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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Обработка PNG файла

Студент гр. 2384 _____ Кузьминых Е.М.

Преподаватель _____ Гаврилов А.В.

Санкт-Петербург

2023

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студент Кузьминых Е.М.

Группа 2384

Тема работы: Обработка PNG файла
Вариант №13

Программа должна иметь CLI или GUI. Более подробно тут: http://se.moevm.info/doku.php/courses:programming:rules_extra_kurs
Общие сведения

Формат картинки PNG (рекомендуем использовать библиотеку libpng) файл всегда соответствует формату PNG
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 реализовывать следующий функционал по обработке PNG-файла:

Преобразовать в Ч/Б изображение (любым простым способом). Функционал определяется:

- Координатами левого верхнего угла области
- Координатами правого нижнего угла области
- Алгоритмом, если реализовано несколько алгоритмов преобразования изображения (по желанию студента)

Рисование отрезка. Отрезок определяется:

- координатами начала
- координатами конца
- цветом
- толщиной

Инвертировать цвета в заданной окружности. Окружность определяется либо координатами левого верхнего и правого нижнего угла квадрата, в который она вписана, либо координатами ее центра и радиусом

Обрезка изображения. Требуется обрезать изображение по заданной области. Область определяется:

- Координатами левого верхнего угла
- Координатами правого нижнего угла

Предполагаемый объем пояснительной записки:
Не менее 14 страниц.
Дата выдачи задания: 22.03.2023

Дата сдачи реферата: 20.05.2023

Дата защиты реферата: 22.05.2023

Студент _____ Кузьминых Е.М.

Преподаватель _____ Гаврилов А.В.

АННОТАЦИЯ

Данный код представляет собой реализацию программы обработки изображений в формате PNG с использованием языка программирования C и библиотеки *libpng*. Для хранения изображений используются структуры, а для работы с ними используются функции стандартной библиотеки языка.

Программа представляет собой CLI-утилиту, которая считывает ключи со значениями пути к исходному изображению, ключи для функций, значения для выполнения функций и название для нового изображения. В зависимости от полученных ключей, программа изменяет исходное изображение и сохраняет его с новым названием.

Для запуска программы необходимо зайти в корневую папку проекта и выполнить команду "gcc main.c && ./a.out" в терминале. Затем пользователь должен ввести ключи, отвечающие за путь к изображению для редактирования, необходимые для выполнения функций, аргументы для функций и имя для измененного файла.

Таким образом, данная программа позволяет легко и быстро обрабатывать изображения в формате PNG с помощью CLI-утилиты и библиотеки *libpng*.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью данной работы является создание программы на языке программирования C для редактирования изображений в формате PNG. Для достижения поставленной цели необходимо решить следующие задачи:

1. Разработать структуру для хранения информации о PNG файле.
2. Реализовать алгоритм открытия файла с расширением PNG.
3. Реализовать алгоритм сохранения изображения в формате PNG.
4. Создать функции для редактирования считанного изображения.
5. Реализовать возможность ввода пользователем команд через *getopt_long* для выбора изображения для редактирования и методов редактирования.

2. ХОД ВЫПОЛНЕНИЯ РАБОТЫ

2.1. Создание структуры для хранения изображения

Структура *Png* была создана для хранения информации об изображении в формате PNG. Она содержит следующие поля:

- *width*: ширина изображения в пикселях
- *height*: высота изображения в пикселях
- *color_type*: тип цветовой палитры изображения
- *bit_depth*: глубина цвета изображения
- *png_ptr*: указатель на структуру *png_struct*, которая используется для чтения изображения из файла
- *info_ptr*: указатель на структуру *png_info*, которая содержит информацию о изображении
- *number_of_passes*: количество проходов при чтении изображения
- *row_pointers*: массив указателей на строки пикселей изображения

2.2. Создание функции для ввода изображения

Функция *terminate* используется для освобождения памяти, выделенной для структуры *Png*, вывода сообщения об ошибке на экран и завершения программы.

