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

Студентка гр. 2381 _____ Газукина Д.Д.

Преподаватель _____ Тиняков С.А.

Санкт-Петербург

2023

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студентка Газукина Д.Д.

Группа 2381

Тема работы: Обработка PNG-файла

Исходные данные:

Вариант 12

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

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

Предполагаемый объем пояснительной записки:
Не менее 15 страниц.

Дата выдачи задания: 19.03.2023

Дата сдачи реферата: 20.05.2023

Дата защиты реферата: 27.05.2023

Студентка гр. 2381 _____ Газукина Д.Д.

Преподаватель _____ Тиняков С.А.

АННОТАЦИЯ

Курсовая работа заключается в разработке программы на языке Си, обрабатывающей PNG-изображение в соответствии с командами пользователя. Для взаимодействия с пользователем был разработан консольный интерфейс. В случае неверных входных данных пользователю выводится сообщение об ошибках. При отсутствии ошибок новое изображение сохраняется в текущую директорию. В работе использовалась библиотека libpng, структуры и функции стандартной библиотеки языка.

SUMMARY

The course work is to develop a C program that processes a PNG image in accordance with user commands. A console interface was developed for user interaction. In case of incorrect input data, an error message is displayed to the user. If there are no errors, the new image is saved to the current directory. The work used the libpng library, structures and functions of the standard library of the language C.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель работы:

Разработать программу, обрабатывающую png-изображение согласно введенным пользователем командам, считывающимся с помощью библиотеки getopt.

Основные задачи:

- Реализовать функции для считывания и записи изображения, используя библиотеку libpng.
- Разработать под каждую из основных четырех команд отдельную функцию обработки изображения.
- Предоставить пользователю возможность выбора команды.
- С помощью getopt реализовать интерфейс командной строки.

1. СЧИТЫВАНИЕ И ЗАПИСЬ ИЗОБРАЖЕНИЯ

1.1. Считывание png-изображения

Разработана функция read_png_file, в которую передается название файла и указатель на структуру Png. В функции используются функции библиотеки libpng, которые записывают необходимую информацию в структуру изображения, ширину, высоту, тип цвета, глубину цвета, содержание пикселей. В случае возникновения ошибок на разных этапах используется функция setjmp и png_jmpbuf, информация об ошибке выводится на экран, а программа завершается. Ошибки могут возникать при открытии файла для чтения, чтении файла, записи информации о нем, создании структуры изображения, при неподходящей глубине цвета изображения и формата цвета.

1.2. Запись измененного png-изображения

Разработана функция write_png_file, в которую передается название нового файла и указатель на структуру Png. В функции используются функции библиотеки libpng, которые создают структуру изображения и записывают необходимую информацию о файле. Информация о возникающих ошибках выводится на экран, программа завершается.

2. ФУНКЦИИ ОБРАБОТКИ ИЗОБРАЖЕНИЯ

2.1. Рисование квадрата

Реализована функция draw_square, которая по заданной начальной координате, длине стороны, ширине контура, цвету контура и по цвету заливки, если она необходима, рисует поверх считанного изображения квадрат с помощью цикла в цикле for. Внутри функции производится проверка на корректность входных данных, они не могут быть меньше нуля, цвета должны быть не меньше 0 и не больше 255, квадрат не может выходить за пределы изображения.

2.2. Смена фрагментов изображения местами

Реализована функция swap_areas, на вход которой подается указатель на структуру Png, координаты левого верхнего и правого нижнего углов и тип передвижения: по кругу или по диагонали. Внутри производится проверка на ошибки в данных. Написаны две подфункции: save_area для сохранения фрагмента изображения, change_frag для замены пикселей фрагмента. Внутри основной функции вызывается функция save_area для первого (левый верхний) и третьего (левый нижний) фрагментов. Далее в зависимости от выбранного типа фрагменты меняются местами. Освобождается память, выделенная под сохраненные фрагменты.

2.3. Замена самого часто встречающегося цвета

Реализована функция change_color, которая находит цвет, который чаще всего встречается на изображении, без учета альфа-канала. Внутри производится проверка на ошибки в данных. Создается трехмерный массив 256*256*256 colors, в котором хранится количество каждого из цветов изображения. Когда находится максимально встречающийся цвет, в цикле каждый из пикселей сравнивается с ним и заменяется в случае совпадения на новый цвет. Память, выделенная под colors, очищается.

2.4. Инвертирование области

Реализована функция invert_colors, на вход которой подается указатель на структуру Png и координаты левого верхнего и правого нижнего углов.

Производится проверка на корректность данных, текст ошибок выводится на экран. В цикле каждый из пикселей нужной области заменяется на новый цвет, равный 255 минус число старого канала r, g, b. Прозрачность, то есть альфа-канал, остается неизменной.

3. ВСПОМОГАТЕЛЬНЫЕ ФУНКЦИИ

3.1. Вывод информации

Функция print_help() вызывается при вводе ключа help/--help/-h. Выводится информация о программе, возможные программы и образец ввода данных в командную строку.

