c->y0 = y;
                c->y1 = y;
                c->x0 = x;
                flag = 1;
            }
        }
        if (!flag)
            for (int y1 = y + 1; y1 < image->height;
y1++) {
                if
                (free_border_data_array[y1][x] != 6) {
                    if
                    (free_border_data_array[y1][x] != 7) break;
                    c->x1 = x;
                    c->y0 = y;
                    c->y1 = y1;
                    c->x0 = x;
                    flag = 1;
                }
            }
        if (flag == 1) coords[coords_count++] = c;
        else {
            free(c);
            continue;
        }
    }
    else if (free_border_data_array[y][x] == 5) {
        int flag = 1;
        for (int x1 = x + 1; x1 < image->width; x1++)
        {
            if (free_border_data_array[y][x1] != 3) {
                if
                (free_border_data_array[y][x1]
== 5) {
                    c->x1 = x1;
                    c->x0 = x;
                }
                else flag = 0;
                break;
            }
        }
        for (int y1 = y + 1; y1 < image->height; y1++)
        {
            if (free_border_data_array[y1][x] != 3) {
                if
                (free_border_data_array[y1][x]
== 5) {
                    c->y1 = y1;
                    c->y0 = y;
                }
            }
        }
    }
}

```

```

        else flag = 0;
        break;
    }
}
if (c->y0 + 1 > c->y1 || c->x0 + 1 > c->x1)
flag = 0;
if (!flag) continue;
for (int x1 = c->x0 + 1; x1 <= c->x1; x1++) {
    if (free_border_data_array[c->y1][x1] !=
3) {
        if
(free_border_data_array[c->y1][x1] != 5) flag = 0;
    }
}
for (int y1 = c->y0 + 1; y1 <= c->y1; y1++) {
    if (free_border_data_array[y1][c->x1] !=
3) {
        if
(free_border_data_array[y1][c->x1] != 5) flag = 0;
    }
}
if (!flag) continue;
for (int y1 = c->y0 + 1; y1 < c->y1; y1++) {
    for (int x1 = c->x0 + 1; x1 < c->x1; x1++)
        if
(free_border_data_array[y1][x1] != 0) {
            flag = 0;
            break;
        }
    if (!flag)
        break;
}
if (flag) coords[coords_count++] = c;
else free(c);
continue;
}
}
}

for (int i = 0; i < coords_count; i++) {
    struct Coordinates *c = coords[i];
    create_border(image, c->x0, c->y0, c->x1, c->y1,
border_color, thickness);
    free(c);
}
free(coords);
free(free_border_data_array);
}

void print_help(){
    printf("\nВспомогательные функции:\n\n\
        -h, -help - справка, которую вы видите сейчас\n\n\
        -info - подробная информация об изображении\n\n\
        -i, -input - задаёт имя входного изображения\n\n\
        -o, -output - задаёт имя выходного изображения\n\n");
    printf("Функции по обработке изображений:\n\n\

```

```

        --copy - копирование заданной области (координаты
задаются в формате `x.y`)\n\
        --left_up - координаты левого верхнего угла области-
источника\n\
        --right_down - координаты правого нижнего угла
области-источника\n\
        --dest_left_up - координаты левого верхнего угла
области-назначения\n\
        --color_replace - заменяет все пиксели одного заданного
цвета на другой цвет\n\
        --old_color - цвет, который требуется заменить
(задаётся строкой `rrr.ggg.bbb`)\n\
        --new_color - цвет, на который требуется заменить
(задаётся строкой `rrr.ggg.bbb`)\n\
        --ornament - сделать рамку в виде узора\n\
        --pattern - узор (возможные значения: rectangle,
circle, semicircles)\n\
        --color - цвет (задаётся строкой `rrr.ggg.bbb`)\n\
        --thickness - ширина\n\
        --count - количество\n\
        --filled_rects - поиск всех залитых прямоугольников
заданного цвета\n\
        --color - цвет искомых прямоугольников (задаётся
строкой `rrr.ggg.bbb`)\n\
        --border_color - цвет линии для обводки\n\
        --thickness - толщина линии для обводки\n");
    }

```

Файл: main.c

```

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

int main(int argc, char **argv) {
    printf("Course work for option 5.16, created by Yakimova
Yuliya.\n");

    if (argc <= 1){
        print_help();
        return 0;
    }

    struct Png image;
    struct Color colors;
    struct Copy copy;
    struct Ornament ornam;
    struct Filled_Rect rect;

    //для getopt_long()
    int opt;
    int long_opt_index = 0;
    char *short_opts = "i:o:h";
    struct option long_opts[] = {
        {"copy", no_argument, NULL, 'c'},
        {"color_replace", no_argument, NULL, 'r'},
    }

```

```

{"ornament", no_argument, NULL, 'b'},
{"filled_rects", no_argument, NULL, 'f'},

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

{"left_up", required_argument, NULL, 'A'},
{"right_down", required_argument, NULL, 'B'},
{"dest_left_up", required_argument, NULL, 'D'},

{"old_color", required_argument, NULL, 'O'},
{"new_color", required_argument, NULL, 'N'},

{"pattern", required_argument, NULL, 'P'},
{"color", required_argument, NULL, 'C'},
{"thickness", required_argument, NULL, 'T'},
{"count", required_argument, NULL, 'X'},

