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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Обработка изображений

Студентка гр. 3342 _____ Смирнова Е.С.

Преподаватель _____ Глазунов С.А.

Санкт-Петербург

2024

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студентка Смирнова Е.С.

Группа 3342

Тема работы: обработка текстовых данных

Исходные данные:
Вариант 4.21

Все подзадачи, ввод/вывод должны быть реализованы в виде отдельной функции.

Содержание пояснительной записки:

Программа обязательно должна иметь CLI (опционально дополнительное использование GUI).

Программа должна реализовывать весь следующий функционал по обработке png-файла.

Общие сведения:

- Формат картинки PNG (рекомендуем использовать библиотеку libpng)
- без сжатия
- файл может не соответствовать формату PNG, т.е. необходимо проверка на PNG формат. Если файл не соответствует формату PNG, то программа должна завершиться с соответствующей ошибкой.
- обратите внимание на выравнивание; мусорные данные, если их необходимо дописать в файл для выравнивания, должны быть нулями.

- все поля стандартных PNG заголовков в выходном файле должны иметь те же значения что и во входном (разумеется, кроме тех, которые должны быть изменены).

Программа должна иметь следующую функции по обработке изображений:

- (1)Рисование прямоугольника. Флаг для выполнения данной операции: `--rect`. Он определяется:
  - Координатами левого верхнего угла. Флаг `--left_up`, значение задаётся в формате `left.up`, где left – координата по x, up – координата по y
  - Координатами правого нижнего угла. Флаг `--right_down`, значение задаётся в формате `right.down`, где right – координата по x, down – координата по y
  - Толщиной линий. Флаг `--thickness`. На вход принимает число больше 0
  - Цветом линий. Флаг `--color` (цвет задаётся строкой `rrr.ggg.bbb`, где rrr/ggg/bbb – числа, задающие цветовую компоненту. пример `--color 255.0.0` задаёт красный цвет)
  - Прямоугольник может быть залит или нет. Флаг `--fill`. Работает как бинарное значение: флага нет – false , флаг есть – true.
  - цветом которым он залит, если пользователем выбран залитый. Флаг `--fill_color` (работает аналогично флагу `--color`)
- (2)Рисование правильного шестиугольника. Флаг для выполнения данной операции: `--hexagon`. Шестиугольник определяется:
  - координатами его центра и радиусом в который он вписан. Флаги `--center` и `--radius`. Значение флаг `--center` задаётся в формате `x.y`, где x – координата по оси x, y – координата по оси y. Флаг `--radius` На вход принимает число больше 0
  - толщиной линий. Флаг `--thickness`. На вход принимает число

больше 0

- цветом линий. Флаг `--color` (цвет задаётся строкой `rrr.ggg.bbb`, где rrr/ggg/bbb – числа, задающие цветовую компоненту. пример `--color 255.0.0` задаёт красный цвет)
- шестиугольник может быть залит или нет. Флаг `--fill`. Работает как бинарное значение: флага нет – false , флаг есть – true.
- цветом которым залит шестиугольник, если пользователем выбран залитый. Флаг `--fill_color` (работает аналогично флагу `--color`)

- (3)Копирование заданной области. Флаг для выполнения данной операции: `--copy`. Функционал определяется:
  - Координатами левого верхнего угла области-источника. Флаг `--left_up`, значение задаётся в формате `left.up`, где left – координата по x, up – координата по y
  - Координатами правого нижнего угла области-источника. Флаг `--right_down`, значение задаётся в формате `right.down`, где right – координата по x, down – координата по y
  - Координатами левого верхнего угла области-назначения. Флаг `--dest_left_up`, значение задаётся в формате `left.up`, где left – координата по x, up – координата по y

Все подзадачи, ввод/вывод должны быть реализованы в виде отдельной функции.

Разделы пояснительной записки: «Содержание», «Введение», «Ввод и вывод изображения», «Обработка изображения», «Заключение», «Список использованных источников».