Функция print_PNG_info(struct Png *image) при вводе ключа info/--info/-i печатает информацию об изображении: ширина, высота, тип цвета, глубина цвета. Для этого происходит обращение к структуре Png.

4. ОБРАБОТКА АРГУМЕНТОВ КОМАНДНОЙ СТРОКИ

4.1. Считывание опций и их аргументов

Используется библиотека getopt для реализации CLI. Разработана функция all_keys, на вход которой передаются: количество аргументов командной строки, список аргументов, а также указатели на все возможные аргументы, которые будут использоваться в функциях обработки изображения. Создается массив struct option long_opts[] со всеми опциями, далее перебираются аргументы командной строки с помощью функции getopt_long. При совпадении аргументов с опциями переменным присваиваются значения аргументов, следующего за опцией -/--.

4.2. Выбор команды

В функции main с помощью if-else третий (второй) аргумент сравнивается с возможными командами: square, swap, often, inversion. Далее при совпадении происходит проверка на возможные ошибки во вводе данных, выводится сообщение об ошибке, программа завершается. Если ошибок нет, вызывается нужная функция. Новое изображение записывается в текущую директорию.

5. БИБЛИОТЕКИ И ОСНОВНАЯ ФУНКЦИЯ

5.1. Использующиеся библиотеки

В программе подключаются заголовочные файлы stdio.h, unistd.h, getopt.h, stdlib.h, string.h, ctype.h, png.h.

5.2. Главная функция main

В функции main происходит проверка на количество аргументов командной строки. Если их меньше трех, выводится справка. Далее создается структура Png для изображения. Вызывается функция чтения файла, далее считывается основная команда choice. Название нового файла записывается в строку new_file_name. Вызывается функция all_keys, далее в зависимости от выбранной пользователем команды изображение обрабатывается, новый файл записывается в директорию. Память, выделенная под строки, освобождается. Программа завершается.

ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы реализованы функции для обработки png-изображения и консольный интерфейс. Программа создает новое png-изображение согласно полученным от пользователя командам: рисует квадрат, меняет местами фрагменты изображения, заменяет самый частый цвет на новый, инвертирует цвет. В программе используются библиотеки getopt и libpng. Можно сделать вывод о соответствии полученного результата поставленной цели.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Брайан Керниган, Деннис Ритчи — Язык программирования Си.

2. Описание функций языка Си // Все о Hi-Tech. URL: http://all-ht.ru/inf/prog/c/func/index.html (дата обращения: 11.05.2023)

3. Курс "Программирование на Си. Практические задания. Второй семестр" // moevm. URL: https://e.moevm.info/course/view.php?id=18 (дата обращения: 11.05.2023)

4. Онлайн-справочник // cplusplus.com. URL: https://cplusplus.com/ (дата обращения: 11.05.2023)

5. Информация о getopt // gnu.org. URL: https://www.gnu.org/software/libc/manual/html_node/Getopt.html (дата обращения: 11.05.2023)

6. Информация о библиотеке libpng // PNG Documentation. URL: http://www.libpng.org/pub/png/pngdocs.html (дата обращения: 11.05.2023)

ПРИЛОЖЕНИЕ А

ПРИМЕРЫ РАБОТЫ ПРОГРАММЫ

Пример 1. Аргументы командной строки: ./pngedit image_to_square.png square -s 100.100 -l 700 -t 50 -c 100.70.180.255 -f 0 image1.png

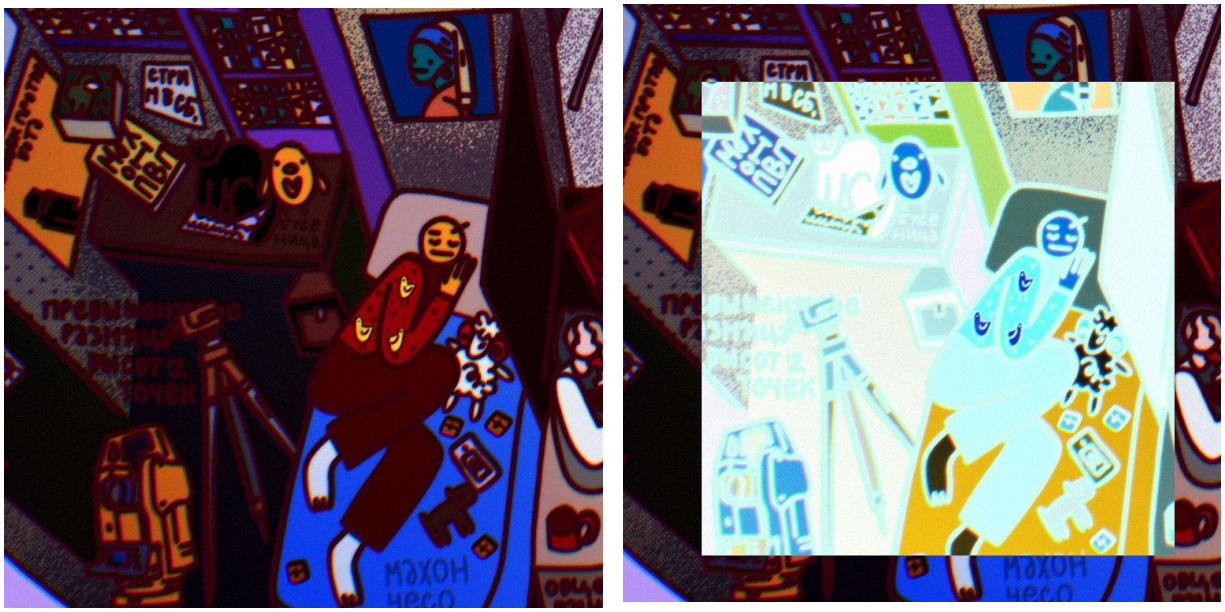
Пример 2. Аргументы командной строки: ./pngedit image_to_square.png square -s 100.100 -l 700 -t 100 -c 100.70.180.255 -f 1 -r 120.90.200.255 image2.png


