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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Обработка изображения на C

Студентка гр. 0382 _____ Охотникова Г.С.

Преподаватель _____ Берленко Т.А.

Санкт-Петербург
2021

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студентка Охотникова Г.С.

Группа 0382

Тема работы : обработка изображений на С.

Исходные данные:
На вход подается изображение в формате PNG. Требуется реализовать считывание, возможность изменять картинку, записывать на диск. Функционал реализовывается с помощью CLI.

Содержание пояснительной записки:
«Содержание», «Введение», «Ход выполнения работы», «Тестирование», «Заключение», «Список использованных источников»

Предполагаемый объем пояснительной записки:
Не менее 28 страниц.

Дата выдачи задания: 05.04.2021

Дата сдачи реферата: 25.05.2021

Дата защиты реферата: 27.05.2021

Студентка _____ Охотникова Г.С.

Преподаватель _____ Берленко Т.А.

АННОТАЦИЯ

В ходе выполнения курсовой работы была разработана программа на языке Си с интерфейсом CLI(Command Line Interface), которая позволяет пользователю с помощью вводимых ключей редактировать изображение формата PNG. Возможность работы с форматов PNG доступна благодаря использованию библиотеки libpng, а реализация CLI выполнена с использованием библиотеки getopt.h.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Требуется реализовать программу, которая осуществляет обработку изображения формата PNG. Программа должна иметь интерфейс CLI и осуществлять выполнение следующих функций:

- рисование квадрата;
- смена местами четырех кусков области;
- нахождение самого часто встречаемого цвета и изменение его;
- инверсия цвета в заданной области.

1. ЦЕЛЬ И ЗАДАНИЕ РАБОТЫ

1.1. Цель работы.

Освоить принципы работы с изображениями в языке Си, изучить библиотеку libpng и getopt.h, реализовать программу по обработке изображения формата PNG.

1.2. Задание.

Вариант 12 Программа должна иметь CLI или GUI.

Общие сведения:

● Формат картинки PNG

● 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:

1. Рисование квадрата. Квадрат определяется:

○ Координатами левого верхнего угла

○ Размером стороны

○ Толщиной линий

○ Цветом линий ○ Может быть залит или нет

○ Цветом которым он залит, если пользователем выбран залитый

2. Поменять местами 4 куска области. Выбранная пользователем прямоугольная область делится на 4 части и эти части меняются местами. Функционал определяется:

○ Координатами левого верхнего угла области

○ Координатами правого нижнего угла области

○ Способом обмена частей: “по кругу”, по диагонали

3. Находит самый часто встречаемый цвет и заменяет его на другой заданный цвет. Функционал определяется:

○Цветом, в который надо перекрасить самый часто встречаемый цвет.

4. Инверсия цвета в заданной области. Функционал определяется:

○ Координатами левого верхнего угла области

○ Координатами правого нижнего угла области

2. ХОД ВЫПОЛНЕНИЯ РАБОТЫ

2.1. Структура Png.

Данная структура содержит в себе информацию о ширине и высоте изображения, глубине цвета, типе цвета, указатели *png_ptr* и *info_ptr* для реализации функций чтения и записи файла, а также массив для доступа к каждому пикселю изображения.

2.2. Считывание PNG файла.

Для считывания файла формата PNG реализована функция *read_png_image()*, принимающая на вход файл и указатель на объект структуры *Png*. Открываем двоичный файл для чтения и считываем подпись PNG файла. При этом, если файл открыть не удалось, а также если переданный файл не PNG, то выводятся сообщения об ошибке. Далее создаются и инициализируются структуры для чтения изображения. Если это не удается, выводятся сообщения об ошибке. Далее устанавливается файл для чтения, считываются первые 8 байт, а а также информация. После этого на основе полученных данных заполняются поля структуры *Png*: ширина, высота, тип цвета и глубина цвета.

Затем выделяется память для хранения изображения. Если возникают ошибки, они «отлавливаются» с помощью *setjmp()*. Затем с помощью функции *png_read_image()* в выделенную память считываются данные. Далее проверяется тип палитры(допустимы RGB или RGBA, иначе выводится сообщение об ошибке). В конце функции закрывается файл.

2.3. Запись PNG файла.

Для запись изображения формата PNG реализована функция *write_png_image()*, принимающая на вход файл и указатель на объект структуры *Png*. Открываем двоичный файл для записи. Если это не удалось, выводится сообщение об ошибке. Затем создаются и инициализируются структуры для

записи файла. Если это не удается, также выводятся сообщения об ошибках. Затем устанавливается файл для записи с помощью *png_init_io().* С помощью *setjmp()* «отлавливаются» возможные ошибки. Функция *png_set_IHDR()* устанавливает параметры изображения в структуры, а функция *png_write_info()* записывает эту информацию в файл. Функция *png_write_image()* записывает в файл карту пикселей. С помощью функции *png_write_end()* запись завершается. Память освобождается, и файл закрывается.