Предполагаемый объем пояснительной записки:

Не менее 15 страниц.

Дата выдачи задания: 18.03.2024

Дата сдачи реферата: 13.05.2024

Дата защиты реферата: 15.05.2024

Студентка _____ Смирнова Е.С.

Преподаватель _____ Глазунов С.А.

АННОТАЦИЯ

Курсовая работа представляет собой программу на языке Си, обрабатывающую PNG изображение по тексту задания. Для взаимодействия с пользователем был разработан CLI. Для работы с изображением использовались функции стандартных библиотек, динамическая память, структуры, библиотека libpng.

Исходный код работы программы приведён в приложении А.

Пример работы программы приведён в приложении Б.

SUMMARY

The coursework is a C program that processes a PNG image based on the text of the assignment. A CLI was developed for user interaction. To work with the image, the functions of standard libraries, dynamic memory, structures, and the libpng library were used.

The source code of the program is given in Appendix A.

An example of the program is given in Appendix B.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью работы является написание программы, обрабатывающей PNG файл с использованием библиотеки libpng, стандартных библиотек языка Си, а также с реализацией консольного интерфейса на getopt для удобного взаимодействия с пользователем. Программа должна реализовывать все функции, описанные в задании, в соответствии с требованиями с обработкой всевозможных ошибок.

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. ВВОД И ВЫВОД ИЗОБРАЖЕНИЯ

1.1. Ввод 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.

1.2. Вывод 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. ОБРАБОТКА ИЗОБРАЖЕНИЯ

2.1. Рисование прямоугольника

Реализована функция draw_rectangle, которая по двум заданным координатам, ширине контура, цвету контура и по цвету заливки, если она необходима, рисует поверх считанного изображения прямоугольник с помощью цикла в цикле for. Внутри функции производится проверка на корректность входных данных: ширина не может быть меньше нуля, цвета должны быть не меньше 0 и не больше 255.

2.2. Рисование правильного шестиугольника

Реализована функция draw_hexagon, которая по координатам центра правильного шестиугольника, радиусу его описанной окружности, ширине контура, цвету контура и по цвету заливки, если она необходима, определяет вершины фигуры и рисует поверх считанного изображения шестиугольник с помощью функции рисования линии. Внутри функции производится проверка на корректность входных данных: они не могут быть меньше нуля, координаты центра не могут выходить за пределы изображения, цвета должны быть не меньше 0 и не больше 255.

2.3. Копирование заданной области

Реализована функция copy, которая по координатам левого верхнего и правого нижнего углов, копирует попиксельно прямоугольник и заменяет каждый пиксель в новой области, заданной координатами верхнего левого угла. Внутри функции производится проверка на корректность входных данных: они не могут быть меньше нуля и выходить за пределы изображения.

3. ОСТАЛЬНЫЕ ФУНКЦИЯ

3.1. Функция main

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

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

Создается структура long_opts, содержащая все аргументы, передаваемые к командную строку. Аргументы строки перебираются с помощью встроенной функции getopt_long и с помощью оператора switch-case определяются введенные пользователем ключи и переменным присваиваются значения аргументов. Далее происходит чтение PNG-файла и обработка PNG изображения в соответствии с передаваемыми флагами и параметрами. Далее обработанное изображение записывается в PNG-файл, и программа завершается.

3.2 Функции print

Функция print_help выводит имя студента, вариант курсовой работы и справку о возможных аргументах.

Функция print_png_info выводит информацию о файле: ширину изображения, высоту изображения, тип цвета и глубину цвета.

3.3 Дополнительные функции

Функция set_pixel принимает на вход координаты пикселя, который нужно перекрасить и цвет, проверяет лежит ли этот пиксель в пределах изображения и окрашивает его.

Функция plot_circle принимает на вход координаты контура, его толщину и цвет, рисует круг с радиусом равном половине толщины с помощью функции set_pixel.

Функция fill_part закрашивает треугольники, ограниченные наклонными, в шестиугольнике.