Пример 3. Аргументы командной строки:

./pngedit image_to_swap.png swap -s 100.100 -e 500.500 -p circle image3.png


Пример 4. Аргументы командной строки:

./pngedit image_to_change_color.png often -c 255.0.0.255 image4.png



Пример 5. Аргументы командной строки:

./pngedit image_to_invert.png inversion -s 100.100 -e 700.700 image5.png

```
daria@MacBook-Air-Dara CURRENT % ./pngedit -h

Это программа с CLI для редактирования png-изображений!
Поддерживаются файлы с глубиной цвета 8 бит, RGBa
Формат ввода: [./pngedit] [command] [in.png] -[o]/--[option] [argument] [out.png]

Доступные команды:
[имя файла] square - нарисовать квадрат
    -s/--start      [x-координата].[y-координата] - левый верхний угол
    -l/--length     [число] - длина стороны
    -t/--thickness  [число] - толщина линий (в пикселях)
    -c/--color      [R].[G].[B].[A] - числа от 0 до 255, цвет линий
    -f/--fill       [число] - опция заливки, по умолчанию без неё (1 - заливка, 0 - нет)
    -r/--colorfill  [R].[G].[B].[A] - числа от 0 до 255, цвет заливки

[имя файла] swap - поменять местами 4 куска области
    -s/--start      [x1-координата].[y1-координата] - левый верхний угол
    -e/--end        [x2-координата].[y2-координата] - правый нижний угол
    -p/-type        [circle / diagonal] - способ (по кругу / по диагонали)

[имя файла] often - заменить самый часто встречающийся цвет на новый
    -c/--color      [R].[G].[B].[A] - числа от 0 до 255, новый цвет (RGBa)

[имя файла] inversion - инвертировать цвет в заданной области
    -s/--start      [x1-координата].[y1-координата] - левый верхний угол
    -e/--end        [x2-координата].[y2-координата] - правый нижний угол

[имя файла] -i/--info/info - получить информацию об изображении
[имя файла] -h/--help/help - вызвать справку
```

Пример 7. Вывод справочной информации.

ПРИЛОЖЕНИЕ Б
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.c

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

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

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

void print_PNG_info(struct Png *image){
    printf("Ширина изображения: %d\n", image->width);
    printf("Высота изображения: %d\n", image->height);
    printf("Тип цвета: %u\n", image->color_type);
    printf("Глубина цвета: %u\n", image->bit_depth);
}

void print_help(){
    printf("\nЭто программа с CLI для редактирования png-изображений! \n"
           "Поддерживаются файлы с глубиной цвета 8 бит, RGBa\n");
    printf("Формат ввода:\033[1;35m [./pngedit] [command] [in.png] -
[o]/--[option] [argument] [out.png]\033[0m\n\n");
    printf("Доступные команды:\n");

    printf("[имя файла] \033[1;35msquare\033[0m - нарисовать квадрат\n");
    printf("    -s/--start      [x-координата].[y-координата] - левый
верхний угол\n");
    printf("    -l/--length     [число] - длина стороны\n");
    printf("    -t/--thickness  [число] - толщина линий (в пикселях)\n");
    printf("    -c/--color      [R].[G].[B].[A] - числа от 0 до 255, цвет
линий\n");
    printf("    -f/--fill       [число] - опция заливки, по умолчанию без
неё (1 - заливка, 0 - нет)\n");
    printf("    -r/--colorfill  [R].[G].[B].[A] - числа от 0 до 255, цвет
заливки\n\n");

    printf("[имя файла] \033[1;35mswap\033[0m - поменять местами 4 куска
области\n");
    printf("    -s/--start      [x1-координата].[y1-координата] - левый
верхний угол\n");
    printf("    -e/--end        [x2-координата].[y2-координата] - правый
нижний угол\n");
    printf("    -p/--type        [circle / diagonal] - способ (по кругу /
по диагонали)\n\n");
```

