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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Обработка png-файлов

Студент гр. 9383 _____ Корсунов А. А.

Преподаватель _____ Размочаева Н. В.

Санкт-Петербург
2020

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студент Корсунов А. А.

Группа 9383

Тема работы: Обработка png-файлов

Исходные данные:

Язык программирования с. Сборка под ОС Linux. Файл формата png,

Содержание пояснительной записки:

«Аннотация», «Содержание», «Введение», «Задание курсовой работы», «Файлы программы», «Функции программы», «Примеры работы программы», «Заключение», «Приложение А».

Предполагаемый объем пояснительной записки:

Не менее 33 страницы.

Дата выдачи задания: 02.03.2020

Дата сдачи реферата: 21.05.2020

Дата защиты реферата: 25.05.2020

Студент _____ Корсунов А. А.

Преподаватель _____ Размочаева Н. В.

АННОТАЦИЯ

В ходе подготовки курсовой работы была написана программа, получающая на вход файл формата png. С помощью интерфейса командной строки (CLI) пользователь может выбрать одну из следующих операций: рисование квадрата с диагоналями с возможностью заливки и выбора цветов для заливки и линий квадрата, отключение или включение на максимум одной из rgb-компонент, поворот выбранной области на 90, 180 или 270 градусов. При этом создается копия исходного изображения, все необходимые действия производится на ней и выводится тоже она.

SUMMARY

During the preparation of the course work, a program was written that receives a png file as input. Using the command-line interface (CLI), the user can choose one of the following operations: drawing a square with diagonals with the ability to fill and select colors for filling and lines of the square, disabling or enabling a maximum of one of the rgb components, rotating the selected area by 90, 180 or 270 degrees. This creates a copy of the original image, all the necessary actions are performed on it, and it is also output.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель работы: написать программу, которая принимает на вход png-изображение в соответствии с CLI.

Для реализации данной цели необходимо решить следующие задачи:

1. Изучение работы с png-изображениями на языке СИ.
2. Изучение необходимых библиотек языка СИ.
3. Создание структуры png-изображения и функций, которые считывают это изображение и выводят.
4. Создание функций обработки изображения.
5. Изучение работы с параметрами CLI.
6. Тестирование программы.
7. Отладка программы.
8. Очистка памяти.

1. ЗАДАНИЕ КУРСОВОЙ РАБОТЫ

ВАРИАНТ 24

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

Программа должна реализовывать весь следующий функционал по обработке png-файла

Общие сведения

- **Формат картинки PNG (рекомендуем использовать библиотеку libpng)**
- без сжатия
- файл 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 с диагоналями. Квадрат определяется:
  - Координатами левого верхнего угла
  - Размером стороны
  - Толщиной линий
  - Цветом линий
  - Может быть залит или нет (диагонали располагаются “поверх” заливки)
  - Цветом которым он залит, если пользователем выбран залитый

- (2) Фильтр rgb-компонент. Этот инструмент должен позволять для всего изображения либо установить в 0 либо установить в 255 значение заданной компоненты. Функционал определяется
    - Какую компоненту требуется изменить
    - В какой значение ее требуется изменить
- (3) Поворот изображения (части) на 90/180/270 градусов. Функционал определяется
    - Координатами левого верхнего угла области
    - Координатами правого нижнего угла области
    - Углом поворота

2.ФУНКЦИИ И СТРУКТУРЫ ПРОГРАММЫ

2.1. struct Png

Данная структура служит для работы с png-изображением. В ней определяются требуемые параметры изображения.

2.2. read_png_file(char *file_name, struct Png *image)

Данная функция считывает png-файл, инициализирует структуру изображения, выводит сообщение ошибки, если считать изображение не удалось.

2.3 write_png_file(char *file_name, struct Png *image)

Данная функция создает изображение после обработки (если она произошла), если нет, то копию исходного изображения, после чего выводит его.

2.4 square(struct Png *image, int start_x, int start_y, int start_a, int line_width, int new_red, int new_green, int new_blue, int condition, int new_red_1, int new_green_1, int new_blue_1)

Данная функция рисует требуемый по заданию квадрат с диагоналями. Функция работает только с png-изображенями с альфа каналом. Если один из аргументов не подходит по каким-то причинам, выведется сообщение об ошибке. Вначале функция проходится по сторонам квадрата и закрашивает каждый пиксель в соответствии с выбранным цветом, толщиной линий и размером стороны, далее она про проверяет значение флага заливки, если пользователь выбрал залить, то функция проходится по каждому пикселю внутри квадрата и меняет его цвет согласно выбранному. В конце программа рисует диагонали (так же меняя цвет каждого пикселя). Толщина диагоналей будет немного меньше толщины сторон (сделано это, чтобы диагональ была ровная).

2.5 rtd(struct Png *image, int condition_1, int condition_2)

Данная функция меняет значение одной из rgb-компонент на 0 либо на 255. Вначале функция проверяет правильность введенных аргументов (если они некорректны, то выводит ошибку), далее функция проходит по всему изображению и отключает или включает на максимум выбранную rgb-компоненту.