2.4. Функция рисования квадрата.

Данная функция *draw_square()* принимает в качестве аргументов указатель на объект структуры *Png* изображение, начальные координаты, размер квадрата, толщину контура, цвет контура, опцию заливки.

Происходит проверка: положительны ли координаты, размер квадрата, толщина контура, не лежат ли координаты за пределами изображения, не зайдет ли будущий квадрат за пределы изображения, корректно ли заданы диапазоны для компонент цвета.

Рисование контура происходит во вложенном цикле, где пиксели, которые соответствуют заданным условиям, то есть принадлежат толщине контура, закрашиваются.

Если выбрана опция заливки, то все, что находится внутри контура, закрашивается в введенный и проверенный на корректность цвет. Если опция выбрана некорректно, выводится сообщение об ошибке.

2.5. Функция смены фрагментов местами.

Данная функция *fragment_changing()* принимает на вход указатель на объект структуры *Png* изображение, объект структуры *Png* копию изображения, координаты верхнего левого угла, координаты нижнего правого угла и опцию: менять фрагменты по кругу или по диагонали.

В начале функции происходит проверка на корректность данных: положительные ли координаты, не лежат ли они за пределами, введены ли

координаты в правильном порядке. Затем создаются переменные, отвечающие за высоту и ширину одного фрагмента.

Если выбрала первая опция, то происходит обмен по кругу. С помощью изменения координат 1 фрагмент заменяется на четвертый: создаем указатель на первый фрагмент и указатель на четвертый фрагмент, для чего координаты высоты увеличим на высоту изображения, затем присвоим указателям на пиксели первого фрагмента новые значения. Аналогично заменяются четвертый на третий, третий на второй. Для того, чтобы заменить второй на первый, обратимся к копии изображения(так как в текущем изображении первый фрагмент уже изменен, а нужно получить исходный). Тогда из копии получаем указатели на пиксели исходного первого фрагмента и действуем аналогично.

Если выбрана вторая опция, то происходит обмен фрагментов по диагонали, что реализовано с помощью двух вложенных циклов: в первом меняются фрагменты на главной диагонали, во втором — на побочной. Создаются указатели на пиксели первого фрагмента, затем указатели на пиксели второго фрагмента — это те объекты, которые надо будет менять. Также создаются указатели еще раз на пиксели второго фрагмента и указатели на пиксели первого фрагмента из копии — это то, на что мы будем заменять. После этого значения первого фрагмента присваиваем значения второго фрагмента, а значениям второго фрагмента присваиваем значения первого фрагмента, которые мы получили из копии. Аналогично меняются фрагменты на побочной диагонали.

Если переданная опция была указана некорректно, об этом выводится сообщение.

2.6. Функция нахождения самого часто встречаемого цвета и его замены.

Данная функция color_changing() получает на вход указатель на объект структуры Png и компоненты цвета, на который будет происходить замена.

В начале функции происходит проверка на корректность диапазонов значений для компонент.

Для нахождения самого часто встречаемого цвета создается динамический трехмерный массив всех цветов размером 256 на 256 на 256. Затем во вложенном цикле с помощью прохождения по каждому пикселю изображения каждому конкретному цвету, при обращении по индексу к каждой из трех его компонент, добавляется единица, если он встречается. Таким образом, после выполнения массив цветом оказывается заполненным тем, сколько раз встретился каждый цвет картинки.

Переменной самого часто встречаемого цвета присваиваем значение первого элемента массива, обнуляем каждую компоненту и еще раз проходимся по всему массиву, находя, сколько раз встречается самый частый цвет и три его компоненты.

В еще одном вложенном цикле проходимся по картинке и, если пиксель совпадает по цвету с самым частым цветом, заменяем его на новый.

В конце функции освобождаем память, которая была выделена под динамический массив всех цветов.

2.7. Функция инверсии цвета в заданной области.

Данная функция *area_inversion()* получает в качестве аргументов указатель на объект структуры *Png*, координаты верхнего левого угла и координаты правого нижнего угла области.

В начале функции происходит проверка: положительны ли координаты, введены ли они в правильном порядке, не лежат ли они за пределами изображения.

После этого во вложенном цикле каждая компонента пикселя заменяется на разность 255 и значения данной компоненты. Таким образом происходит инверсия изображения.

2.8. Функции, выводящие на экран информацию о программе и файле.

Функция *print_help()* выводит на экран подсказку для пользователя о том, какие функции выполняет программа и с помощью каких ключей они выполняются.

Функция *print_info()* выводит на экран основную информацию про изображение: ширину, высоту, тип цвета и глубину цвета. В качестве аргумента данная функция получает указатель на объект структуры.

2.9. Реализация CLI.