```
    printf("[имя файла] \033[1;35moften\033[0m - заменить самый часто
встречающийся цвет на новый\n");
    printf("    -c/--color      [R].[G].[B].[A] - числа от 0 до 255,
новый цвет (RGBa)\n\n");

    printf("[имя файла] \033[1;35minversion\033[0m - инвертировать цвет в
заданной области\n");
    printf("    -s/--start      [x1-координата].[y1-координата] - левый
верхний угол\n");
    printf("    -e/--end        [x2-координата].[y2-координата] - правый
нижний угол\n\n");

    printf("[имя файла] \033[1;35m-i/--info/info\033[0m - получить
информацию об изображении\n");
    printf("[имя файла] \033[1;35m-h/--help/help\033[0m - вызвать
справку\n\n");
}

void read_png_file(char * file_name, struct Png * image) {
    char header[8];
    FILE *fp = fopen(file_name, "rb");
    if (!fp){
        printf("Ошибка: не удалось открыть файл для чтения. Введите
название файла с расширением '.png'\n");
        exit(1);
    }

    fread(header, 1, 8, fp);
    if (png_sig_cmp((const unsigned char *)header, 0, 8)){
        printf("Ошибка: формат изображения не PNG.\n");
        exit(1);
    }

    image->png_ptr = png_create_read_struct(PNG_LIBPNG_VER_STRING, NULL,
NULL, NULL);
    if (!image->png_ptr){
        printf("Ошибка: не удалось создать структуру изображения.\n");
        exit(1);
    }

    image->info_ptr = png_create_info_struct(image->png_ptr);
    if (!image->info_ptr){
        printf("Ошибка: не удалось создать структуру с информацией об
изображении.\n");
        exit(1);
    }

    if (setjmp(png_jmpbuf(image->png_ptr))){
        printf("Ошибка инициализации.\n");
        exit(1);
    }

    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);

    if (image->bit_depth != 8) {
        printf("Ошибка: поддерживается только 8-битная глубина цвета\n");
        exit(1);
    }

    png_read_update_info(image->png_ptr, image->info_ptr);
    if (setjmp(png_jmpbuf(image->png_ptr))){
        printf("Ошибка чтения изображения.\n");
        exit(1);
    }

    if (png_get_color_type(image->png_ptr, image->info_ptr) !=
PNG_COLOR_TYPE_RGB_ALPHA){
        printf("Поддерживается только тип цвета RGBa.\n");
        exit(1);
    }

    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) {
    if (strstr(file_name, ".png") != &(file_name[strlen(file_name)-4])) {
        printf("Ошибка: не передан аргумент для итогового изображения в
расширении '.png'.\n");
        exit(1);
    }
    FILE *fp = fopen(file_name, "wb");
    if (!fp){
        printf("Ошибка при создании файла итогового изображения.'\n");
        exit(1);
    }

    image->png_ptr = png_create_write_struct(PNG_LIBPNG_VER_STRING, NULL,
NULL, NULL);
    if (!image->png_ptr){
        printf("Ошибка при создании структуры итогового изображения.'\
n");
        exit(1);
    }

    image->info_ptr = png_create_info_struct(image->png_ptr);
    if (!image->info_ptr){
        printf("Ошибка при создании структуры с информацией об итоговом
изображении.'\n");
```

```
        exit(1);
    }

    if (setjmp(png_jmpbuf(image->png_ptr))){
        printf("Ошибка инициализации\n");
        exit(1);
    }
    png_init_io(image->png_ptr, fp);

    if (setjmp(png_jmpbuf(image->png_ptr))){
        printf("Ошибка записи заголовка итогового файла.\n");
        exit(1);
    }

    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 записи данных итогового файла.\n");
        exit(1);
    }
    png_write_image(image->png_ptr, image->row_pointers);

    if (setjmp(png_jmpbuf(image->png_ptr))){
        printf("Ошибка при окончании записи итогового файла.\n");
        exit(1);
    }
    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 draw_square(struct Png * image, int x, int y, int l, int t, int *
color, int fill, int * colorF) {
    if (x < 0 || y < 0 || l < 0 || t < 0)  {
        printf("Введены некорретные данные: "
               "координаты, длина стороны квадрата и ширина линий "
               "не могут иметь отрицательные значения\n");
        return;
    }
    if (x < 0 ||  x >= image->width || y < 0 || y >= image->height) {
        printf("Введены некорретные данные: координаты должны "
               "находиться в пределах изображения и быть не меньше 0\n");
        return;
    }
    if ((x + l) >= image->width || (y + l) >= image->height) {
        printf("Введены некорретные данные: квадрат не может "
               "выходить за пределы изображения\n");
        return;
    }
```