2.6 turn(struct Png *image, int x1, int y1, int x2, int y2, int corner)

Данная функция поворачивает выбранную область на 90 или 180 или 270 градусов. Вначале функция проверяет правильность введенных координат, если они не подходят, то будет выведено сообщение об ошибке. Далее функция создает копию изображения, с которой будет «перемещать» пиксели на переданную в функцию картинку. Стоит учесть, что поворот происходит относительно левого верхнего угла. То есть, если выбранная область в длину была больше, чем в ширину, то есть перевернуть эту картинку на 90 градусов, то она будет располагаться начиная с координаты левого верхнего угла. В конце происходит очистка выделенной памяти.

2.7 help()

Данная функция выводит справку с информацией о пользовании всей программы.

2.8 clear(struct Png *image)

Данная функция очищает выделенную память.

2.9 main(int argc, char **argv)

С помощью данной функции производится обращение к требуемой по заданию функции с помощью созданных в ней же ключей (структура и ключи созданы под функцию getopt_long). В данной функции также идет проверка на тип входных параметров (с помощью функции isdigit), а также проверка на

правильное число введенных аргументов. В случае несовпадения, будет выведено сообщение об ошибке.

3. ПРИМЕРЫ РАБОТЫ ПРОГРАММЫ

3.1. Вызов справки

```
annnton@antonkorsunov:~/курсовая_2$ ./a.out 2.png out5.png -h
Использование утилиты: <имя_программы> <имя входного файла> <имя выходного файла> <ключ> <параметры>
ключ может быть полным (тогда используйте <--> перед ним) и сокращенным (тогда используйте <-> перед ним
вам доступны следующие ключи:
1. -s или --square - рисование квадрата с дигоналями с возможностью заливки, с выбором цветов заливки и линий квадрата
параметры для квадрата должны указываться в следующем порядке через пробел:
<координата x левого верхнего угла> <координата y левого верхнего угла> <размер стороны> <толщина линий>
<красный цвет линий (от 0 до 255)> <зеленый цвет линий (от 0 до 255)> <синий цвет линий (от 0 до 255)>
<заливка (1 - заливать, 0 или любое другое число - не заливать)> <красный цвет заливки (от 0 до 255, если вы не выбрали
заливку, укажите 0)> <зеленый цвет заливки (от 0 до 255, если вы не выбрали заливку, укажите 0)> <синий цвет заливки
(от 0 до 255, если вы не выбрали заливку укажите 0)>
2. -r или --rtd - выключить или включить на максимум одну из компонент цвета картинки
параметры для rtd должны указываться в следующем порядке через пробел:
<выбор компоненты (от 0 до 2)> <цвет выбранной компоненты (0 либо 255)>
3. -t или --turn - повернуть выбранную область на углы 90, 180 и 270
параметры для поворота должны указываться в следующем порядке через пробел
<координата x левого верхнего угла> <координата y левого верхнего угла> <координата x правого нижнего угла>
<координата y правого нижнего угла> <угол поворота (90, 180 или 270)>
4. -i или --info - распечать картинку
5. -h или --help - получить справку о помоще
если вы что-то введете неправильно, вам сообщат об ошибке
annnton@antonkorsunov:~/курсовая_2$
```

3.2. Ключ <s>

Исходник:

1) запуск через ./a.out 2.png out1.png --square 123 134 532 50 231 123 54 0 0 0 0

2) запуск через ./a.out 2.png out2.png --square 231 21 532 32 123 43 54 1 234 42 222