Функция *read_png_file* используется для чтения PNG-изображения из файла и заполнения структуры *Png* соответствующими данными. Первым шагом функция определяет массив *header*, который используется для проверки, содержит ли файл заголовок PNG. Затем функция открывает файл с именем *file_name* с помощью функции *fopen*. Если файл не может быть открыт, функция вызывает функцию *terminate*. С помощью функции *fread* функция считывает первые 8 байтов файла в массив *header*. Затем функция использует функцию *png_sig_cmp* для проверки, содержит ли файл корректный заголовок PNG. Затем функция создает структуру *png_struct* с

помощью *png_create_read_struct*, которая инициализирует структуру *png_struct* с версией библиотеки PNG_LIBPNG_VER_STRING. Далее функция создает структуру *png_info* с помощью *png_create_info_struct* для хранения информации о изображении. Функция устанавливает обработчик ошибок с помощью функции *setjmp*, чтобы можно было восстановиться от ошибок в процессе чтения файла. Затем вызывается *png_init_io* для инициализации ввода/вывода в структуре *png_struct* и устанавливает количество байтов, которые должны быть проигнорированы в начале файла, с помощью функции *png_set_sig_bytes*. Затем вызывается *png_read_info* для чтения информации об изображении и заполнения структуры *png_info*. Затем функция вызывает функцию *png_set_interlace_handling* для установки обработки интерфейса и вызывает функцию *png_read_update_info* для обновления информации о изображении. Функция выделяет память для хранения строк пикселей с помощью *malloc* и заполняет эту память с помощью функции *png_read_image*. Если происходят ошибки в процессе чтения изображения, функция вызывает функцию *terminate*. После чтения изображения, функция закрывает файл с помощью функции *fclose* и возвращает структуру *Png*.

2.3. Создание функции для сохранения изображения

Функция *write_png_file* была создана для записи изображения в формате PNG в файл. Сначала функция открывает файл с именем *file_name* с помощью функции *fopen*. Если файл не может быть открыт, функция вызывает функцию *terminate*. Затем функция создает структуру *png_struct*, используя функцию *png_create_write_struct*, которая инициализирует структуру *png_struct* с версией библиотеки PNG_LIBPNG_VER_STRING и несколькими другими параметрами. Далее создается структура *png_info* с помощью функции *png_create_info_struct* для хранения информации о изображении. Функция устанавливает обработчик ошибок с помощью функции *setjmp*, чтобы можно было восстановиться от ошибок в процессе

записи файла. Затем функция вызывает функцию *png_init_io* для инициализации ввода/вывода в структуре *png_struct*. Функция использует функцию *png_set_IHDR* для установки заголовка PNG, включая ширину, высоту, глубину цвета и тип цветовой палитры. Затем функция вызывает функцию *png_write_info* для записи информации о изображении в файл. Для записи изображения в файл функция выделяет память для хранения строк пикселей изображения с помощью функции *malloc* и затем заполняет эту память с помощью функции *png_write_image*. Затем функция вызывает функцию *png_write_end* для завершения записи файла. Наконец, функция освобождает память, занятую строками пикселей, с помощью функции *free*, и закрывает файл с помощью функции *fclose*.

2.4. Создание функции вывода справки

Для вывода информации о работе программы была реализована функция *print_help,* которая выводит справку о работе программы. Печатаются ключи, а также описание к каждому из них.

2.5. Написание 1-ой функции

Была создана функция *black_and_white.* Эта функция принимает указатель на struct Png и флаг (0 или 1), и изменяет изображение, чтобы оно было черно-белым. Функция перебирает все пиксели изображения и меняет их цвет покомпонентно. Если флаг не задан или равен 0, то каждый компонент пикселя заменяется средним арифметическим всех компонентов пикселя. Если флаг равен 1, то каждая компонента цвета заполняется по следующей формуле: 0.36R+0.53G+0.11*B, где R, G, B – компоненты пикселя.

2.6. Написание 2-ой функции

Для рисования линии была написана функция *draw*, принимающая на вход *struct Png* image*, хранящая информацию об изображении, координаты 2

точек, через которые будет проходить прямая, компоненты для цвета в формате RGBA, размер толщины линии, а также вспомогательная функция *change_pixel,* принимающая на вход координаты пикселя и значения цвета для него. Функция *change_pixel* получает пиксель по координатам и меняет его значения цвета на полученные. Функция рисует линию, используя алгоритм Брезенхема. Алгоритм принимает на вход координаты начала и конца линии (x0, y0) и (x1, y1), а также изображение, на котором линия будет нарисована. Алгоритм начинается с вычисления значений *dx* и *dy* - разницы между координатами x и y начальной и конечной точек соответственно. Затем определяются значения *sx* и *sy*, которые будут использоваться для определения направления движения по осям x и y. Затем алгоритм начинает итерацию цикла, пока не будет достигнута конечная точка линии. На каждом шаге цикла вычисляется значение ошибки err, которое используется для определения, какую следующую точку нарисовать. Если значение err больше, чем *-dy*, то следующая точка будет находиться на той же строке, что и текущая. Если значение *err* меньше, чем *dx*, то следующая точка будет находиться на следующей строке. На каждом шаге цикла также проверяется, находится ли текущая точка в пределах изображения, и если да, то рисуется пиксель в этой точке. Для этого используется функция *change_pixel*, которая устанавливает цвет пикселя в соответствии с переданными значениями цвета и альфа-канала. Для реализации рисования толщины была написана функция *draw_circle*, которая рисует закрашенный круг в точке на прямой с радиусом, равным толщине линии/2.