Функции draw_line1, draw_line2, draw_line3 рисует отрезки, образующие шестиугольник с учетом ширины контура.

Функция get_color получает искомый цвет пикселя для функции copy.

ЗАКЛЮЧЕНИЕ

Разработана программа на языке программирования Си, обрабатывающая PNG изображения. В процессе разработки были задействованы библиотеки libpng, math и getopt, а также было написано множество вспомогательных функций и структур. Была разработана программа, обрабатывающая изображение согласно его запросу.

Сначала программа запрашивает у пользователя следующие доступные действий:

- --help : вызов функции помощи
- --info : вывод на экран информации о файле
- --input : ввод имени входного файла
- --output : ввод имени выходного файла
- --rect (--left_up --right_down --thickness --color --fill --fill_color) : рисование прямоугольника
- --hexagon (--center --radius --thickness --color --fill --fill_color) : рисование правильного шестиугольника
- --copy (--left_up --right_down --dest_left_up) : копирование заданной области

Программа обрабатывает изображение, сохраняет его.

Программа в соответствии с введенными параметрами выполняет определенный блок и завершает работу.

В приложении А представлены результаты тестирования программы. В приложении Б представлен код программы.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1) БАЗОВЫЕ СВЕДЕНИЯ К ВЫПОЛНЕНИЮ КУРСОВОЙ РАБОТЫ ПО ДИСЦИПЛИНЕ «ПРОГРАММИРОВАНИЕ». ВТОРОЙ СЕМЕСТР URL: https://se.moevm.info/lib/exe/fetch.php/courses:programming:programming_cw_metoda_2nd_course_last_ver.pdf.pdf

2) Требования к курсовым (весенний семестр) URL: http://se.moevm.info/doku.php/courses:programming:rules_extra_kurs

3) A description on how to use and modify libpng URL: http://www.libpng.org/pub/png/libpng-1.2.5-manual.html