3.3 ключ <r>

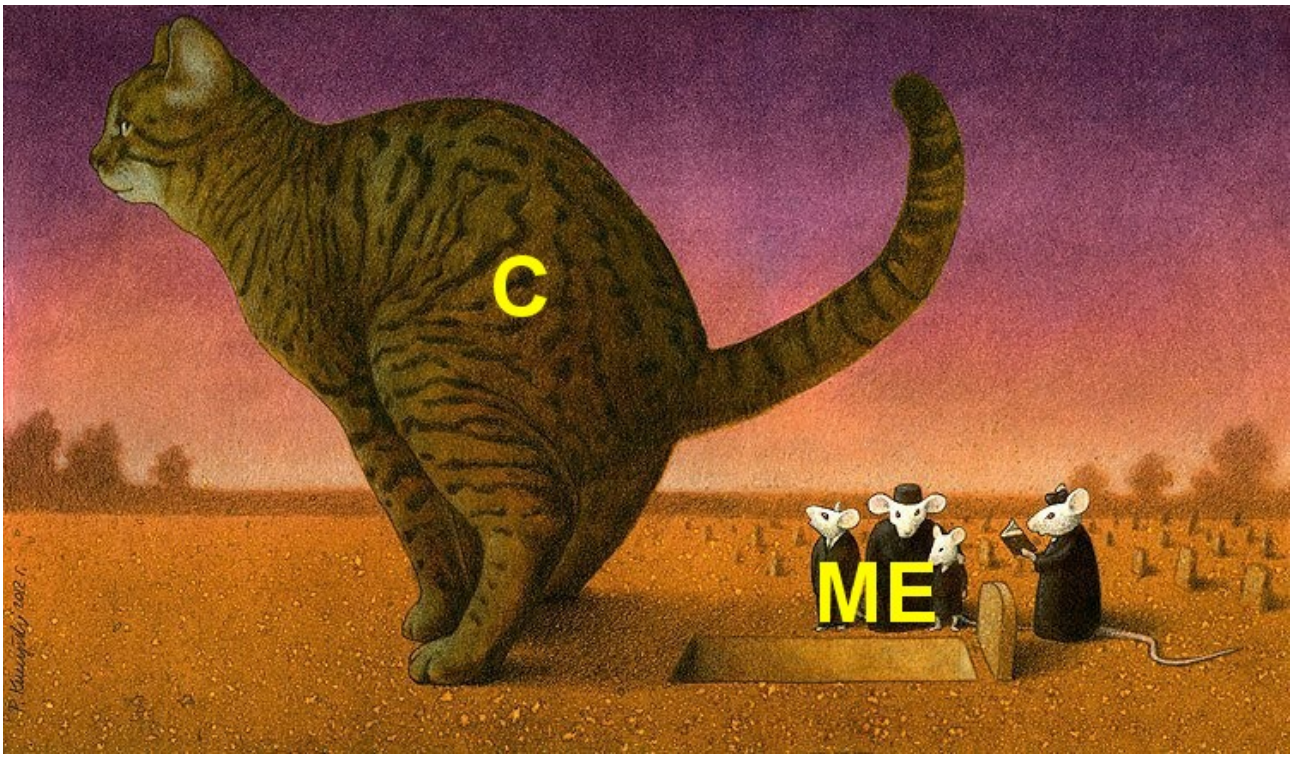
Исходник:

1) запуск через ./a.out 1.png out3.png -r 0 255

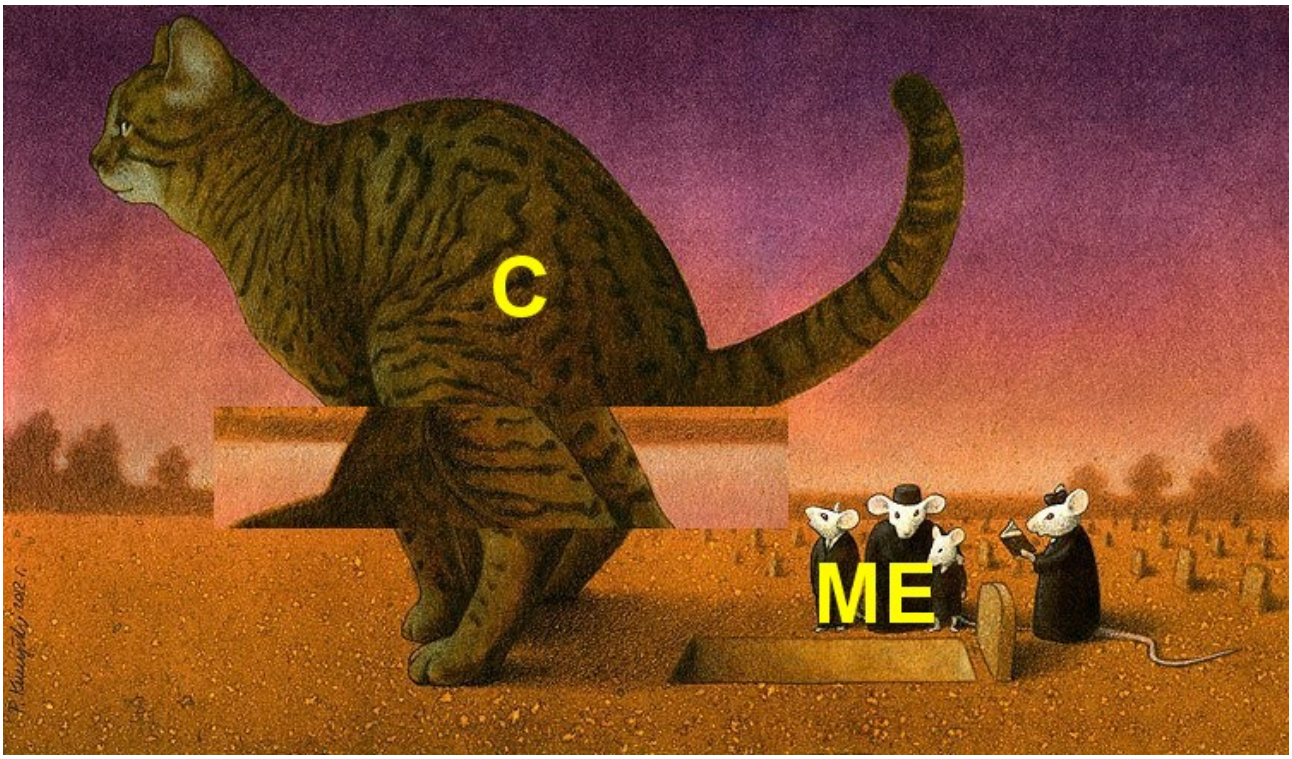
2) запуск через ./a.out 1.png out4.png -r 1 0