2.7. Написание 3-ей функции

Для инвертирования цвета в изображении были созданы функции *inverse_circle* и *inverse_square*. *inverse_circle* принимает указатель на *struct Png*, а также координаты центра круга и его радиус. Функция перебирает только те пиксели, которые находятся внутри круга, заданного координатами центра и радиуса, и инвертирует цвет каждого пикселя покомпонентно по

следующей формуле: 255 - (значение цвета пикселя). *inverse_square* принимает указатель на *struct Png*, а также координаты квадрата, в который вписан круг. Из данных координат функция вычисляет координаты центра круга и его радиус, а затем вызывает функцию *inverse_circle* с этими значениями, чтобы инвертировать цвет каждого пикселя, находящегося внутри круга. Для вызова одной из функций используется ключ --inverse. В зависимости от введенных пользователем данных, вызывается либо функция *inverse_circle*, если введены координаты центра круга и его радиус, либо функция *inverse_square*, если введены координаты квадрата, в который вписан круг.

2.8. Написание 4-ой функции

Для обрезки изображения была создана функция *crop*. Эта функция принимает указатель на *struct Png*, а также координаты области, которую нужно обрезать. Функция создает новый массив пикселей *new_pointers* с помощью *malloc*, выделяя для него память, достаточную для хранения пикселей изображения, находящихся в заданной области. Затем функция копирует в новый массив пикселей *new_pointers* пиксели изображения, находящиеся в заданной области, используя функцию *memcpy*. После копирования пикселей в новый массив, функция *crop* заменяет изначальное значение поля *row_pointers* в структуре *image* на значение нового массива пикселей *new_pointers*. Также функция изменяет значения полей *height* и *width* в структуре *image* на соответствующие значения для обрезанного изображения. Таким образом, результатом выполнения функции *crop* является изображение с обрезанной областью и измененными значениями *height* и *width* в структуре *image*.

2.9. Создание функции вывода информации об изображении

Для вывода информации об изображении была реализована функция *print_info*. Эта функция принимает указатель на *struct Png* и выводит

информацию об изображении, такую как его ширина, высота, тип и глубина цвета. Информация, выводимая функцией *print_info*, соответствует текущему состоянию изображения, которое может быть изменено до вызова этой функции.

2.10. Написание ввода ключей (CLI)

В функции *main* реализовано получение аргументов из командной строки и обрабатывает их с помощью библиотеки *getopt_long*. Она начинается с определения структуры *Config*, которая содержит набор свойств, используемых для обработки изображения, таких как координаты x0, x1, y0, y1, цветовая палитра *red, green, blue, alpha*, радиус и толщина линии, а также другие параметры, такие как *crop, draw, invert* и т.д. Затем функция определяет переменные *path_to_file* и *name_file*, которые используются для хранения пути к файлу и его имени соответственно. Они выделяются с помощью *malloc*. Затем определена структура *option,* которая используется для определения длинных параметров командной строки, таких как --help или --crop. Эта структура содержит информацию о параметре, его аргументах и соответствующем ему символе в короткой записи команды. Далее определена структура *Png,* которая используется для работы с изображением в формате PNG. Цикл *while* в функции *main* использует функцию *getopt_long* для обработки параметров командной строки. Он проходит по всем параметрам, переданным в командной строке, и соответствующим образом устанавливает значения свойств структуры *Config*. Затем функция осуществляет обработку изображения в соответствии с заданными параметрами, используя функции *crop, inverse_circle, inverse_square, draw_tg, black_and_while* и *print_info*. По завершении работы функция записывает измененное изображение в файл с заданным именем или по умолчанию.

ЗАКЛЮЧЕНИЕ

В результате изучения библиотеки *libpng* и методов редактирования изображений была разработана программа, которая позволяет пользователю передавать аргументы через командную строку с помощью библиотеки *getopt_long*. Программа принимает на вход от пользователя ключи с названием файла для редактирования, необходимыми функциями, значениями для функций и названием файла, в котором будет сохранено измененное изображение.