{"border_color", required_argument, NULL, 'R'},
{NULL, 0, NULL, 0}
};
opt = getopt_long(argc, argv, short_opts, long_opts,
&long_opt_index);

int key = -1;
int input_flag = 0;
int output_flag = 0;
char input_file[255];
char output_file[255];
while (opt != -1) {
    switch (opt) {
        case 'h':
            print_help();
            return 0;
        case 'c':
            no_argschecker(argv[optind], "--copy");
            key = 'c';
            break;
        case 'r':
            no_argschecker(argv[optind], "--color_replace");
            key = 'r';
            break;
        case 'b':
            no_argschecker(argv[optind], "--ornament");
            key = 'b';
            break;
        case 'f':
            no_argschecker(argv[optind], "--filled_rects");
            key = 'f';
            break;
        case 'i':
            input_flag = 1;
            strcpy(input_file, optarg); //копирует значение
следующего аргумента командной строки в input_file

```

```

        break;
case 'o':
    output_flag = 1;
    strcpy(output_file, optarg);
    break;
case 'n':
    read_png_file(optarg, &image);
    print_png_info(&image);
    free_image_data(&image);
    return 0;
case 'O':
    colors.old_color = optarg;
    colors.old_color_f = 1;
    break;
case 'N':
    colors.new_color = optarg;
    colors.new_color_f = 1;
    break;
case 'A':
    copy.left_up = optarg;
    copy.left_up_f = 1;
    break;
case 'B':
    copy.right_down = optarg;
    copy.right_down_f = 1;
    break;
case 'D':
    copy.dest_left_up = optarg;
    copy.dest_left_up_f = 1;
    break;
case 'P':
    ornam.pattern = optarg;
    ornam.pattern_f = 1;
    break;
case 'C':
    ornam.color = optarg;
    ornam.color_f = 1;
    rect.color = optarg;
    rect.color_f = 1;
    break;
case 'T':
    ornam.thickness = optarg;
    ornam.thickness_f = 1;
    rect.thickness = optarg;
    rect.thickness_f = 1;
    break;
case 'X':
    ornam.count = optarg;
    ornam.count_f = 1;
    break;
case 'R':
    rect.border_color = optarg;
    rect.border_color_f = 1;
    break;
case '?':
    print_help();
    return 0;
default:

```

```

        break;
    };
    opt = getopt_long(argc, argv, short_opts, long_opts,
&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");
}
if (strcmp(input_file, output_file)==0){
    printf("Input and output file names cannot be the same\n");
    return 0;
}

read_png_file(input_file, &image);

switch (key) {
    case 'c':
        if (!copy.left_up_f || !copy.right_down_f
|| !copy.dest_left_up_f) {
            printf("Too few arguments have been entered to
copy.\n");
            return 0;
        }
        copy_img(&image, copy.left_up, copy.right_down,
copy.dest_left_up);
        write_png_file(output_file, &image);
        break;
    case 'r':
        if (!colors.new_color_f || !colors.old_color_f){
            printf("Too few arguments have been entered to
replace color.\n");
            return 0;
        }
        replace_color(&image, colors.new_color,
colors.old_color);
        write_png_file(output_file, &image);
        break;
    case 'b': //ornament
        if (!ornam.pattern_f) {
            printf("The pattern must be entered.\n");
            return 0;
        }
        if ((strcmp(ornam.pattern, "rectangle") != 0) &&
(strcmp(ornam.pattern, "circle") != 0) && (strcmp(ornam.pattern,
"semicircles") != 0)) {
            printf("The pattern must be `rectangle`, `circle` or
`semicircles`.\n");
            exit(41);
        }
        if (!strcmp(ornam.pattern, "rectangle")) {

```



```

        if      (!ornam.color_f      ||      !ornam.thickness_f
|| !ornam.count_f) {
            printf("Too few arguments have been entered to
create an ornament.\n");
            return 0;
        }
        create_rect_ornam(&image,                ornam.color,
ornam.thickness, ornam.count);
    }
    if (!strcmp(ornam.pattern, "circle")) {
        if(!ornam.color_f) {
            printf("The color has to be entered to create an
ornament.\n");
            return 0;
        }
        create_circle_ornam(&image, ornam.color);
    }
    if (!strcmp(ornam.pattern, "semicircles")) {
        if      (!ornam.color_f      ||      !ornam.thickness_f
|| !ornam.count_f) {
            printf("Too few arguments have been entered to
create an ornament.\n");
            return 0;
        }
        create_semicir_ornam(&image,                ornam.color,
ornam.thickness, ornam.count);
    }
    write_png_file(output_file, &image);
    break;
    case 'f':
        if      (!rect.color_f      ||      !rect.border_color_f
|| !rect.thickness_f) {
            printf("Too few arguments have been entered to trace
the rectangles.\n");
            return 0;
        }
        trace_rects(&image,      rect.color,      rect.border_color,
rect.thickness);
        write_png_file(output_file, &image);
        break;
    default:
        printf("Arguments have been entered incorrectly.\n");
        return 0;
    };
}

```

Файл: Makefile

```

all: main.o input.o functions.o
    gcc *.o -lpng -lm -o cw -std=c99
main.o: main.c input.o functions.o *.h
    gcc -c -lpng -lm main.c -std=c99
input.o: input.c input.h library.h structures.h
    gcc -c -lpng -lm input.c -std=c99
functions.o: functions.c *.h
    gcc -c -lpng -lm functions.c -std=c99

```

```
clean:  
    rm *.o cw
```