3. Ключ <t>

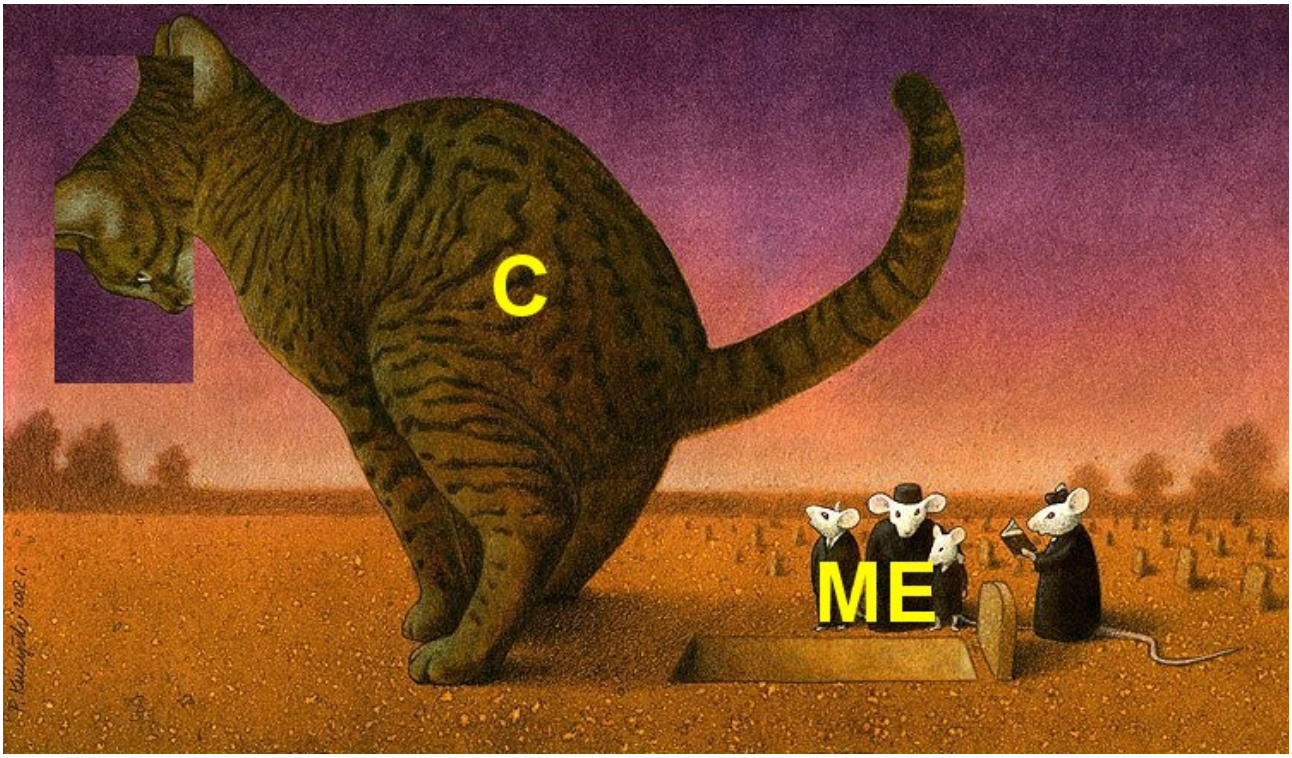
Исходник:



1) запуск через ./a.out 3.png out5.png -t 123 234 457 304 180



2) запуск через ./a.out 3.png out6.png -t 10 20 200 100 270



3.5 Обработка ошибок

1) Вводится файл не типа png:

```
annnton@antonkorsunov:~/курсовая_2$ ./a.out 22.jpj out3.png -r 0 255
При считывание файла произошла ошибка
annnton@antonkorsunov:~/курсовая_2$
```

2) Вводится несуществующий файл:

```
annnton@antonkorsunov:~/курсовая_2$ ./a.out 212.jpj out3.png -r 0 255
При считывание файла произошла ошибка
```

3) Вводятся некорректные данные:

```
annnton@antonkorsunov:~/курсовая_2$ ./a.out 1.png out10.png -r 0 10000
Доступны только два цвет - под номер 0 и под номером 255
```

ЗАКЛЮЧЕНИЕ

Были изучены библиотеки СИ. Реализованы функции обработки png-изображений с альфа каналом. Расширены знания о стандартной библиотеке СИ. Была написана требуемая по заданию программа. Был предусмотрен ввод некорректных данных и вывод соответствующей ошибки. Были продемонстрированы возможности программы и подведены итоги.