Программа считывает выбранный пользователем файл и применяет к нему заданные функции. Затем программа сохраняет измененный файл с указанным пользователем названием.

Примеры работы программы см. в приложении А.

Разработанный программный код см. в приложении Б.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Керниган Б., Ритчи Д. Язык программирования Си. СПб.: "Невский Диалект", 2001. 352 с.

2. gnu.org: [Электронный ресурс]. URL: https://www.gnu.org

3. *libpng* manual: [Электронный ресурс]. URL: http://www.libpng.org

4. Cplusplus: [Электронный ресурс]. URL: https://cplusplus.com/

ПРИЛОЖЕНИЕ А
ПРИМЕРЫ РАБОТЫ ПРОГРАММЫ

1. Запуск программы без указания флагов:

```
./cw_2
```

Рисунок 1 – Ввод пользователя

```
Usage: [OPTIONS] [ARGUMENTS]

OPTIONS:
    --help, -h          Information about using the program
   --black_and_white,-W Translating an image into black and white
    --crop, -C          Cropping an image to a specified area
    --draw, -D          Drawing a line
    --inverse, -I       Inverting the image in a given circle

ARGUMENTS:
    --path, -P          Entering an image name to edit
    --name, -N          Entering the name of the final image
    --x0, -x            Coordinates of 1 point on Ox
    --y0, -y            Coordinates of 1 point on the Oy
    --x1, -w            Coordinates of 2 points on Ox
    --y1, -z            Coordinates of 2 points on the Oy
    --radius, -p        Circle radius length
    --red, -r           Value for red (0 to 255)
    --green, -g         Value for green (0 to 255)
    --blue, -b          Value for blue (0 to 255)
    --alpha, -a         Value for transparency (0 to 255)
    --thickness, -t     Value for line thickness
    --black, -B         Changing the algorithm for converting the image to black and white
    --print, -i         Displaying information about the captured image
```

Рисунок 2 – Результат работы программы

2. Ввод пользователем случайного текста:

```
./cw_2 bla bla bla randomtext
```

Рисунок 3 – Ввод пользователя

```
Usage: [OPTIONS] [ARGUMENTS]

OPTIONS:
    --help, -h          Information about using the program
   --black_and_white,-W Translating an image into black and white
    --crop, -C          Cropping an image to a specified area
    --draw, -D          Drawing a line
    --inverse, -I       Inverting the image in a given circle

ARGUMENTS:
    --path, -P          Entering an image name to edit
    --name, -N          Entering the name of the final image
    --x0, -x            Coordinates of 1 point on Ox
    --y0, -y            Coordinates of 1 point on the Oy
    --x1, -w            Coordinates of 2 points on Ox
    --y1, -z            Coordinates of 2 points on the Oy
    --radius, -p        Circle radius length
    --red, -r           Value for red (0 to 255)
    --green, -g         Value for green (0 to 255)
    --blue, -b          Value for blue (0 to 255)
    --alpha, -a         Value for transparency (0 to 255)
    --thickness, -t     Value for line thickness
    --black, -B         Changing the algorithm for converting the image to black and white
    --print, -i         Displaying information about the captured image
```

Рисунок 4 – Результат работы программы

3. Ввод пользователем несуществующего ключа:

```
./cw_2 --random_flag
```

Рисунок 5 – Ввод пользователя

```
./cw_2: unrecognized option '--random_flag'
Usage: [OPTIONS] [ARGUMENTS]

OPTIONS:
    --help, -h          Information about using the program
   --black_and_white,-W Translating an image into black and white
    --crop, -C          Cropping an image to a specified area
    --draw, -D          Drawing a line
    --inverse, -I       Inverting the image in a given circle

ARGUMENTS:
    --path, -P          Entering an image name to edit
    --name, -N          Entering the name of the final image
    --x0, -x            Coordinates of 1 point on Ox
    --y0, -y            Coordinates of 1 point on the Oy
    --x1, -w            Coordinates of 2 points on Ox
    --y1, -z            Coordinates of 2 points on the Oy
    --radius, -p        Circle radius length
    --red, -r           Value for red (0 to 255)
    --green, -g         Value for green (0 to 255)
    --blue, -b          Value for blue (0 to 255)
    --alpha, -a         Value for transparency (0 to 255)
    --thickness, -t     Value for line thickness
    --black, -B         Changing the algorithm for converting the image to black and white
    --print, -i         Displaying information about the captured image
```

Рисунок 6 – Результат работы программы

4. Выполнение перевода изображения в Ч/Б

```
./cw_2 --path kot.png --black_and_white --name black_kot.png
```