```
    if (color[0] > 255 || color[0] < 0 || color[1] > 255 || color[1] < 0
    || color[2] > 255 || color[2] < 0 || color[3] > 255 || color[3] < 0)
{
        printf("Введены некорретные данные: цвета должны лежать от 0 до
255\n");
        return;
    }
    if (fill) {
        if (colorF[0] > 255 || colorF[0] < 0 || colorF[1] > 255 ||
colorF[1] < 0
            || colorF[2] > 255 || colorF[2] < 0 || colorF[3] > 255 ||
colorF[3] < 0) {
            printf("Введены некорретные данные: цвета должны лежать от 0
до 255\n");
            return;
        }
    }

    int x1 = x;
    int y1 = y;
    int x2 = x + l - 1;
    int y2 = y + l - 1;

    int number_of_channels = 4;
    int bit_depth = image->bit_depth;
    int stride = number_of_channels * bit_depth / 8;

    for (int i = x1; i <= x2; i++) {
        for (int j = y1; j <= y2; j++) {
            for (int c = 0; c < 4; c++) {
                if ((j >= y1 && j < y1+t) || (j <= y2 && j > y2-t)
                    || (i >= x1 && i < x1+t)  || (i <= x2 && i > x2-t)) {
                    // рисование границ
                    image->row_pointers[j][i * stride + c] = color[c];
                } else if (fill && (i >= x1+t && i <= x2-t) && (j >= y1+t
&& j <= y2-t)) {
                    // заливка
                    image->row_pointers[j][i * stride + c] = colorF[c];
                }
            }
        }
    }

}

void change_frag(struct Png * image, int to_x1, int to_x2, int to_y1, int
to_y2, int h, int w, png_bytep ** save) {
    int number_of_channels = 4;
    int bit_depth = image->bit_depth;
    int stride = number_of_channels * bit_depth / 8;

    if (!save) {
        for (int y = to_y1; y <= to_y2; y++) {
            png_byte *row_to = image->row_pointers[y];
            png_byte *row_from = image->row_pointers[y + h];
            for (int x = to_x1; x <= to_x2; x++) {
                png_byte *ptr_to = &(row_to[x * stride]);
                png_byte *ptr_from = &(row_from[(x + w) * stride]);
```

```
                ptr_to[0] = ptr_from[0];
                ptr_to[1] = ptr_from[1];
                ptr_to[2] = ptr_from[2];
                ptr_to[3] = ptr_from[3];
            }
        }
    } else {
        for (int y = to_y1, y_save = 0; y <= to_y2 && y_save <= h; y++,
y_save++) {
            png_byte *row = image->row_pointers[y];
            for (int x = to_x1, x_save = 0; x <= to_x2 && x_save <= w; x+
+, x_save++) {
                png_byte *ptr2 = &(row[x * stride]);
                ptr2[0] = save[y_save][x_save][0];
                ptr2[1] = save[y_save][x_save][1];
                ptr2[2] = save[y_save][x_save][2];
                ptr2[3] = save[y_save][x_save][3];
            }
        }
    }


}

png_bytep ** save_area(struct Png * image, int x1, int x2, int y1, int
y2) {
    int number_of_channels = 4;
    int bit_depth = image->bit_depth;
    int stride = number_of_channels * bit_depth / 8;

    png_bytep ** save = calloc(y2-y1+1, sizeof(png_bytep *));
    for (int i = 0; i < y2-y1+1; i++) {
        save[i] = (png_bytep*) calloc(x2-x1+1, sizeof(png_bytep));
        for (int j = 0; j < x2-x1+1; j++) {
            save[i][j] = (png_bytep) calloc(4, sizeof(png_byte));
        }
    }
    for (int y = y1, y_s = 0; y <= y2; y++, y_s++) {
        png_byte *row = image->row_pointers[y];
        for (int x = x1, x_s = 0; x <= x2; x++, x_s++) {
            png_byte *ptr = &(row[x * stride]);
            save[y_s][x_s][0] = ptr[0];
            save[y_s][x_s][1] = ptr[1];
            save[y_s][x_s][2] = ptr[2];
            save[y_s][x_s][3] = ptr[3];
        }
    }
    return save;
}

void swap_areas(struct Png * image, int x1, int y1, int x2, int y2, char
* type) {
    if (x1 < 0 ||  x1 >= image->width || y1 < 0 || y1 >= image->height
        || x2 < 0 ||  x2 >= image->width || y2 < 0 || y2 >= image-
>height) {
        printf("Введены некорретные данные: координаты должны "
               "находиться в пределах изображения и быть не меньше нуля\
n");
```