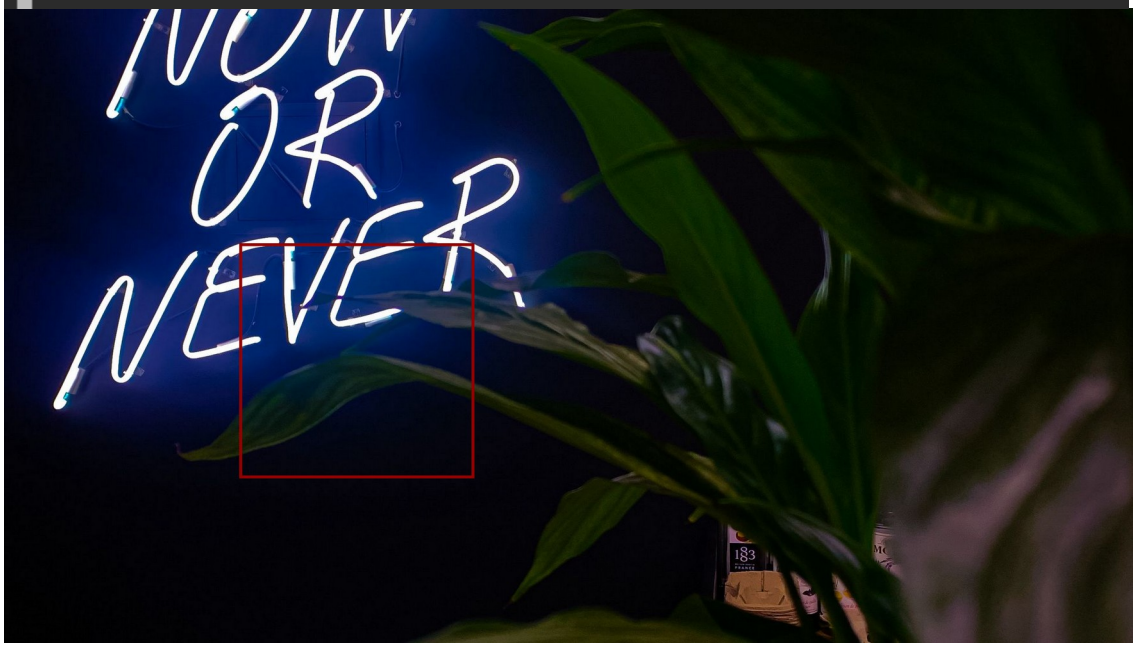
Для реализации CLI была использована библиотека *getopt.h* и функция данной библиотеки *getopt_long()* для того, чтобы была возможность работать и с длинными, и с короткими ключами. В статической структуре option указаны все ключи, предполагают ли они аргументы, длинная версия и короткая версия каждого.

Обработка входных параметров происходит в цикле *while* до тех пор, пока *getopt_long()* не вернет -1, что будет означать, что больше ключей нет. В операторе *switch*, где каждый из *case* отвечает за определенную функцию в зависимости от введенного ключа, каждый раз происходит проверка на количество введенных аргументов.