Рисунок 7 – Ввод пользователя

Рисунок 8 – Результат работы программы

5. Выполнение инвертирование цвета в заданной окружности:

```
./cw_2 --path kot.png --inverse --x0 600 --y0 600 --radius 250 --name new_kot.png
```

Рисунок 9 – Ввод пользователя

Рисунок 10 – Результат работы программы

6. Выполнение обрезки изображения:

```
./cw_2 --path kot.png --crop --x0 450 --y0 450 --x1 800 --y1 800 --name crop_kot.png
```

Рисунок 11 – Ввод пользователя

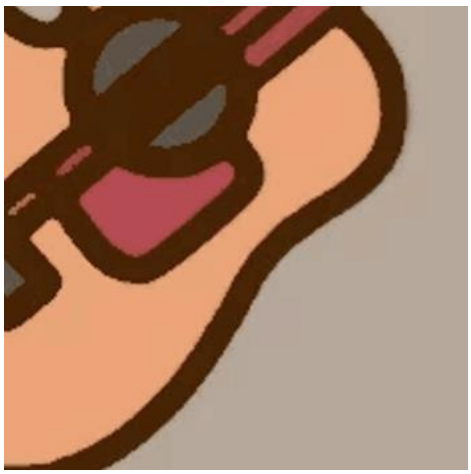

Рисунок 12 – Результат работы программы

7. Выполнение рисования линии:

```
./cw_2 -P kot.png -D --x0 350 --y0 800 --x1 870 --y1 500 -r 255 -g 170 -b 0 -t 15 -N drawkot.png
```

Рисунок 13 – Ввод пользователя

Рисунок 14 – Результат работы программы

8. Выполнение рисования линии с выходящими за область координатами:

```
./cw_2 -P kot.png -D --x0 3500 --y0 80 --x1 -10 --y1 500 -r 255 -g 170 -b 0 -t 15 -N drawkot.png
```

Рисунок 15 – Ввод пользователя

Рисунок 16 – Результат работы программы

9. Выполнение вывода справки для работы с программой:

```
./cw_2 --help
```

Рисунок 17 – Ввод пользователя

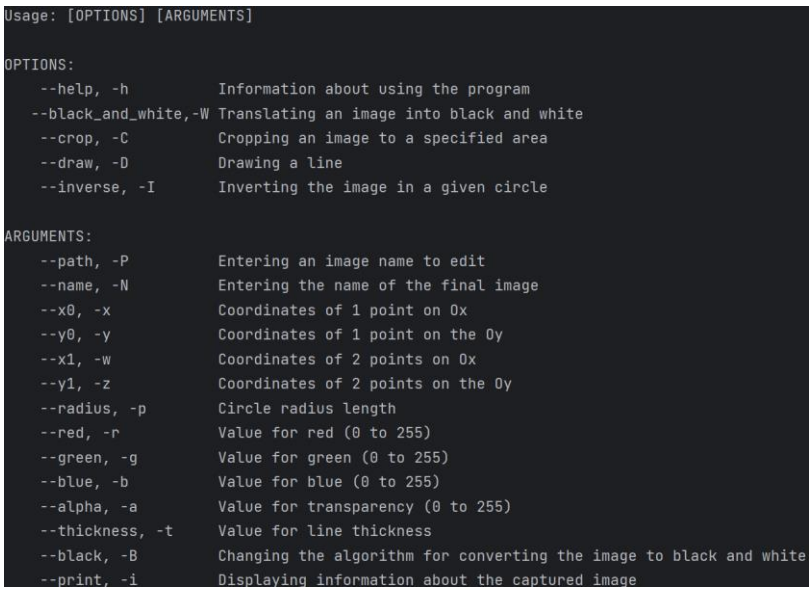

Рисунок 18 – Результат работы программы

10.Вывод информации об изображении:

```
./cw_2 --path kot.png --black_and_white --name black_kot.png --print
```

Рисунок 19 – Ввод пользователя

```
Information about the image:
Image width: 1024
Image height: 1024
Image color type: 6
Image bit depth: 8
```