```
        return;
    }
    if (x1 > x2 || y1 > y2) {
        printf("Введены некорретные данные: координаты верхнего левого
угла "
               "должны быть меньше координат нижнего правого угла.\n");
        return;
    }
    int number_of_channels = 4;
    int bit_depth = image->bit_depth;
    int stride = number_of_channels * bit_depth / 8;

    int h = y2 - y1 + 1, w = x2 - x1 + 1;
    if (w % 2) { x2 -= 1; w--; }
    if (h % 2) { y2 -= 1; h--; }
    int h_area = h / 2, w_area = w / 2;

    int area_1_x1 = x1,    area_1_x2 = x1 + w_area-1,    area_1_y1 = y1,
area_1_y2 = y1 + h_area-1;
    int area_2_x1 = area_1_x2+1,    area_2_x2 = area_2_x1 + w_area-1,
area_2_y1 = y1,    area_2_y2 = y1 + h_area-1;
    int area_3_x1 = x1,    area_3_x2 = area_1_x2,    area_3_y1 =
area_1_y2+1,    area_3_y2 = area_3_y1 + h_area-1;
    int area_4_x1 = area_3_x2+1,    area_4_x2 = area_4_x1 + w_area-1,
area_4_y1 = area_2_y2+1,    area_4_y2 = area_4_y1 + h_area-1;

    png_bytep ** save_pix_1 = save_area(image, area_1_x1, area_1_x2,
area_1_y1, area_1_y2); // save area 1
    png_bytep ** save_pix_3 = save_area(image, area_3_x1, area_3_x2,
area_3_y1, area_3_y2); // save area 3

    if (!strcasecmp(type, "circle")) {
        change_frag(image, area_1_x1, area_1_x2, area_1_y1, area_1_y2,
h_area, 0, NULL); // area 3 to area 1
        change_frag(image, area_3_x1, area_3_x2, area_3_y1, area_3_y2, 0,
w_area, NULL); // area 4 to area 3
        change_frag(image, area_4_x1, area_4_x2, area_4_y1, area_4_y2, 0-
h_area, 0, NULL); // area 2 to area 4
        change_frag(image, area_2_x1, area_2_x2, area_2_y1, area_2_y2,
h_area, w_area, save_pix_1); // area 1 (saved) to 2
    } else if (!strcasecmp(type, "diagonal")) {
        change_frag(image, area_1_x1, area_1_x2, area_1_y1, area_1_y2,
h_area, w_area, NULL); // area 3 to area 1
        change_frag(image, area_3_x1, area_3_x2, area_3_y1, area_3_y2, 0-
h_area, w_area, NULL); // area 3 to area 1
        change_frag(image, area_4_x1, area_4_x2, area_4_y1, area_4_y2,
h_area, w_area, save_pix_1); // area 1 (saved) to 4
        change_frag(image, area_2_x1, area_2_x2, area_2_y1, area_2_y2,
h_area, w_area, save_pix_3); // area 3 (saved) to 2
    }

    for (int i = 0; i < h_area; i++) {
        for (int j = 0; j < w_area; j++) {
            free(save_pix_1[i][j]);
            free(save_pix_3[i][j]);
        }
        free(save_pix_1[i]);
        free(save_pix_3[i]);
```

```
    }
    free(save_pix_1);
    free(save_pix_3);
}

void change_color(struct Png * image, int * new_color) {
    if (new_color[0] > 255 || new_color[0] < 0 || new_color[1] > 255 ||
new_color[1] < 0
        || new_color[2] > 255 || new_color[2] < 0 || new_color[3] > 255
|| new_color[3] < 0) {
        printf("Введены некорретные данные: цвета должны лежать от 0 до
255\n");
        return;
    }
    int number_of_channels = 4;
    int bit_depth = image->bit_depth;
    int stride = number_of_channels * bit_depth / 8;

    int *** colors = calloc(256, sizeof(int**));
    for (int i = 0; i < 256; i++) {
        colors[i] = calloc(256, sizeof(int*));
        for (int j = 0; j < 256; j++) {
            colors[i][j] = 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 * stride]);
            colors[ptr[0]][ptr[1]][ptr[2]]++;
        }
    }

    int freq = colors[0][0][0];
    int max_colors[] = {0,0,0};
    for (int i = 0; i < 256; i++) {
        for (int j = 0; j < 256; j++) {
            for (int k = 0; k < 256; k++) {
                if (colors[i][j][k] > freq) {
                    max_colors[0] = i;
                    max_colors[1] = j;
                    max_colors[2] = k;
                    freq = colors[i][j][k];
                }
            }
        }
    }

    for (int j = 0; j < image->height; j++) {
        png_bytep row = image->row_pointers[j];
        for (int i = 0; i < image->width; i++) {
            png_bytep ptr = &(row[i * stride]);
            if (ptr[0] == max_colors[0] && ptr[1] == max_colors[1] &&
ptr[2] == max_colors[2]) {
                ptr[0] = new_color[0];
                ptr[1] = new_color[1];
                ptr[2] = new_color[2];
```