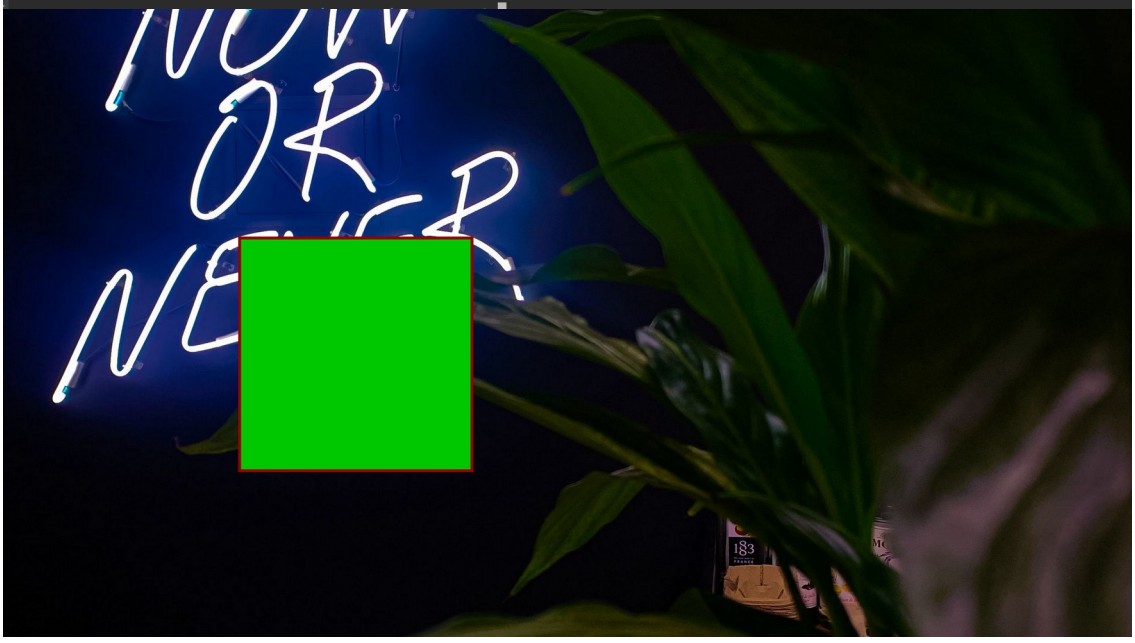
3. ТЕСТИРОВАНИЕ

3.1. Рисование квадрата.

```
gcc main.c -lpng && ./a.out test8.png -s 400,400,400,5,150,0,0,0 res8.png
```




```
oem@oem-Lenovo-IdeaPad-S145-15IKB:~/Desktop/coursework$ gcc main.c -lpng && ./a.out test8.png -s 400,400,400,5,150,0,0,1 res8.png
0,200,0
```



3.1. Смена фрагментов местами

```
$ gcc main.c -lpng && ./a.out test8.png -f 400,400,1000,1000,1 res8.png
```

```
gcc main.c -lpng && ./a.out test8.png -f 400,400,1000,1000,2 res8.png
```

3.3. Нахождение самого часто встречаемого цвета и его замена

Исходное изображение:

```
gcc main.c -lpng && ./a.out test2.png --color 0,0,180 res2.png
```

3.4. Инверсия цвета в заданной области

```
gcc main.c -lpng && ./a.out test8.png -a 300,300,600,700 res8.png
```

3.5. Вывод справки

```
oem@oem-Lenovo-IdeaPad-S145-15IKB:~/Desktop/coursework$ ./a.out -h
Описание функций, которые выполняет программа. Пожалуйста, передавайте аргументы через запятую!
-s --square, рисование квадрата. Нужно указать такие параметры, как:
1)координаты верхнего левого угла
2)размер стороны квадрата
3)толщина линий
4)цвет линий
5)осуществлять заливку или нет
6)если выбрана заливка, указать ее цвет

-f --fragment, поменять местами 4 куска области. Нужно указать такие параметры, как:
1)координаты верхнего левого угла области
2)координаты правого нижнего угла области
3)способ обмена частей: по кругу, по диагонали

-c --color, нахождение самого часто встречаемого цвета.
Указать, в какой цвет нужно перекрасить его.

-a --a_inversion, инверсия цвета в заданной области. Параметры:
1)координаты верхнего левого угла области
2)координаты нижнего правого угла области
-i --information, информация о файле.
-h --help, инструкция к программе.
```

3.6. Пример обработки ошибок

```
oem@oem-Lenovo-IdeaPad-S145-15IKB:~/Desktop/coursework$ ./a.out test8.png --color 300,100,100 res8.png
Данные заданы некорректно, недопустимый диапазон значений.
```

```
oem@oem-Lenovo-IdeaPad-S145-15IKB:~/Desktop/coursework$ ./a.out test8.png -a -1,1,500,500 res8.png
Ошибка: координаты не могут иметь отрицательные значения
```

```
oem@oem-Lenovo-IdeaPad-S145-15IKB:~/Desktop/coursework$ ./a.out test8.png -c 0,0,0
Для записи результата не был передан файл с расширением png.
```

ЗАКЛЮЧЕНИЕ

Реализована программа на языке Си, которая получает на вход изображение формата PNG. Программа считывает файл формата PNG, выполняет различные функции по его обработке и записывает измененный файл на диск. Интерфейс программы осуществляется с использованием CLI.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. http://www.libpng.org/
2. https://refspecs.linuxfoundation.org/

ПРИЛОЖЕНИЕ А

НАЗВАНИЕ ПРИЛОЖЕНИЯ

Название файла: main.c

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


struct Png {
    unsigned int width, height;
    png_byte color_type; //беззнаковая однобайтовая переменная
    png_byte bit_depth;
    png_structp png_ptr; //объект типа картинка
    png_infop info_ptr;
    //int number_of_passes;
    png_bytep* row_pointers; //массив для доступа к каждому пикселю,
указатель на указатель на bytep
};

void print_help() {
    printf("Описание функций, которые выполняет программа. Пожалуйста,
передавайте аргументы через запятую!\n");
    printf("-s --square, рисование квадрата. Нужно указать такие
параметры, как:\n"
           "1)координаты верхнего левого угла\n"
           "2)размер стороны квадрата\n"
           "3)толщина линий\n"
           "4)цвет линий\n"
           "5)осуществлять заливку или нет\n"
           "6)если выбрана заливка, указать ее цвет\n\n");
    printf("-f --fragment, поменять местами 4 куска области. Нужно
указать такие параметры, как:\n"
           "1)координаты верхнего левого угла области\n"
           "2)координаты правого нижнего угла области\n"
           "3)способ обмена частей: по кругу, по диагонали\n\n");
    printf("-c --color, нахождение самого часто встречаемого цвета.\n"
           "Указать, в какой цвет нужно перекрасить его.\n\n");
    printf("-a --a_inversion, инверсия цвета в заданной области.
Параметры:\n"
           "1)координаты верхнего левого угла области\n"
           "2)координаты нижнего правого угла области\n");
    printf("-i --information, информация о файле.\n");
    printf("-h --help, инструкция к программе.\n");
}