Рисунок 20 – Результат работы программы

ПРИЛОЖЕНИЕ Б
ИСХОДНЫЙ КОД ПРОГРАММЫ

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

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

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

void terminate(char *text, struct Png *image) {
    puts(text);
    png_destroy_read_struct(&image->png_ptr,     &image->info_ptr,
NULL);
    if (image->row_pointers) {
        for (int y = 0; y < image->height; y++)
            free(image->row_pointers[y]);
        free(image->row_pointers);
        free(image);
    }
    exit(-1);
}


void read_png_file(char *file_name, struct Png *image) {
    char header[8];
```

```
        FILE *fp = fopen(file_name, "rb");
        if (!fp) {
            terminate("Sorry, file don't opened\n", image);
        }

        fread(header, 1, 8, fp);
        if (png_sig_cmp(header, 0, 8)) {
            terminate("File does not have a PNG extension\n", image);
        }

        image->png_ptr                                              =
png_create_read_struct(PNG_LIBPNG_VER_STRING, NULL, NULL, NULL);

        if (!image->png_ptr) {
            terminate("Error creating the PNG structure\n", image);
        }

        image->info_ptr = png_create_info_struct(image->png_ptr);
        if (!image->info_ptr) {
            terminate("Error creating the PNG structure\n", image);
        }

        if (setjmp(png_jmpbuf(image->png_ptr))) {
            terminate("Error  calling  the  png_init_io  function\n",
image);
        }

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

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

        image->row_pointers = (png_bytep *) malloc(sizeof(png_bytep) *
image->height);
        for (int 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) {

        FILE *fp = fopen(file_name, "wb");
        if (!fp) {
            terminate("Error  when  opening  the  resulting  file\n",
image);
        }

        image->png_ptr                                                 =
png_create_write_struct(PNG_LIBPNG_VER_STRING, NULL, NULL, NULL);

        if (!image->png_ptr) {
```

```
            terminate("Error:    png_create_write_struct    failed\n",
image);
        }

        image->info_ptr = png_create_info_struct(image->png_ptr);
        if (!image->info_ptr) {
            terminate("Error:    png_create_info_struct    failed\n",
image);
        }

        if (setjmp(png_jmpbuf(image->png_ptr))) {
            terminate("Error  calling  the  png_init_io  function\n",
image);
        }

        png_init_io(image->png_ptr, fp);


        if (setjmp(png_jmpbuf(image->png_ptr))) {
            terminate("Error during writing header\n", image);
        }

        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);

        if (setjmp(png_jmpbuf(image->png_ptr))) {
            terminate("Byte reading error\n", image);
        }

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

        if (setjmp(png_jmpbuf(image->png_ptr))) {
```

```
        terminate("Error during end of write\n", image);
    }

    png_write_end(image->png_ptr, NULL);

    for (int y = 0; y < image->height; y++)
        free(image->row_pointers[y]);
    free(image->row_pointers);

    fclose(fp);
}

void check_coords(struct Png *image, int *x, int *y) {
    if (*x < 0) {
        *x = 0;
    }
    if (*x >= image->width) {
        *x = image->width - 1;
    }
    if (*y < 0) {
        *y = 0;
    }
    if (*y >= image->height) {
        *y = image->height - 1;
    }

}

void black_and_white(struct Png *image, int flag) {
    int x, y, sum;
    int count = png_get_channels(image->png_ptr, image->info_ptr);
    for (y = 0; y < image->height; y++) {
        png_byte *row = image->row_pointers[y];
        for (x = 0; x < image->width; x++) {
            png_byte *ptr = &(row[x * count]);
            if (flag == 0) {
                sum = (ptr[0] + ptr[1] + ptr[2]) / 3;
            } else {
```

```
                    sum = 0.36 * ptr[0] + 0.53 * ptr[1] + 0.11 * ptr[2];
                }
                for (int i = 0; i < 3; i++) {
                    ptr[i] = sum;
                }
            }
        }
    }

    void crop(struct Png *image, int x_start, int y_start, int x_end,
int y_end) {
        check_coords(image, &x_start, &y_start);
        check_coords(image, &x_end, &y_end);
        if (x_start <= 0 && y_start <= 0 && x_end <= 0 && y_end <= 0)
{
            printf("Enter the correct coordinates\n");
            return;
        }
        if (x_start >= image->width - 1 && x_end >= image->width - 1
&& y_start >= image->height - 1 &&
            y_end >= image->height - 1) {
            printf("Enter the correct coordinates\n");
            return;
        }
        png_bytep      *new_pointers      =      (png_bytep      *)
malloc(sizeof(png_bytep) * (y_end - y_start));
        int count = png_get_channels(image->png_ptr, image->info_ptr);
        for (int y = y_start; y < y_end; y++) {
            png_byte *row = image->row_pointers[y];
            new_pointers[y    -    y_start]    =    (png_byte    *)
malloc(png_get_rowbytes(image->png_ptr, image->info_ptr));
            memcpy(new_pointers[y - y_start], row + x_start * count,
(x_end - x_start) * count);
        }
        image->row_pointers = new_pointers;
        image->height = y_end - y_start;
        image->width = x_end - x_start;
    }
```