ПРИЛОЖЕНИЕ А

РАЗРАБОТАННЫЙ КОД

```
#include <unistd.h>
#include <getopt.h>
#include <ctype.h>
#include <stdlib.h>
#include <stdio.h>
#include <stdarg.h>
#include <string.h>
#define PNG_DEBUG 3
#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;
};

int read_png_file(char *file_name, struct Png *image){
	int x,y;
	char header[8]; //8 - максимальный размер

	/* открытие файла и его на проверка на тип png */
	FILE *fp = fopen(file_name, "rb");
	if(!fp){
		fprintf(stderr, "При считывание файла произошла ошибка\n");
		return 1;
	}

	fread(header, 1, 8, fp);
	if(png_sig_cmp(header, 0, 8)){
		fprintf(stderr, "Считанный файл не является png-файлом\n");
		return 1;
	}

	/* Инициализирование png_ptr структурой */
```

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

	if(!image->png_ptr){
		fprintf(stderr, "Произошла ошибка при создании файла\n");
		return 1;
	}

	image->info_ptr = png_create_info_struct(image->png_ptr);
	if(!image->info_ptr){
		fprintf(stderr, "Произошла ошибка при создании структуры\n");
		return 1;
	}

	if(setjmp(png_jmpbuf(image->png_ptr))){
		fprintf(stderr, "Произошла ошибка при инициализации ввода/вывода\n");
		return 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);

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

	/* чтение файла */
	if(setjmp(png_jmpbuf(image->png_ptr))){
		fprintf(stderr, "Произошла ошибка при считывании файла\n");
		return 1;
	}