void print_info(struct Png* image) {
    printf("Ширина изображения в пикселях: %d, "
           "высота изображения в пикселях: %d\n", image->height, image-
>width);
    printf("Тип цвета: %u\n", image->color_type);
```

```
    printf("Глубина цвета: %u\n", image->bit_depth);
}

int read_png_file(char* file_name, struct Png* image) {
    char header[8];
    FILE* file = fopen(file_name, "rb"); //открывает двоичный файл для
чтения
    if (!file) {
        printf("Ошибка при чтении файла\n");
        return 0;
    }
    fread(header, 1, 8, file);
    if (png_sig_cmp(header, 0, 8)) {
        printf("Ошибка: формат не PNG\n");
        return 0;
    }

    //Инициализация, создание структур для чтения изображения
    image->png_ptr = png_create_read_struct(PNG_LIBPNG_VER_STRING, NULL,
NULL, NULL);
    if (image->png_ptr == NULL) {
        printf("Ошибка: не удалось создать структуру\n");
        return 0;
    }
    image->info_ptr = png_create_info_struct(image->png_ptr);
    if (image->info_ptr == NULL) {
        printf("Ошибка: не удалось создать структуру, в которой хранится"
               "информация об изображении PNG\n");
        return 0;
    }
    if (setjmp(png_jmpbuf(image->png_ptr))) {   //проверка ошибок при
использовании библиотеки libpng
        printf("Ошибка инициализации\n");
        return 0;
    }
    png_init_io(image->png_ptr, file); //устанавливает файл для чтения
    png_set_sig_bytes(image->png_ptr, 8); //считываем первые 8 байт
    png_read_info(image->png_ptr, image->info_ptr); //чтение информации

    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);

    png_read_update_info(image->png_ptr, image->info_ptr); //выделение
памяти дл хранения изображения

    if (setjmp(png_jmpbuf(image->png_ptr))){
        printf("Ошибка чтения изображения");
        return 0;
    }

    image->row_pointers = (png_bytep*)malloc(image-
>height*sizeof(png_bytep)); //выделение памяти
    for (int y = 0; y < image->height; y++) {
```

```
        image->row_pointers[y] =
(png_byte*)malloc(png_get_rowbytes(image->png_ptr, image->info_ptr));
//выделение памяти для каждой строки пикселей, функция возвращает
количество байт, необходимых для хранения строки

    }
    png_read_image(image->png_ptr, image->row_pointers); //считывает
данные изобображения в выделенную память

    if (png_get_color_type(image->png_ptr, image->info_ptr) !=
PNG_COLOR_TYPE_RGBA &&
    png_get_color_type(image->png_ptr, image->info_ptr) !=
PNG_COLOR_TYPE_RGB) {
        printf("Неподдерживаемый тип палитры\n");
        return 0;
    }
    fclose(file);
    return 1;
}

void write_png_file(char *file_name, struct Png *image) {
    if (file_name[strlen(file_name)-1] != 'g' &&
file_name[strlen(file_name)-2] != 'n' &&
        file_name[strlen(file_name)-3] != 'p' &&
file_name[strlen(file_name)-4] != '.') {
        printf("Для записи результата не был передан файл с расширением
png.\n");
        return;
    }
    FILE* file = fopen(file_name, "wb"); // создает двоичный файл для
записи
    if (!file) {
        printf("Ошибка при создании файла\n");
        return;
    }

    //инициализация структур для записив файл
    image->png_ptr = png_create_write_struct(PNG_LIBPNG_VER_STRING, NULL,
NULL, NULL);
    if (image->png_ptr == NULL) {
        printf("Ошибка: не удалось создать структуру\n");
        return;
    }
    image->info_ptr = png_create_info_struct(image->png_ptr);
    if (image->info_ptr == NULL) {
        printf("Ошибка: не удалось создать структуру, в которой хранится"
               "информация об изображении PNG\n");
        return;
    }
    if (setjmp(png_jmpbuf(image->png_ptr))) {   //проверка ошибок при
использовании библиотеки libpng
        printf("Ошибка инициализации\n");
        return;
    }

    png_init_io(image->png_ptr, file); //устанавливает файл для записи
    if (setjmp(png_jmpbuf(image->png_ptr))){
        printf("Ошибка при записи заголовка\n");
```