ПРИЛОЖЕНИЕ А
ПРИМЕРЫ РАБОТЫ ПРОГРАММЫ

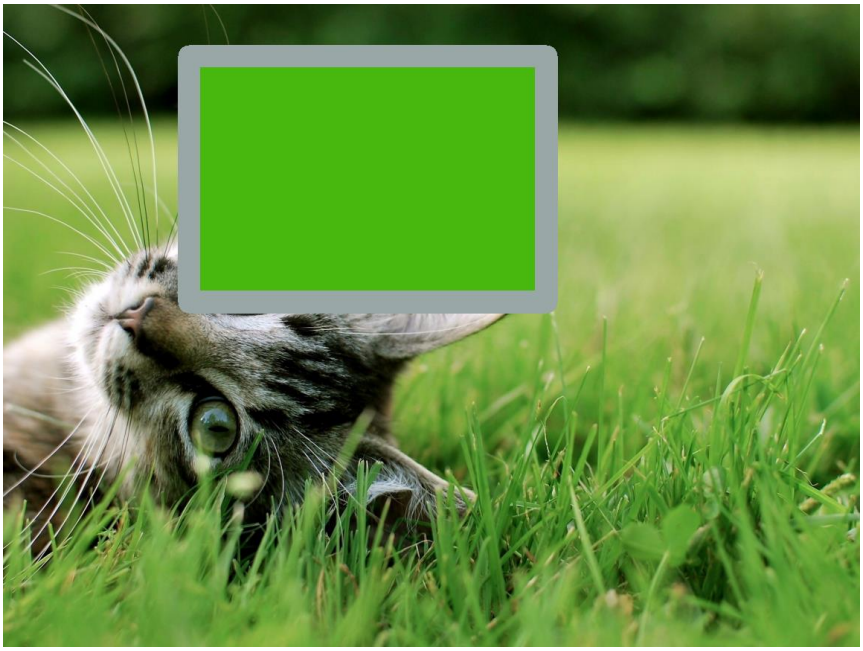

Рисунок 1- Пример рисования прямоугольника

Рисунок 2 - Пример рисования правильного шестиугольника

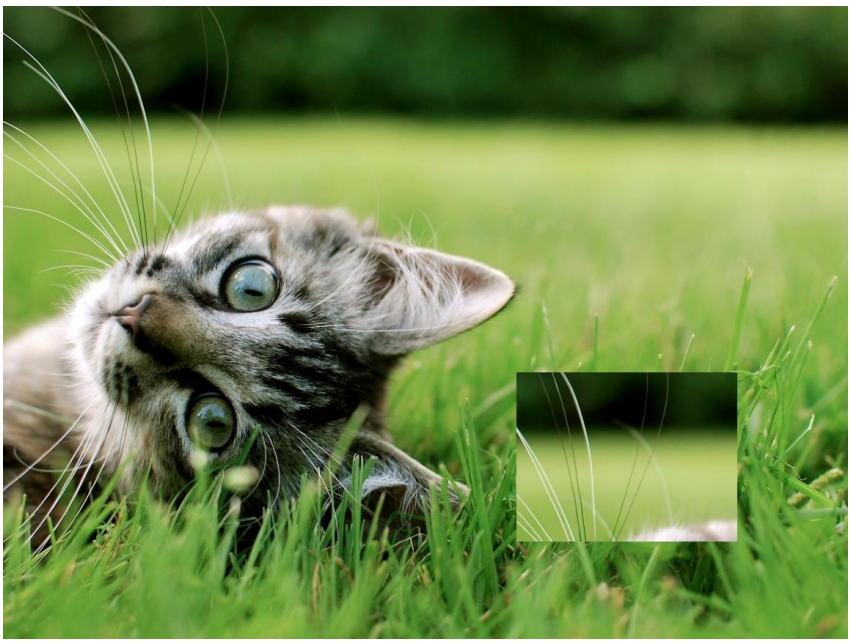

Рисунок 3 - Пример копирования заданной области

ПРИЛОЖЕНИЕ Б
КОД ПРОГРАММЫ

Название файла: main.c

```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <stdarg.h>
#include <stdbool.h>
#include <math.h>
#include <getopt.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;
    int number_of_passes;
    png_bytep* row_pointers;
};


void read_png_file(char* file_name, struct Png* image) {
    int y;
    unsigned 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(40);
    }

    fread(header, 1, 8, fp);
    if (png_sig_cmp(header, 0, 8)) {
        printf("Ошибка: формат изображения не PNG.\n");
        exit(41);
    }

    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(41);
    }

    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(41);
    }
```

```
    if (setjmp(png_jmpbuf(image->png_ptr))) {
        printf("Ошибка инициализации.\n");
        exit(41);
    }

    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))) {
        printf("Ошибка чтения изображения.\n");
        exit(41);
    }

    image->row_pointers = (png_bytep*)malloc(sizeof(png_bytep) * image-
>height);
    if (image->row_pointers == NULL) {
        printf("Ошибка выделения памяти.\n");
        exit(41);
    }
    for (y = 0; y < image->height; y++){

        image->row_pointers[y]                                          =
(png_byte*)malloc(png_get_rowbytes(image->png_ptr, image->info_ptr));
        if (image->row_pointers[y] == NULL){
            printf("Ошибка выделения памяти.\n");
            exit(41);
        }
    }
    png_read_image(image->png_ptr, image->row_pointers);

    fclose(fp);
}


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

    FILE* fp = fopen(file_name, "wb");
    if (!fp) {
        printf("Ошибка при создании файла итогового изображения.\n");
        exit(42);
    }
```

```
    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(42);
    }

    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(42);
    }

    if (setjmp(png_jmpbuf(image->png_ptr))) {
        printf("Ошибка инициализации\n");
        exit(42);
    }

    png_init_io(image->png_ptr, fp);

    if (setjmp(png_jmpbuf(image->png_ptr))) {
        printf("Ошибка записи заголовка итогового файла.\n");
        exit(42);
    }

    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(42);
    }

    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(42);
    }

    png_write_end(image->png_ptr, NULL);

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

    fclose(fp);
}

void set_pixel(struct Png* image, int x, int y, int* color) {
    if (!(x >= image->width || y >= image->height || x < 0 || y < 0)){
```

```
        png_byte *row = image->row_pointers[y];
        png_byte *ptr = &(row[x * 3]);

        ptr[0]=color[0];
        ptr[1]=color[1];
        ptr[2]=color[2];
    }
}

void plot_circle(struct Png* image, int xm, int ym, float thickness, int*
color){
    int x0 = xm - thickness/2, x1 = xm + thickness/2, y0 = ym -
thickness/2, y1 = ym + thickness/2;

    for (int x = x0; x <= x1; x++){
        for (int y = y0; y <= y1; y++){
            if       ((x-xm)*(x-xm)       +       (y-ym)*(y-ym)       <=