```
    void inverse_circle(struct Png *image, int x0, int y0, int r) {
        check_coords(image, &x0, &y0);
        int count = png_get_channels(image->png_ptr, image->info_ptr);
        for (int y = y0 - r; y < y0 + r; y++) {
            png_byte *row = image->row_pointers[y];
            for (int x = x0 - r; x < x0 + r; x++) {
                if ((y - y0) * (y - y0) + (x - x0) * (x - x0) <= r *
r) {
                    if (x >= 0 && x < image->width && y >= 0 && y <
image->height) {
                        png_byte *ptr = &(row[x * count]);
                        for (int i = 0; i < 3; i++) {
                            ptr[i] = 255 - ptr[i];
                        }
                    }
                }
            }
        }

    void inverse_square(struct Png *image, int x0, int y0, int x1, int
y1) {
        int radius = (x1 - x0) / 2;
        int x = (x1 + x0) / 2;
        int y = (y1 + y0) / 2;
        inverse_circle(image, x, y, radius);
    }

    void change_pixel(struct Png *image, int x, int y, png_byte r,
png_byte g, png_byte b, png_byte a) {
        if (x < 0 || x >= image->width || y < 0 || y >= image->height)
{
            return;
        }
        int count = png_get_channels(image->png_ptr, image->info_ptr);
        png_bytep ptr = image->row_pointers[y] + x * count;
        ptr[0] = r;
```

```
        ptr[1] = g;
        ptr[2] = b;
        if (count > 3) {
            ptr[3] = a;
        }
    }

    void draw_circle(struct Png *image, int x, int y, png_byte r,
png_byte g, png_byte b, png_byte a, int thickness) {
        int radius = thickness / 2;
        for (int i = y - radius; i < y + radius; i++) {
            for (int j = x - radius; j < x + radius; j++) {
                if ((i - y) * (i - y) + (j - x) * (j - x) <= radius *
radius) {
                    change_pixel(image, j, i, r, g, b, a);
                }
            }
        }
    }

    void draw(struct Png *image, int x0, int y0, int x1, int y1,
png_byte r, png_byte g, png_byte b, png_byte a,
              int thickness) {
        check_coords(image, &x0, &y0);
        check_coords(image, &x1, &y1);

        int dx = abs(x1 - x0);
        int dy = abs(y1 - y0);

        int sx = x0 < x1 ? 1 : -1;
        int sy = y0 < y1 ? 1 : -1;

        int err = dx - dy;

        while (x0 != x1 || y0 != y1) {
            if (x0 >= 0 && x0 < image->width && y0 >= 0 && y0 <
image->height) {
                draw_circle(image, x0, y0, r, g, b, a, thickness);
```

```
                change_pixel(image, x0, y0, r, g, b, a);
            }

            int e2 = 2 * err;

            if (e2 > -dy) {
                err -= dy;
                x0 += sx;
            }

            if (e2 < dx) {
                err += dx;
                y0 += sy;
            }

        }
    }

    void print_info(struct Png *image) {
        printf("Information about the image:\n");
        printf("Image width: %d\n", image->width);
        printf("Image height: %d\n", image->height);
        printf("Image color type: %d\n", image->color_type);
        printf("Image bit depth: %d\n", image->bit_depth);
    }

    void print_help() {
        puts("Usage: [OPTIONS] [ARGUMENTS]\n");
        puts("OPTIONS:");
        puts("     --help, -h             Information about using the
program");
        puts("   --black_and_white,-W Translating an image into black
and white");
        puts("    --crop, -C          Cropping an image to a specified
area");
        puts("    --draw, -D          Drawing a line");
        puts("    --inverse, -I        Inverting the image in a given
circle\n");
```

```
        puts("ARGUMENTS:");
        puts("    --path, -P         Entering an image name to edit");
        puts("     --name, -N          Entering the name of the final
image");
        puts("    --x0, -x           Coordinates of 1 point on Ox");
        puts("     --y0, -y            Coordinates of 1 point on the
Oy");
        puts("    --x1, -w           Coordinates of 2 points on Ox");
        puts("     --y1, -z            Coordinates of 2 points on the
Oy");
        puts("    --radius, -p       Circle radius length");
        puts("    --red, -r          Value for red (0 to 255)");
        puts("    --green, -g        Value for green (0 to 255)");
        puts("    --blue, -b         Value for blue (0 to 255)");
        puts("     --alpha, -a          Value for transparency (0 to
255)");
        puts("    --thickness, -t    Value for line thickness");
        puts("     --black, -B           Changing the algorithm for
converting the image to black and white");
        puts("    --print, -i        Displaying information about the
captured image");
    }

    struct Config {
        int x0;
        int x1;
        int y0;
        int y1;
        int red;
        int green;
        int blue;
        int alpha;
        int radius;
        int thickness;
        int black_and_white;
        int crop;
        int invert;
        int draw;
```