```
        return;
    }

    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))){
        printf("Ошибка при записи данных изображения\n");
        return;
    }
    png_write_image(image->png_ptr, image->row_pointers); //запись карты
пикселей

    if (setjmp(png_jmpbuf(image->png_ptr))){
        printf("Ошибка при окончании записи\n");
        return;
    }
    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(file);
}

void draw_square(struct Png* image, int x0, int y0, int a, int t, int
r_c,
        int g_c, int b_c, int filling) {
    if (x0 < 0 || y0 < 0 || a < 0 || t < 0)  {
        printf("Недопустимые параметры:\n "
               "координаты, размер квадрата и ширина контура не могут "
               "иметь отрицательные значения\n");
        return;
    }
    if (x0 > image->width || y0 > image->height) {
        printf("Недопустимые параметры:\n "
               "координаты не могут лежать за пределами изображения\n");
        return;
    }
    if ((x0 + a) >= image->width || (y0 + a) >= image->height) {
        printf("Квадрат не может заходить за пределы изображения\n");
        return;
    }
    if (r_c > 255 || r_c < 0 || g_c > 255 || g_c < 0 || b_c > 255 || b_c
< 0) {
        printf("Некорректно заданные диапазоны.\n");
        return;
    }
    for (int y = y0; y < y0 + a; y++) {
        png_bytep row1 = image->row_pointers[y];
        for (int x = x0; x < x0 + a; x++) {
            png_bytep ptr1 = &(row1[x * 4]);
            if ((x - x0) < t || (y - y0) < t || ((x0 + a) - x) <= t ||
((y0 + a) - y) <= t) {
                ptr1[0] = r_c;
                ptr1[1] = g_c;
```

```
                ptr1[2] = b_c;
                ptr1[3] = 255;
            }
        }
    }
    if (filling == 1) {
        int r_f, g_f, b_f;
        scanf("%d,%d,%d", &r_f, &g_f, &b_f);
        if (r_f > 255 || r_f < 0 || g_f > 255 || g_f < 0 || b_f > 255 ||
b_f < 0 ) {
            printf("Некорректно заданные диапазоны.\n");
            return;
        }
        for (int y = y0 + t; y < y0 + a - t; y++) {
            png_bytep row2 = image->row_pointers[y];
            for (int x = x0 + t; x < x0 + a - t; x++) {
                png_bytep ptr2 = &(row2[x*4]);
                ptr2[0] = r_f;
                ptr2[1] = g_f;
                ptr2[2] = b_f;
                ptr2[3] = 255;
            }
        }
    }
    else if (filling != 0){
        printf("Опция заливки выбрана некорректно.\n");
        return;
    }
}

void fragment_changing(struct Png* image, struct Png* copy, int x1, int
y1, int x2, int y2, int option) {
    int x, y;
    if (x1 < 0 ||  y1 < 0 || x2 < 0 || y2 < 0) {
        printf("Ошибка: координаты не могут иметь отрицательные значений\
n");
        return;
    }
    if (x1 > image->width || y1 > image->height || x2 > image->width ||
y2 > image->height) {
        printf("Ошибка: координаты не должны быть за пределами
изображения.\n");
        return;
    }
    if (x1 > x2 || y1 > y2) {
        printf("Ошибка: координаты верхнего левого угла должны быть
меньше координат"
               "нижнего правого угла.\n");
        return;
    }
    int w_f = (x2 - x1)/2;
    int h_f = (y2 - y1)/2;
    if (option == 1) {
        for (y = y1; y < y1 + h_f; y++) {
            png_bytep row = image->row_pointers[y];
            png_bytep row_change = image->row_pointers[y+h_f];
            for (x = x1; x < x1 + w_f; x++) {
                png_bytep ptr = &(row[x*4]);
```

```
                png_bytep ptr_change = &(row_change[x*4]);
                ptr[0] = ptr_change[0];
                ptr[1] = ptr_change[1];
                ptr[2] = ptr_change[2];
                ptr[3] = ptr_change[3];
            }

        }
        for (y = y1 + h_f; y < y2; y++) {
            png_bytep row = image->row_pointers[y];
            png_bytep row_change = image->row_pointers[y];
            for (x = x1; x < x1 + w_f; x++) {
                png_bytep ptr = &(row[x*4]);
                png_bytep ptr_change = &(row_change[(x+w_f)*4]);
                ptr[0] = ptr_change[0];
                ptr[1] = ptr_change[1];
                ptr[2] = ptr_change[2];
                ptr[3] = ptr_change[3];