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

```
	}

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

	fclose(fp);
	return 0;
}

void write_png_file(char *file_name, struct Png *image){
	int x,y;
	/* создание файла */
	FILE *fp = fopen(file_name, "wb");
	if (!fp){
		fprintf(stderr, "Произошла ошибка при создании файла под запись\
n");
		return;
	}

	/* Инициализирование структурой */
	image->png_ptr = png_create_write_struct(PNG_LIBPNG_VER_STRING,
NULL, NULL, NULL);

	if(!image->png_ptr){
		fprintf(stderr, "Произошла ошибка при инициализации структурой\
n");
		return;
	}

	image->info_ptr = png_create_info_struct(image->png_ptr);
	if(!image->info_ptr){
		fprintf(stderr, "Произошла ошибка при инициализации\n");
		return;
	}

	if(setjmp(png_jmpbuf(image->png_ptr))){
		fprintf(stderr, "Произошла ошибка при инициализации
ввода/вывода\n");
		return;
	}

	png_init_io(image->png_ptr, fp);

	/* записываем header */
	if(setjmp(png_jmpbuf(image->png_ptr))){
```

```
            fprintf(stderr, "Произошла ошибка при записи header\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);

        /* записываем bytes */
        if(setjmp(png_jmpbuf(image->png_ptr))){
            fprintf(stderr, "Произошла ошибка при записи bytes\n");
            return;
        }

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

        /* записываем конец файла */
        if(setjmp(png_jmpbuf(image->png_ptr))){
            fprintf(stderr, "Произошла ошибка при записи конца файла\n");
            return;
        }

        png_write_end(image->png_ptr, NULL);
        fclose(fp);
}

void square(struct Png *image, int start_x, int start_y, int start_a, int line_width, int new_red, int new_green, int new_blue, int condition, int new_red_1, int new_green_1, int new_blue_1){

        int x,y;
        if(png_get_color_type(image->png_ptr, image->info_ptr) != PNG_COLOR_TYPE_RGBA){
            fprintf(stderr, "Изображение должно соответствовать PNG_COLOR_TYPE_RGBA\n");
            return;
        }

        if(start_x < 0 || start_y < 0 || start_x > image->width - 1 || start_y > image->height - 1){
            fprintf(stderr, "Недопустимые координаты левого верхнего угла\n");
            return;
```

```
	}

	if(start_a < 0 || start_a > image->width - start_x - 1 || start_a > image->height - start_y - 1){
		fprintf(stderr, "Недопустимый размер стороны\n");
		return;
	}

	if(line_width < 0 || line_width > start_a){
		fprintf(stderr, "Недопустимая толщина линий\n");
		return;
	}

	for(y = start_y; y < start_y + line_width; y++){
		png_byte *row = image->row_pointers[y];
		for(x = start_x; x < start_x + start_a; x++){
			png_byte *ptr = &(row[x * 4]);
			ptr[0] = new_red;
			ptr[1] = new_green;
			ptr[2] = new_blue;
			ptr[3] = 255;
		}
	}

	for(y = start_y + start_a; y > start_y + start_a - line_width; y--){
		png_byte *row = image->row_pointers[y];
		for(x = start_x; x < start_x + start_a; x++){
			png_byte *ptr = &(row[x * 4]);
			ptr[0] = new_red;
			ptr[1] = new_green;
			ptr[2] = new_blue;
			ptr[3] = 255;
		}
	}

	for(y = start_y; y < start_y + start_a; y++){
		png_byte *row = image->row_pointers[y];
		for(x = start_x; x < start_x + line_width; x++){
			png_byte *ptr = &(row[x * 4]);
			ptr[0] = new_red;
			ptr[1] = new_green;
			ptr[2] = new_blue;
			ptr[3] = 255;
		}
```

```
        }

        for(y = start_y; y < start_y + start_a; y++){
                png_byte *row = image->row_pointers[y];
                for(x = start_x + start_a ; x > start_x + start_a - line_width; x--){
                        png_byte *ptr = &(row[x * 4]);
                       ptr[0] = new_red;
                        ptr[1] = new_green;
                        ptr[2] = new_blue;
                        ptr[3] = 255;
                }
        }

        if(condition == 1){
                for(y = start_y + line_width; y < start_y + start_a - line_width + 1; y++){
                        png_byte *row = image->row_pointers[y];
                        for(x = start_x + line_width; x < start_x + start_a - line_width + 1;
x++){
                                png_byte *ptr = &(row[x * 4]);
                                ptr[0] = new_red_1;
                                ptr[1] = new_green_1;
                                ptr[2] = new_blue_1;
                                ptr[3] = 255;
                        }
                }
        }

        x = 0;
       for(y = start_y + line_width; y < start_y + start_a - line_width + 1; y++){
        png_byte *row = image->row_pointers[y];
             if(start_x + line_width +  x < start_x + start_a - line_width + 1){
                png_byte *ptr = &(row[(start_x + line_width + x) * 4]);
                   ptr[0] = new_red;
                   ptr[1] = new_green;
                   ptr[2] = new_blue;
                   ptr[3] = 255;
        }
        x++;
       }

        for(int k = 0; k < line_width/2; k++){
                x = 0;
                for(y = start_y + line_width; y < start_y + start_a - line_width + 1; y++){
                        png_byte *row = image->row_pointers[y];
```

```
                    if(start_x + line_width +  x + k+1 < start_x + start_a - line_width
+ 1){
                        png_byte *ptr = &(row[(start_x + line_width + x + k+1) *
4]);
                        ptr[0] = new_red;
                        ptr[1] = new_green;
                        ptr[2] = new_blue;
                        ptr[3] = 255;
                    }
                    x++;
                }
                x = 0;
                for(y = start_y + line_width; y < start_y + start_a - line_width + 1; y++){
                    png_byte *row = image->row_pointers[y];
                    if(start_x + line_width + x - k-1 > start_x + line_width - 1){
                        png_byte *ptr = &(row[(start_x + line_width + x - k-1) *
4]);
                        ptr[0] = new_red;
                        ptr[1] = new_green;
                        ptr[2] = new_blue;
                        ptr[3] = 255;
                    }
                    x++;
                }
            }
            x = 0;
            for(y = start_y + start_a - line_width; y > start_y + line_width-1; y--){