(thickness/2)*(thickness/2)){
                if (!(x >= image->width || y >= image->height || x < 0 ||
y < 0))
                    set_pixel(image, x, y, color);
            }
        }
    }
}

void fill_part(struct Png* image, int x, int x0, int x1, int y, bool
fill, int* fill_color){
    int dx = x1 - x;
    int dx0 = x0 + 3 * (x1 - x0);

    for (int j = x; j <= x1; j++){
        if (fill && (j > x && j <= x1) ){
            set_pixel(image, j, y, fill_color);
        }
    }

    for (int j = dx0; j <= dx0 + dx; j++){
        if (fill && (j >= dx0 && j < dx0 + dx) ){
            set_pixel(image, j, y, fill_color);
        }
    }

}
void fill_rect(struct Png* image, int x0, int y0, int x1, int y1, bool
fill, int* fill_color){

    for (int i = y0; i <= y1; i++){
          for (int j = x0; j <= x1; j++){
            if (fill && (j >= x0 && j <= x1) && (i >= y0 && i <= y1)) {
                set_pixel(image, j, i, fill_color);
            }
          }
     }
}
```

```
void draw_line1(struct Png* image, int x0, int y0, int x1, int y1, float
r, float thickness, int* color, bool fill, int* fill_color) {//
    int dx = abs(x1 - x0), dy = (abs(y1 - y0));
    int d = 2 * dx - dy;
    int incrE = 2 * dx;
    int incrNE = 2 * (dx - dy);
    int x = x0, y = y0;

    while (y <= y1) {
        fill_part(image, x, x0, x1, y, fill, fill_color);
        set_pixel(image, x, y, color);
        if (d <= 0){
            d += incrE;
            y++;
        }
        else{
            d += incrNE;
            x++;
            y++;
        }
    }
    for (x = x0 - (thickness*sqrt(3)/4); x <= x0 + thickness*sqrt(3)/4;
x++){
        draw_line1(image, x, y0 + thickness/4, x + r/2, y1 + thickness/4,
0, -1, color, false, fill_color);
    }
    for (x = x0; x <= x0 + (thickness*sqrt(3)/4); x++){
        draw_line1(image, x, y0 - thickness/4, x + r/2, y1 - thickness/4,
0, -1, color, false, fill_color);
    }
}

void draw_line2(struct Png* image, int x0, int y0, int x1, int y1, float
thickness, int* color) {
    int dx = abs(x1 - x0), dy = (abs(y1 - y0));
    int d = 2 * dy - dx;
    int incrE = 2 * dy;
    int incrNE = 2 * (dy - dx);
    int x = x0, y = y0;

    while (x <= x1) {

        if (d <= 0){
            d += incrE;
            x++;
        }
        else{
            d += incrNE;
            x++;
            y++;
        }
    }
    for ( x = x0; x <= x1; x++){
        for (int y = y0 - thickness/2; y <= y0 + thickness/2; y++){
            set_pixel(image, x, y, color);
        }
    }
```

```
}

void draw_line3(struct Png* image, int x0, int y0, int x1, int y1, float
r, float thickness, int* color, bool fill, int* fill_color) { //
    int dx = abs(x1 - x0), dy = (abs(y1 - y0));
    int d = 2 * dx - dy;
    int incrE = 2 * dx;
    int incrNE = 2 * (dx - dy);
    int x = x0, y = y0;

    while (y >= y1) {
        fill_part(image, x, x0, x1, y, fill, fill_color);
        set_pixel(image, x, y, color);
        if (d <= 0){
            d += incrE;
            y--;
        }
        else{
            d += incrNE;
            y--;
            x++;
        }
    }
    for (x = x0 - (thickness*sqrt(3)/4); x <= x0 + thickness*sqrt(3)/4;
x++){
        draw_line3(image, x, y0 - thickness/4, x + r/2, y1 -thickness/4,
0, -1, color, false, fill_color);
    }
    for (x = x0; x <= x0 + (thickness*sqrt(3)/4); x++){