            }
        }
        for (y = y1 + h_f; y < y2; y++) {
            png_bytep row = image->row_pointers[y];
            png_bytep row_change = image->row_pointers[y-h_f];
            for (x = x1 + w_f; x < x2; x++) {
                png_bytep ptr = &(row[x*4]);
                png_bytep ptr_change = &(row_change[x*4]);
                ptr[0] = ptr_change[0];
                ptr[1] = ptr_change[1];
                ptr[2] = ptr_change[2];
                ptr[3] = ptr_change[3];
            }
        }
        for (y = y1; y < y1 + h_f; y++) {
            png_bytep row = image->row_pointers[y];
            png_bytep row_change = copy->row_pointers[y];
            for (x = x1 + w_f; x < x2; x++) {
                png_bytep ptr = &(row[x*4]);
                png_bytep ptr_change = &(row_change[(x-w_f)*4]);
                ptr[0] = ptr_change[0];
                ptr[1] = ptr_change[1];
                ptr[2] = ptr_change[2];
                ptr[3] = ptr_change[3];
            }
        }
    }
    else if (option == 2) {
        for (y = y1; y < y1 + h_f; y++) {
            png_bytep row1 = image->row_pointers[y];
            png_bytep row2 = image->row_pointers[y+h_f];
            png_bytep row_change1 = copy->row_pointers[y];
            png_bytep row_change2 = image->row_pointers[y+h_f];
            for (x = x1; x < x1 + w_f; x++) {
                png_bytep ptr1 = &(row1[x*4]);
                png_bytep ptr_change1 = &(row_change2[(x+w_f)*4]);
                ptr1[0] = ptr_change1[0];
                ptr1[1] = ptr_change1[1];
                ptr1[2] = ptr_change1[2];
```

```
                ptr1[3] = ptr_change1[3];
                png_bytep ptr2 = &(row2[(x+w_f)*4]);
                png_bytep ptr_change2 = &(row_change1[x*4]);
                ptr2[0] = ptr_change2[0];
                ptr2[1] = ptr_change2[1];
                ptr2[2] = ptr_change2[2];
                ptr2[3] = ptr_change2[3];
            }
        }
        for (y = y1; y < y1 + h_f; y++) {
            png_bytep row1 = image->row_pointers[y];
            png_bytep row2 = image->row_pointers[y + h_f];
            png_bytep row_change1 = copy->row_pointers[y];
            png_bytep row_change2 = image->row_pointers[y + h_f];
            for (x = x1 + w_f; x < x2; x++) {
                png_bytep ptr1 = &(row1[x * 4]);
                png_bytep ptr_change1 = &(row_change2[(x - w_f) * 4]);
                ptr1[0] = ptr_change1[0];
                ptr1[1] = ptr_change1[1];
                ptr1[2] = ptr_change1[2];
                ptr1[3] = ptr_change1[3];
                png_bytep ptr2 = &(row2[(x - w_f) * 4]);
                png_bytep ptr_change2 = &(row_change1[x * 4]);
                ptr2[0] = ptr_change2[0];
                ptr2[1] = ptr_change2[1];
                ptr2[2] = ptr_change2[2];
                ptr2[3] = ptr_change2[3];
            }
        }
    }
    else {
        printf("Опция перестановки фрагментов введена некоррректно: "
               "может принимать только значения 1 или 2\n");
        return;
    }
}

void color_changing(struct Png* image, int r_new, int g_new, int b_new) {
    if (r_new > 255 || r_new < 0 || g_new > 255 || g_new < 0 || b_new >
255 || b_new < 0) {
        printf("Данные заданы некорректно, недопустимый диапазон
значений.\n");
        return;
    }
    int ***array_colors = calloc(256, sizeof(int**));
    {
        for (int y = 0; y < 256; y++) {
            array_colors[y] = calloc(256, sizeof(int*));
            for (int x = 0; x < 256; x++) {
                array_colors[y][x] = calloc(256, sizeof(int));
            }
        }
    }
    for (int y = 0; y < image->height; y++) {
        png_bytep row = image->row_pointers[y];
        for (int x = 0; x < image->width; x++) {
            png_bytep ptr = &(row[x*4]);
            array_colors[ptr[0]][ptr[1]][ptr[2]]++;
```

```
        }
    }
    int max_freq = array_colors[0][0][0];
    int max_r = 0, max_g = 0, max_b = 0;
    for (int y = 0; y < 256; y++) {
        for (int x = 0; x < 256; x++) {
            for (int z = 0; z < 256; z++) {
                if (array_colors[y][x][z] > max_freq) {
                    max_r = y;
                    max_g = x;
                    max_b = z;
                    max_freq = array_colors[y][x][z];
                }
            }
        }
    }
    for (int y = 0; y < image->height; y++) {
        png_bytep row2 = image->row_pointers[y];
        for (int x = 0; x < image->width; x++) {
            png_bytep ptr2 = &(row2[x*4]);
            if (ptr2[0] == max_r && ptr2[1] == max_g && ptr2[2] == max_b)
{
                ptr2[0] = r_new;
                ptr2[1] = g_new;
                ptr2[2] = b_new;
            }
        }
    }
    for (int y = 0; y < 256; y++) {
        for (int x = 0; x < 256; x++) {
            free(array_colors[y][x]);
        }
    }
    for (int y = 0; y < 256; y++) {
        free(array_colors[y]);
    }
    free(array_colors);
}