                png_byte *row = image->row_pointers[y];
                if(start_x + line_width +  x < start_x + start_a - line_width + 1){
                    png_byte *ptr = &(row[(start_x + line_width + x) * 4]);
                    ptr[0] = new_red;
                    ptr[1] = new_green;
                    ptr[2] = new_blue;
                    ptr[3] = 255;
                }
                x++;
            }

            for(int k = 0; k < line_width/2; k++){
                x = 0;
                for(y = start_y + start_a - line_width; y > start_y + line_width-1; y--){
                    png_byte *row = image->row_pointers[y];
                     if(start_x + line_width +  x + k+1 < start_x + start_a - line_width + 1)
{
```

```
                png_byte *ptr = &(row[(start_x + line_width + x + k+1) * 4]);
                ptr[0] = new_red;
                ptr[1] = new_green;
                ptr[2] = new_blue;
                ptr[3] = 255;
            }
            x++;
        }
        x = 0;
        for(y = start_y + start_a - line_width; y > start_y + line_width-1; y--){
            png_byte *row = image->row_pointers[y];
            if(start_x + line_width + x - k-1 > start_x + line_width - 1){
                png_byte *ptr = &(row[(start_x + line_width + x - k-1) * 4]);
                ptr[0] = new_red;
                ptr[1] = new_green;
                ptr[2] = new_blue;
                ptr[3] = 255;
            }
            x++;
        }
    }
}

void rtd(struct Png *image, int condition_1, int condition_2){

    int x,y;

    if(png_get_color_type(image->png_ptr, image->info_ptr) !=
PNG_COLOR_TYPE_RGBA){
        fprintf(stderr, "Изображние должно соответствовать
PNG_COLOR_TYPE_RGBA\n");
        return;
    }

    if(condition_1 != 0 && condition_1 != 1 && condition_1 != 2){
        fprintf(stderr, "Доступны только три состоянии - 0, 1 и 2\n");
        return;
    }

    if(condition_2 != 0 && condition_2 != 255){
        fprintf(stderr, "Доступны только два цвет - под номер 0 и под
номером 255\n");
        return;
    }
```

```
	for(y = 0; y < image->height; y++){
		png_byte *row = image->row_pointers[y];
		for(x = 0; x < image->width; x++){
			png_byte *ptr = &(row[x * 4]);
			ptr[condition_1] = condition_2;
		}
	}
}

void turn(struct Png *image, int x1, int y1, int x2, int y2, int corner){

	int x,y,k;

	if(png_get_color_type(image->png_ptr, image->info_ptr) != PNG_COLOR_TYPE_RGBA){
		fprintf(stderr, "Изображение должно соответствовать PNG_COLOR_TYPE_RGBA\n");
		return;
	}

	if(x1 < 0 || y1 < 0 || x2 < 0 || y2 < 0 || x1 > image->width-1 || x2 > image->width || y1 > image->height-1 || y2 > image->height-1 || y1 > y2 || x1 > x2){
		fprintf(stderr, "Введены некорректные координаты\n");
		return;
	}

	if(corner != 90 && corner != 180 && corner != 270){
		fprintf(stderr, "Угол поворота может быть только 90, 180 и 270\n");
		return;
	}

	png_bytep *row_change;

	row_change = (png_bytep *) malloc(sizeof(png_bytep) * image->height);
	for(y = 0; y < image->height; y++){
		row_change[y] = (png_byte *) malloc(sizeof(png_bytep) * image->width);
	}

	for(y = 0; y < image->height; y++){
		png_byte *row1 = image->row_pointers[y];
		png_byte *row2 = row_change[y];
		for(x = 0; x < image->width; x++){
```

```
                png_byte *ptr1 = &(row1[x * 4]);
                png_byte *ptr2 = &(row2[x * 4]);
                ptr2[0] = ptr1[0];
                ptr2[1] = ptr1[1];
                ptr2[2] = ptr1[2];
                ptr2[3] = ptr1[3];
            }
        }

        int kx;
        int ky;
        if(corner == 90){

            if(image->width-1 < x1+y2){
            fprintf(stderr, "Часть изображения не может быть повернута, потому как выходит за рамки\n");
                return;
            }

            if(image->height-1 < y1+x2){
                fprintf(stderr, "Часть изображения не может быть повернутна, потому как выходит за рамки\n");
                return;
            }

            ky = y1;
            kx = x1;
            for(y = y2; y > y1-1; y--){
                png_byte *row2 = row_change[y];
                for(x = x1; x < x2+1; x++){
                    png_byte *row1 = image->row_pointers[ky];
                    ky++;
                    png_byte *ptr1 = &(row1[kx * 4]);
                    png_byte *ptr2 = &(row2[x * 4]);
                    ptr1[0] = ptr2[0];
                 ptr1[1] = ptr2[1];
                 ptr1[2] = ptr2[2];
                 ptr1[3] = ptr2[3];
                }
                kx++;
                ky = y1;
            }
        }
```

```
        if(corner == 180){

                for(y = y1; y < y2+1; y++){
                        png_byte *row2 = row_change[y];
                        png_byte *row1 = image->row_pointers[y1+y2-y];
                        for(x = x1; x < x2+1; x++){
                                png_byte *ptr2 = &(row2[x * 4]);
                                png_byte *ptr1 = &(row1[(x1+x2-x) * 4]);
                                ptr1[0] = ptr2[0];
                                ptr1[1] = ptr2[1];
                                ptr1[2] = ptr2[2];
                                ptr1[3] = ptr2[3];
                        }
                }
        }

        if(corner == 270){

                if(image->width-1 < x1+y2){
                        fprintf(stderr, "Часть изображения не может быть повернута,
потому как выходит за рамки\n");
                    return;
                }

                if(image->height-1 < y1+x2){
                        fprintf(stderr, "Часть изображения не может быть повернутна,
потому как выходит за рамки\n");
                    return;
                }

                    ky = y1+x2;
                    kx = x1+y2;
                    for(y = y2; y > y1-1; y--){
                        png_byte *row2 = row_change[y];
                        for(x = x1; x < x2+1; x++){
                                png_byte *row1 = image->row_pointers[ky];
                                ky--;
                                png_byte *ptr1 = &(row1[kx * 4]);
                                png_byte *ptr2 = &(row2[x * 4]);
                                ptr1[0] = ptr2[0];