```
    int help;
    int black;
    int print;
};

int main(int argc, char **argv) {
    int rez = 0;
    char *path_to_file;
    char *name_file;
    struct Config config = {0, 0, 0, 0, 255, 0, 0, 255, -1, 1, 0,
                            0, 0, 0, 0, 0, 0};
    path_to_file = malloc(sizeof(char) * PATH_MAX);
    name_file = malloc(sizeof(char) * PATH_MAX);
    static struct option long_option[] = {
            {"help",           no_argument,       0, 'h'},
            {"x0",             required_argument, 0, 'x'},
            {"x1",             required_argument, 0, 'y'},
            {"y0",             required_argument, 0, 'w'},
            {"y1",             required_argument, 0, 'z'},
            {"radius",         required_argument, 0, 'p'},
            {"black_and_white", no_argument,      0, 'W'},
            {"black",          no_argument,       0, 'B'},
            {"crop",           no_argument,       0, 'C'},
            {"inverse",        no_argument,       0, 'I'},
            {"draw",           no_argument,       0, 'D'},
            {"red",            required_argument, 0, 'r'},
            {"green",          required_argument, 0, 'g'},
            {"blue",           required_argument, 0, 'b'},
            {"alpha",          required_argument, 0, 'a'},
            {"thickness",      required_argument, 0, 't'},
            {"path",           required_argument, 0, 'P'},
            {"name",           required_argument, 0, 'N'},
            {"print",          no_argument,       0, 'i'}
    };
    struct Png image;
    while       ((rez       =       getopt_long(argc,       argv,
"hx:y:w:z:WCiDIN:P:r:g:b:a:t:p:B", long_option, NULL)) != -1) {
        switch (rez) {
```

```
case 'h':
    config.help = 1;
    break;
case 'x':
    config.x0 = atoi(optarg);
    break;
case 'y':
    config.x1 = atoi(optarg);
    break;
case 'w':
    config.y0 = atoi(optarg);
    break;
case 'z':
    config.y1 = atoi(optarg);
    break;
case 'p':
    config.radius = atoi(optarg);
    break;
case 'W':
    config.black_and_white = 1;
    break;
case 'r':
    config.red = atoi(optarg);
    break;
case 'g':
    config.green = atoi(optarg);
    break;
case 'b':
    config.blue = atoi(optarg);
    break;
case 'a':
    config.alpha = atoi(optarg);
    break;
case 't':
    config.thickness = atoi(optarg);
    break;
case 'C':
    config.crop = 1;
```

```
            break;
        case 'I':
            config.invert = 1;
            break;
        case 'D':
            config.draw = 1;
            break;
        case 'P':
            strcpy(path_to_file, optarg);
            break;
        case 'N':
            strcpy(name_file, optarg);
            break;
        case 'B':
            config.black = 1;
            break;
        case 'i':
            config.print = 1;
            break;

    }
}

if (strstr(path_to_file, ".png") != NULL) {
    read_png_file(path_to_file, &image);
} else {
    print_help();
    exit(0);
}
if (config.help != 0) {
    print_help();
}
if (config.crop == 1) {
    crop(&image, config.x0, config.y0, config.x1, config.y1);
}
if (config.invert == 1) {
    if (config.radius == -1) {
```

```
                inverse_square(&image,      config.x0,      config.y0,
config.x1, config.y1);
            } else {
                inverse_circle(&image,      config.x0,      config.y0,
config.radius);
            }
        }
        if (config.draw == 1) {
            draw(&image, config.x0, config.y0, config.x1, config.y1,
config.red, config.green, config.blue, config.alpha,
                 config.thickness);
        }
        if (config.black_and_white == 1) {
            black_and_white(&image, config.black);
        }
        if (config.print == 1) {
            print_info(&image);
        }
        if (strstr(name_file, ".png") != NULL) {
            write_png_file(name_file, &image);
        } else {
            write_png_file("new_file.png", &image);
        }
        return 0;
    }
```