void area_inversion(struct Png* image, int x1, int y1, int x2, int y2) {
    int x, y;
    if (x1 < 0 ||  y1 < 0 || x2 < 0 || y2 < 0) {
        printf("Ошибка: координаты не могут иметь отрицательные значения\
n");
        return;
    }
    if (x1 > image->width || y1 > image->height || x2 > image->width ||
y2 > image->height) {
        printf("Ошибка: координаты не должны быть за пределами
изображения.\n");
        return;
    }
    if (x1 > x2 || y1 > y2) {
        printf("Ошибка: координаты верхнего левого угла должны быть
меньше координат"
               "нижнего правого угла.\n");
        return;
```

```
    }
    for (y = y1; y < y2; y++) {
        png_bytep row = image->row_pointers[y];
        for (x = x1; x < x2; x++) {
            png_bytep ptr = &(row[x*4]);
            ptr[0] = 255 - ptr[0];
            ptr[1] = 255 - ptr[1];
            ptr[2] = 255 - ptr[2];
        }
    }
}

int main(int num_arg, char **argv) {
    static const char* optStr = "s:f:c:a:ih?";
    static const struct option longOpt[] = {
        {"square", required_argument, NULL, 's'},
        {"fragment", required_argument, NULL, 'f'},
        {"color", required_argument, NULL, 'c'},
        {"a_inversion", required_argument, NULL, 'a'},
        {"help", no_argument, NULL, 'h'},
        {"information", no_argument, NULL, 'i'},
        {NULL, no_argument, NULL, 0}
    };
    struct Png image;
    int longInd;
    int opt = getopt_long(num_arg, argv, optStr, longOpt, &longInd);
    if (opt == -1) {
        print_help();
        return 0;
    }
    while(opt!=-1) {
        switch (opt) {
            case 's': {
                if (read_png_file(argv[1], &image)) {
                    read_png_file(argv[1], &image);
                    int x0, y0, a, t, r_c, g_c, b_c, filling;
                    int count_arg = sscanf(optarg, "%d,%d,%d,%d,%d,%d,%d,
%d", &x0, &y0, &a, &t, &r_c,
                                                    &g_c, &b_c, &filling);
                    if (count_arg < 8) {
                        printf("Недостаточно аргументов, вы ввели не все
значения!\n");
                        return 0;
                    }
                    draw_square(&image, x0, y0, a, t, r_c, g_c, b_c,
filling);
                    write_png_file(argv[num_arg - 1], &image);
                }
                break;
            }
            case 'f': {
                if (read_png_file(argv[1], &image)) {
                    read_png_file(argv[1], &image);
                    struct Png copy;
                    read_png_file(argv[1], &copy);
                    int x1, y1, x2, y2, option;
                    int count_arg = sscanf(optarg, "%d,%d,%d,%d,%d", &x1,
&y1, &x2, &y2, &option);
```

```
                    if (count_arg < 5) {
                        printf("Недостаточно аргументов, вы ввели не все
значения!\n");
                        return 0;
                    }
                    fragment_changing(&image, &copy, x1, y1, x2, y2,
option);
                    write_png_file(argv[num_arg - 1], &image);
                }
                break;
            }
            case 'c': {
                if (read_png_file(argv[1], &image)) {
                    read_png_file(argv[1], &image);
                    int r_new, g_new, b_new;
                    int count_arg = sscanf(optarg, "%d,%d,%d", &r_new,
&g_new, &b_new);
                    printf("Вы здесь\n");
                    if (count_arg < 3) {
                        printf("Недостаточно аргументов, вы ввели не все
значения!\n");
                        return 0;
                    }
                    color_changing(&image, r_new, g_new, b_new);
                    write_png_file(argv[num_arg - 1], &image);
                }
                break;
            }
            case 'a': {
                if (read_png_file(argv[1], &image)) {
                    read_png_file(argv[1], &image);
                    int x1, y1, x2, y2;
                    int count_arg = sscanf(optarg, "%d,%d,%d,%d", &x1,
&y1, &x2, &y2);
                    if (count_arg < 4) {
                        printf("Недостаточно аргументов, вы ввели не все
значения!\n");
                        return 0;
                    }
                    area_inversion(&image, x1, y1, x2, y2);
                    write_png_file(argv[num_arg - 1], &image);
                }
                break;

            }
            case 'i': {
                print_info(&image);
                break;
            }
            case 'h':
            case '?': {
                print_help();
                return 0;
            }
            default: {
                printf("Опция не существует.\n");
            }
        }
```

```
        opt = getopt_long(num_arg, argv, optStr, longOpt, &longInd);
    }
    return 0;
}
```