                                ptr1[1] = ptr2[1];
                                ptr1[2] = ptr2[2];
                                ptr1[3] = ptr2[3];
                        }
```

```
                        kx--;
                        ky = y1+x2;
                }
        }

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

void help(){
        printf("Использование утилиты: <имя_программы> <имя входного файла> <имя выходного файла> <ключ> <параметры>\n");
        printf("ключ может быть полным (тогда используйте <--> перед ним) и сокращенным (тогда используйте <-> перед ним\n");
        printf("вам доступны следующие ключи:\n");
        printf("1. -s или --square - рисование квадрата с дигоналями с возможностью заливки, с выбором цветов заливки и линий квадрата\n");
        printf("параметры для квадрата должны указываться в следующем порядке через пробел:\n");
        printf("<координата x левого верхнего угла> <координата y левого верхнего угла> <размер стороны> <толщина линий>\n");
       printf("<красный цвет линий (от 0 до 255)> <зеленый цвет линий (от 0 до 255)> <синий цвет линий (от 0 до 255)>\n");
        printf("<заливка (1 - заливать, 0 или любое другое число - не заливать)> <красный цвет заливки (от 0 до 255, если вы не выбрали\n");
      printf("заливку, укажите 0)> <зеленый цвет заливки (от 0 до 255, если вы не выбрали заливку, укажите 0)> <синий цвет заливки\n");
      printf("(от 0 до 255, если вы не выбрали заливку укажите 0)>\n");
        printf("2. -r или --rtd - выключить или включить на максимум одну из компонент цвета картинки\n");
        printf("параметры для rtd должны указываться в следующем порядке через пробел:\n");
        printf("<выбор компоненты (от 0 до 2)> <цвет выбранной компоненты (0 либо 255)>\n");
        printf("3. -t или --turn - повернуть выбранную область на углы 90, 180 и 270\n");
        printf("параметры для поворота должны указываться в следующем порядке через пробел\n");
        printf("<координата x левого верхнего угла> <координата y левого верхнего угла> <координата x правого нижнего угла>\n");
       printf("<координата y правого нижнего угла> <угол поворота (90, 180 или 270)>\n");
```

```
	printf("4. -i или --info - распечать картинку\n");
	printf("5. -h или --help - получить справку о помоще\n");
	printf("если вы что-то введете неправильно, вам сообщат об ошибке\n");
}

void clear(struct Png *image){

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

int main(int argc, char **argv){
	const char *options = "srtih";
	struct option longOpts[] = {
		{"square", no_argument, NULL, 's'},
		{"rtd", no_argument, NULL, 'r'},
		{"turn", no_argument, NULL, 't'},
		{"info", no_argument, NULL, 'i'},
		{"help", no_argument, NULL, 'h'}
	};
	struct Png image;
       int read = read_png_file(argv[1], &image);
	if(read != 0){
		return 1;
	}
	int longIndex;
	int way = getopt_long(argc, argv, options, longOpts, &longIndex);
	while(way != -1){
		switch(way){
			case 's':
				if(argc != 15){
					fprintf(stderr, "Были введены некорректные
данные\n");
					help();
					return 0;
				}
				for(int i = 4; i < 15; i++){
				for(int j = 0; j < strlen(argv[i]); j++){
				if(isdigit(argv[i][j]) == 0){
					fprintf(stderr, "Были введены некорректные
данные\n");
```

```
                                        help();
                                    return 0;
                                }
                                }
                            }
                                square(&image, atoi(argv[4]), atoi(argv[5]), atoi(argv[6]), atoi(argv[7]), atoi(argv[8]), atoi(argv[9]), atoi(argv[10]), atoi(argv[11]), atoi(argv[12]), atoi(argv[13]), atoi(argv[14]));
                                write_png_file(argv[2], &image);
                                break;
                        case 'r':
                                if(argc != 6){
                                        fprintf(stderr, "Были введены некорректные данные\n");
                                        help();
                                        return 0;
                                }
                                for(int i = 4; i < 6; i++){
                                for(int j = 0; j < strlen(argv[i]); j++){
                                if(isdigit(argv[i][j]) == 0){
                                        fprintf(stderr, "Были введены некорректные данные\n");
                                                help();
                                            return 0;
                                        }
                                }
                            }
                                rtd(&image, atoi(argv[4]), atoi(argv[5]));
                                write_png_file(argv[2], &image);
                                break;
                        case 't':
                                if(argc != 9){
                                        fprintf(stderr, "Были введены некорректные данные\n");
                                        help();
                                        return 0;
                                }
                                for(int i = 4; i < 9; i++){
                                for(int j = 0; j < strlen(argv[i]); j++){
                                    if(isdigit(argv[i][j]) == 0){
                                            fprintf(stderr, "Были введены некорректные данные\n");
                                                help();
                                            return 0;
```

```
                        }
                    }
                }
                    turn(&image, atoi(argv[4]), atoi(argv[5]), atoi(argv[6]),
atoi(argv[7]), atoi(argv[8]));
                    write_png_file(argv[2], &image);
                    break;
                case 'i':
                    write_png_file(argv[2], &image);
                    break;
                case 'h':
                    help();
                    return 0;
                    break;
                default:
                    help();
                    return 0;
                    break;
            }
            way = getopt_long(argc, argv, options, longOpts, &longIndex);
        }
        clear(&image);
        return 0;
}
```