        draw_line3(image, x, y0 + thickness/4, x + r/2, y1 + thickness/4,
0, -1, color, false, fill_color);
    }
}

void draw_rectangle(struct Png* image, int x0, int y0, int x1, int y1,
float thickness, int* color, bool fill, int* fill_color) {
    if (thickness < 0 )  {
        printf("Введены некорретные данные: "
               "ширина линий не может иметь"
               " отрицательное значение\n");
        exit(45);
    }
    if (x0 >= x1){
        int tmpx = x0;
        x0 = x1;
        x1 = tmpx;
    }
    if (y0 >= y1){
        int tmpy = y0;
        y0 = y1;
        y1 = tmpy;
    }
    if (color[0] > 255 || color[0] < 0 || color[1] > 255 || color[1] < 0
    || color[2] > 255 || color[2] < 0 ) {
        printf("Введены некорретные данные: цвета должны лежать от 0 до
255\n");
        exit(45);
```

```
    }
    if ((fill_color[0] > 255 || fill_color[0] < 0 || fill_color[1] > 255
|| fill_color[1] < 0
        || fill_color[2] > 255 || fill_color[2] < 0 ) && (fill)){
        printf("Введены некорретные данные: цвета должны лежать от 0 до
255\n");
        exit(45);
    }

    for (int i = y0; i <= y1; i++){
          for (int j = x0; j <= x1; j++){
               if ((i == y0) || (i == y1) || (j == x0) || (j == x1)) {
               plot_circle(image, j, i, thickness, color);

           }
           else if (fill && (j >= x0 + thickness/2 && j <= x1 -
thickness/2) && (i >= y0 + thickness/2 && i <= y1 - thickness/2)) {
               set_pixel(image, j, i, fill_color);
           }
          }
     }
}
void draw_hexagon(struct Png* image, int x0, int y0, float r, float
thickness, int* color, bool fill, int* fill_color){

    if (x0 < 0 || y0 < 0 || r < 0 || thickness < 0 )  {
        printf("Введены некорретные данные: "
               "координаты, радиус и ширина линий "
               "не могут иметь отрицательные значения\n");
        exit(45);
    }
    if (x0 >= image->width || y0 >= image->height ) {
        printf("Введены некорретные данные: координаты должны "
               "находиться в пределах изображения\n");
        exit(45);
    }
    if (color[0] > 255 || color[0] < 0 || color[1] > 255 || color[1] < 0
    || color[2] > 255 || color[2] < 0 ) {
        printf("Введены некорретные данные: цвета должны лежать от 0 до
255\n");
        exit(45);
    }
    if ((fill_color[0] > 255 || fill_color[0] < 0 || fill_color[1] > 255
|| fill_color[1] < 0
        || fill_color[2] > 255 || fill_color[2] < 0 ) && (fill)){
        printf("Введены некорретные данные: цвета должны лежать от 0 до
255\n");
        exit(45);
    }

    int x1 = x0 + r, y1 = y0;
    int x2 = x0 + (r/2), y2 = y0 - (r * (sqrt(3))/2);
    int x3 = x0 - (r/2), y3 = y0 - (r * (sqrt(3))/2);
    int x4 = x0 - r, y4 = y0;
    int x5 = x0 - (r/2), y5 = y0 + (r * (sqrt(3))/2);
    int x6 = x0 + (r/2), y6 = y0 + (r * (sqrt(3))/2);
```

```
    fill_rect(image, x3, y3, x6, y6, fill, fill_color);

    draw_line2(image, x3, y3, x2, y2, thickness, color) ;
    draw_line3(image,  x4,  y4,  x3,  y3,  r,  thickness,  color,  fill,
fill_color) ;
    draw_line1(image,  x4,  y4,  x5,  y5,  r,  thickness,  color,  fill,
fill_color) ;
    draw_line2(image, x5, y5, x6, y6, thickness, color) ;
    draw_line3(image,  x6,  y6,  x1,  y1,  r,  thickness,  color,  false,
fill_color) ;
    draw_line1(image,  x2,  y2,  x1,  y1,  r,  thickness,  color,  false,
fill_color) ;