```
                ptr[3] = new_color[3];
            }
        }
    }

    for (int i = 0; i < 256; i++) {
        for (int j = 0; j < 256; j++) {
            free(colors[i][j]);
        }
        free(colors[i]);
    }
    free(colors);
}

void invert_colors(struct Png * image, int x1, int y1, int x2, int y2) {
    if (x1 < 0 ||  x1 >= image->width || y1 < 0 || y1 >= image->height
    || x2 < 0 ||  x2 >= image->width || y2 < 0 || y2 >= image->height) {
        printf("Введены некорретные данные: координаты должны "
               "находиться в пределах изображения и быть не меньше нуля\
n");
        return;
    }
    if (x1 > x2 || y1 > y2) {
        printf("Введены некорретные данные: координаты верхнего левого
угла "
               "должны быть меньше координат нижнего правого угла.\n");
        return;
    }
    int number_of_channels = 4;
    int bit_depth = image->bit_depth;
    int stride = number_of_channels * bit_depth / 8;

    for (int i = x1; i <= x2; i++) {
        for (int j = y1; j <= y2; j++) {
            image->row_pointers[j][i * stride + 0] = 255 - image-
>row_pointers[j][i * stride + 0];
            image->row_pointers[j][i * stride + 1] = 255 - image-
>row_pointers[j][i * stride + 1];
            image->row_pointers[j][i * stride + 2] = 255 - image-
>row_pointers[j][i * stride + 2];
            image->row_pointers[j][i * stride + 3] = image-
>row_pointers[j][i * stride + 3];
        }
    }
}


void all_keys(int argc, char *argv[], int* length, int* thickness,
              int* fill, int* x1, int* y1, int* x2, int* y2, int** color,
              int** color_fill, char** swap_type, struct Png* image);

int main(int argc, char **argv) {
    if(argc == 1 || argc == 2){
        print_help();
        return 0;
    }
    struct Png image;
    read_png_file(argv[1], &image);
```

```
    char * choice = malloc(10 * sizeof(char));
    strcpy(choice, argv[2]);

    char * new_file_name = malloc(1024 * sizeof(char));
    strcpy(new_file_name, argv[argc-1]);

    int length = -1; int thickness = -1;
    int fill = 0; int x1 = -1; int y1 = -1; int x2 = -1; int y2 = -1;
    int * color = calloc(4, sizeof(int)); color[0] = -1; color[1] = -1;
color[2] = -1; color[3] = -1;
    int * color_fill = calloc(4, sizeof(int)); color_fill[0] = -1;
color_fill[1] = -1; color_fill[2] = -1; color_fill[3] = -1;
    char * swap_type = malloc(10 * sizeof(char)); strcpy(swap_type,
"none");

    all_keys(argc, argv, &length, &thickness, &fill, &x1, &y1, &x2, &y2,
&color, &color_fill, &swap_type, &image);

    if (!strcasecmp(choice, "square")) {
        if ((x1 == -1 || y1 == -1 || length == -1 || thickness == -1 ||
        color[0] == -1 || color[1] == -1 || color[2] == -1 || color[3] ==
-1) ||
        (fill == 1 && (color_fill[0] == -1 || color_fill[1] == -1 ||
color_fill[2] == -1 || color_fill[3] == -1))){
            printf("Ошибки во вводе параметров. \n"
                   "Введите параметры -s x.y -l len -t width -c r.g.b.a -
f 0/1 -r r.g.b.a \n");
            return 0;
        }
        if (fill) draw_square(&image, x1, y1, length, thickness, color,
fill, color_fill);
        else draw_square(&image, x1, y1, length, thickness, color, fill,
NULL);
        write_png_file(new_file_name, &image);
    }
    else if (!strcasecmp(choice, "swap")) {
        if ((x1 == -1 || y1 == -1 || x2 == -1 || y2 == -1) || !
strcasecmp(swap_type, "none")) {
            printf("Неправильно введены координаты или тип смены местами
областей.\n"
                   "Введите параметры -s x.y -e x.y -p circle/diagonal \
n");
            return 0;
        }
        swap_areas(&image, x1, y1, x2, y2, swap_type);
        write_png_file(new_file_name, &image);
    }
    else if (!strcasecmp(choice, "often")){
        if (color[0] == -1 || color[1] == -1 || color[2] == -1 ||
color[3] == -1) {
            printf("Неправильно введён новый цвет.\n"
                   "Введите параметр -c r.g.b.a \n");
            return 0;
        }
        change_color(&image, color);
        write_png_file(new_file_name, &image);
    }
```

```
    else if (!strcasecmp(choice, "inversion")) {
        if ((x1 == -1 || y1 == -1 || x2 == -1 || y2 == -1)) {
            printf("Неправильно введены координаты.\n"
                   "Введите параметры -s x.y -e x.y \n");
            return 0;
        }
        invert_colors(&image, x1, y1, x2, y2);
        write_png_file(new_file_name, &image);
    }
    else if (!strcasecmp(choice, "info")){
        print_PNG_info(&image);
    }
    else if (!strcasecmp(choice, "help")){
        print_help();
    }
    else if (strcasecmp(choice, "help") && strcasecmp(choice, "-h") &&
strcasecmp(choice, "--help") && strcasecmp(choice, "info") &&
strcasecmp(choice, "-i") && strcasecmp(choice, "--info")) {
        printf("Неизвестное название опции.\n");
    }

    free(swap_type);
    free(color);
    free(color_fill);
    free(new_file_name);
    return 0;
}

void all_keys(int argc, char *argv[], int* length, int* thickness, int*
fill, int* x1, int* y1, int* x2, int* y2,
                int** color, int** color_fill, char** swap_type, struct
Png* image){
    struct option long_opts[] = {
            {"info", no_argument, NULL, 'i'},
            {"help", no_argument, NULL, 'h'},
            {"start", required_argument, NULL, 's'},
            {"end", required_argument, NULL, 'e'},
            {"length", required_argument, NULL, 'l'},
            {"thinkness", required_argument, NULL, 't'},
            {"fill", required_argument, NULL, 'f'},
            {"color", required_argument, NULL, 'c'},
            {"colorfill", required_argument, NULL, 'r'},
            {"type", required_argument, NULL, 'p'},
            {NULL, 0, NULL, 0}
    };
    char *short_opts = "ihs:e:l:t:f:c:r:p:";
    int opt;

    while ((opt = getopt_long(argc, argv, short_opts, long_opts, NULL)) !
= -1){
        switch (opt){
            case 'i':
                print_PNG_info(image);
                break;
            case 'h':
                print_help();
                break;
            case 's': {
```

```
        int ind = optind - 1;
        int arg_len = strlen(argv[ind]);
        if (!isdigit(argv[ind][0])) break;
        *x1 = atoi(argv[ind]);
        int i = 0;
        for (; argv[ind][i] != '.'; i++)
            if (i >= arg_len) break;
        if (i == arg_len) break;
        if (!isdigit(argv[ind][i + 1])) break;
        *y1 = atoi(&argv[ind][i + 1]);
        break;
    }
    case 'e': {
        int ind = optind - 1;
        int arg_len = strlen(argv[ind]);
        if (!isdigit(argv[ind][0])) break;
        *x2 = atoi(argv[ind]);
        int i = 0;
        for (; argv[ind][i] != '.'; i++)
            if (i >= arg_len) break;
        if (i == arg_len) break;
        if (!isdigit(argv[ind][i + 1])) break;
        *y2 = atoi(&argv[ind][i + 1]);
        break;
    }
    case 'l': {
        int ind = optind - 1;
        if (!isdigit(argv[ind][0])) {
            optind--;
            break;
        }
        *length = atoi(argv[ind]);
        break;
    }
    case 't': {
        int ind = optind - 1;
        if (!isdigit(argv[ind][0])) {
            optind--;
            break;
        }
        *thickness = atoi(argv[ind]);
        break;
    }
    case 'f': {
        if (!isdigit(argv[optind - 1][0])) {
            optind--;
            break;
        }
        *fill = atoi(argv[optind - 1]);
        if (*fill != 0 && *fill != 1) {
            *fill = 0;
            break;
        }
        break;
    }
    case 'c': {
        int ind = optind - 1;
        int arg_len = strlen(argv[ind]);
```

```
        if (!isdigit(argv[ind][0])) break;
        (*color)[0] = atoi(argv[ind]);

        int i = 0;
        for (; argv[ind][i] != '.'; i++)
            if (i >= arg_len) break;
        if (i == arg_len) break;
        i ++;
        if (!isdigit(argv[ind][i])) break;
        (*color)[1] = atoi(&argv[ind][i]);

        for (; argv[ind][i] != '.'; i++)
            if (i >= arg_len) break;
        if (i == arg_len) break;
        i ++;
        if (!isdigit(argv[ind][i])) break;
        (*color)[2] = atoi(&argv[ind][i]);

        for (; argv[ind][i] != '.'; i++)
            if (i >= arg_len) break;
        if (i == arg_len) break;
        i ++;
        if (!isdigit(argv[ind][i])) break;
        (*color)[3] = atoi(&argv[ind][i]);

        break;
    }
    case 'r': {
        int ind = optind - 1;
        int arg_len = strlen(argv[ind]);

        if (!isdigit(argv[ind][0])) break;
        (*color_fill)[0] = atoi(argv[ind]);

        int i = 0;
        for (; argv[ind][i] != '.'; i++)
            if (i >= arg_len) break;
        if (i == arg_len) break;
        i ++;
        if (!isdigit(argv[ind][i])) break;
        (*color_fill)[1] = atoi(&argv[ind][i]);

        for (; argv[ind][i] != '.'; i++)
            if (i >= arg_len) break;
        if (i == arg_len) break;
        i ++;
        if (!isdigit(argv[ind][i])) break;
        (*color_fill)[2] = atoi(&argv[ind][i]);

        for (; argv[ind][i] != '.'; i++)
            if (i >= arg_len) break;
        if (i == arg_len) break;
        i ++;
        if (!isdigit(argv[ind][i])) break;
        (*color_fill)[3] = atoi(&argv[ind][i]);

        break;
```

```
            }
            case 'p': {
                strcpy(*swap_type, argv[optind - 1]);
                if (strcasecmp(*swap_type, "circle") != 0 &&
strcasecmp(*swap_type, "diagonal") != 0) {
                    strcpy(*swap_type, "none");
                }
                break;
            }
            default:
                break;
        }
    }

}
```