    plot_circle(image, x1, y1, thickness, color);
    plot_circle(image, x2, y2, thickness, color);
    plot_circle(image, x3, y3, thickness, color);
    plot_circle(image, x4, y4, thickness, color);
    plot_circle(image, x5, y5, thickness, color);
    plot_circle(image, x6, y6, thickness, color);
}

void get_color(struct Png* image, int x, int y, int x0, int y0, int*
color_pixel){

    png_byte *row = image->row_pointers[y];
    png_byte *ptr = &(row[x * 3]);
    color_pixel[0] = ptr[0];
    color_pixel[1] = ptr[1];
    color_pixel[2] = ptr[2];
    set_pixel(image, x0, y0, color_pixel);
}

void copy(struct Png* image, int x0, int y0, int x1, int y1, int x2, int
y2){

    if (x0 < 0 || y0 < 0 || x1 < 0 || y1 < 0 || x2 < 0 || y2 < 0)  {
        printf("Введены некорретные данные: "
               "координаты, радиус и ширина линий "
               "не могут иметь отрицательные значения\n");
        exit(45);
    }
    if (x0 >= image->width || x1 >= image->width || y0 >= image->height
|| y1 >= image->height
    || x2 >= image->width || y2 >= image->height) {
        printf("Введены некорретные данные: координаты должны "
               "находиться в пределах изображения\n");
        exit(45);
    }
    if (x1 < x0 || y1 < y0 ) {
        printf("Введены некорретные данные: координаты верхнего левого
угла "
               "должны быть меньше координат нижнего правого угла\n");
        exit(45);
    }

    int dx = x2 - x0, dy = y2 - y0;
    int color_pixel[3];
```

```
    if ((x2 <= x1) && (x2 >= x0) && (y2 <= y1) && (y2 >= y0)){
        for (int i = y1; i >= y0; i--){
            for (int j = x1; j >= x0; j--){
                get_color(image, j, i, j + dx, i + dy, color_pixel);
            }
        }
    }
    else {
        for (int i = y0; i <= y1; i++){
            for (int j = x0; j <= x1; j++){
                get_color(image, j, i, j + dx, i + dy, color_pixel);
            }
        }
    }

}

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("Course work for option 4.21, created by Smirnova
Elizaveta.\n\n");
    printf("Флаги:\n\n");
    printf("--help : вызов функции помощи\n\n");
    printf("--info : вывод на экран информации о файле\n\n");
    printf("--input : ввод имени входного файла\n\n");
    printf("--output : ввод имени выходного файла\n\n");
    printf("--rect (--left_up --right_down --thickness --color --fill --
fill_color) : рисование прямоугольника\n\n");
    printf("--hexagon (--center --radius --thickness --color --fill --
fill_color) : рисование правильного шестиугольника\n\n");
    printf("--copy (--left_up --right_down --dest_left_up) : копирование
заданной области\n\n");
}

int main(int argc, char* argv[]) {

    char* input_file = argv[argc - 1];
    char* output_file = "res.png";

    struct Png image;

    const char* short_opts = "hI:rs:f:t:q:Ff:HC:R:cS:o:i:";
    const struct option long_opts[] ={
            {"help", no_argument, NULL, 'h'},
            {"info", required_argument,NULL, 'I'},
            {"rect", no_argument,NULL, 'r'},
            {"left_up", required_argument,NULL, 's'},
            {"right_down", required_argument,NULL, 'e'},
            {"thickness", required_argument, NULL, 't'},
            {"color", required_argument, NULL, 'q'},
```

```
            {"fill", no_argument, NULL, 'F'},
            {"fill_color", required_argument, NULL, 'f'},
            {"hexagon", no_argument, NULL, 'H'},
            {"center", required_argument, NULL, 'C'},
            {"radius", required_argument, NULL, 'R'},
            {"copy", no_argument, NULL, 'c'},
            {"dest_left_up", required_argument, NULL, 'S'},
            {"output", required_argument, NULL, 'o' },
            {"input", required_argument, NULL, 'i' },
    };

    int x0 = 0, y0 = 0, x1 = 0, y1 = 0, x2 = 0, y2 = 0, opt = -1,
opt_number = -1;
    float thickness = -1, radius = 0;
    int fill_color[3] = {0,0,0};
    int color[3] = {0,0,0};
    bool fill = false;


    while ((opt = getopt_long(argc, argv, short_opts, long_opts, NULL))
!= -1){

        switch(opt){
            case 'h':
            {
                print_help();
                exit(0);
                break;
            }
            case 'I':
            {
                read_png_file(input_file, &image);
                print_png_info(&image);
                break;
            }
            case 'r':
            {
                opt_number = 1;
                break;
            }
            case 's':
            {
                sscanf(optarg, "%d.%d", &x0, &y0);
                break;
            }
            case 'e':
            {
                sscanf(optarg, "%d.%d", &x1, &y1);
                break;
            }
            case 't':
            {
                sscanf(optarg, "%f", &thickness);
                break;
            }
            case 'q':
            {
```

```
                sscanf(optarg,    "%d.%d.%d",    &color[0],    &color[1],
&color[2]);
                break;
            }
            case 'F':
            {
                fill = true;
                break;
            }
            case 'f':
            {
                sscanf(optarg,        "%d.%d.%d",        &fill_color[0],
&fill_color[1], &fill_color[2]);
                break;
            }
            case 'H':
            {
                opt_number = 2;
                break;
            }
            case 'C':
            {
                sscanf(optarg, "%d.%d", &x0, &y0);
                break;
            }
            case 'R':
            {
                sscanf(optarg, "%f", &radius);
                break;
            }
            case 'c':
            {
                opt_number = 3;
                break;
            }
            case 'S':
            {
                sscanf(optarg, "%d.%d", &x2, &y2);
                break;
            }
            case 'o':
            {
                output_file = optarg;
                break;
            }
            case 'i':
            {
                input_file = optarg;
                break;
            }
            default:
            {
                break;
            }
        }
    }
    read_png_file(input_file, &image);
```

```
    switch (opt_number)
    {
        case 1:
        {
            draw_rectangle(&image, x0, y0, x1, y1, thickness, color,
fill, fill_color);
            break;
        }
        case 2:
        {
            draw_hexagon(&image, x0, y0, radius, thickness, color, fill,
fill_color);
            break;
        }
        case 3:
        {
            copy(&image, x0, y0, x1, y1, x2, y2);
            break;
        }
        default:
        {
            break;
        }
    }

    write_png_file(output_file, &image);
    return 0;
}
```