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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Обработка PNG файла.

Студент гр. 9382 _____ Бочаров Г.С.

Преподаватель _____ Берленко Т.А.

Санкт-Петербург

2020

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студент Бочаров Г.С.

Группа 9382

Тема работы: Обработка PNG файла.

Исходные данные: Программа должна иметь CLI или GUI.

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

Предполагаемый объем пояснительной записки:
Не менее 10 страниц.

Дата выдачи задания: 01.03.2020

Дата сдачи реферата: 27.05.2020

Дата защиты реферата: 26.05.2020

Студент гр. 9382 _____ Бочаров Г.С.

Преподаватель _____ Берленко Т.А.

АННОТАЦИЯ

В данной курсовой работе была реализованна программа, обрабатывающая поданное ей в качестве аргумента PNG изображение и соответствующий *CLI*. Освоена работа с библиотекой libpng а также с аргументами командной строки.

SUMMARY

In this course work was written a program that processes a PNG image. Implemented command line interface. The work with the libpng library was mastered. Processing with command line arguments was mastered.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель работы: создание программы, которая обрабатывает PNG изображение и управляется посредством аргументов командной строки.

Разработка велась на базе операционной системы Linux в среде разработке Clion. Для работы с PNG файлом использовалась библиотека libpng. Для реализации консольного интерфейса использовалась функция getopt.

В итоге была создана прорамма, которая принимает на вход название файла с PNG изображением и одну из описанных ниже команд. Результатом работы программы служит полученный в результате работы новый PNG файл.

1. ЗАДАНИЕ
ВАРИАНТ 22

Программа должна иметь CLI или GUI. Программа должна реализовывать весь следующий функционал по обработке png-файла

Общие сведения

- Формат картинки PNG.
- 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) Заменяет все пиксели одного заданного цвета на другой цвет. Функционал определяется:
    - Цвет, который требуется заменить
    - Цвет на который требуется заменить
- (2) Сделать рамку в виде узора. Рамка определяется:
    - Узором (должно быть несколько на выбор. Красивый узор можно получить используя фракталы)
    - Цветом
    - Шириной
- (3) Поиск всех залитых прямоугольников заданного цвета. Требуется найти все прямоугольники заданного цвета и обвести их линией. Функционал определяется:
    - Цветом искомых прямоугольников
    - Цветом линии для обводки
    - Толщиной линии для обводки

2. ХОД ВЫПОЛНЕНИЯ РАБОТЫ

2.1 Установка библиотеки libpng

Для облегчения работы с изображениями в формате PNG была установлена библиотека libpng.

2.2 Считывание , хранение и запись изображения.

Создание структур для хранения текста. Для хранения изображения и последующей работы с ним была создана структура PNG, хранящая необходиму информацию для работы с изображением (Ширина, высота, тип палитры, и т.д.).

Считывание изображения из файла. Для считывания изображения использыется функция read_png_file, принимающая на вход PNG структуру для хранения изображения и название файла для считывания изображения. В данной функции также производится обработка ошибок при считывании файла.

Считывание изображения в файл. Для записи изображения в фай используется функция write_png_file, принимающая на вход PNG структуру, содержащую информацию об изображении, и название файла для записи изображения. В данной функции также производится обработка ошибок записи файла.

2.3 Вспомогательные функции для работы с текстом.

В ходе выполнения задания потребовалось определить ряд вспомогательных функций для работы с с изображением.

1. point_in_image — функция принимает на вход структуру PNG и две координаты пикселя. Функция возвращает false если точка находится за пределами изображения.
2. set_pixel - функция принимает на вход структуру PNG, координаты пикселя и цвет. Функция меняет цвет указанного пикселя на заданный.
3. get_pixel - функция принимает на вход структуру PNG и координаты пикселя. Функция возвращает цвет заданного пикселя.
4. resize_image функция — функция меняет высоту и ширину изображения на заданные. Для этого создается новая PNG структура с заданными шириной и высотой, куда переносятся данные о цвете пикселей из старой.
5. is_eq_color — функция принимает на вход два цвета и определяет одинаковы ли эти цвета.

2.4 Функция замены всех пикселей заданного цвета другим заданным цветом. replace_color.

replace_color — функция принимает на вход изображение, цвет который требуется заменить и цвет на который требуется заменить. Проверяет цвет каждого пикселя изображения. Если цвет пикселя совпадает с цветом который нужно заменить, меняет цвет пикселя на заданный.

2.5 Функция поиска всех прямоугольников заданного цвета. find_rectangles.

find_rectangles — функция принимает на вход изображение, цвет заливки прямоугольников, цвет обводки прямоугольников и толщину обводки прямоугольников. Функция проверяет каждый пиксель, цвет которого совпадает с заданным, на предмет того, является ли он началом(левый верхний угол) прямоугольника размером больше ширины обводки. Если да, то обводит область заданным цветом.

2.6 Функция рисования рамки заданного типа и ширины. draw_border_p.

Функция draw_border_p рисует рамку заданной ширины. В зависимости от выбранного типа рамки выполняются следующие функции:

draw_border_pattern - рисует рамку используя заранее заготовленный шаблон.

draw_border_gradient — заливает рамку градиентом с центром, совпадающим с центром рамки.

draw_border_fractal — рисует рамку с фрактальным узором. Алгоритм генерации узора следующий : из левого верхнего угла картинки под углом 45 градусов пускается луч, отражающийся от краев изображения, до тех пор пока не придет в один из углов изображения. Цвет пикселя опряделяется двумя факторами : 1 — проходит ли луч через данный пиксель. 2 — удаленностью пикселя от центра изображения.

2.7 Функция обработки аргументов командной строки. get_method.

get_method — функция принимает на вход кол-во аргументов командной строки, сами эти аргументы и структуру ARG_P для записи результата обработки аргументов командной строки. Функция записывает в структуру ARG_P имя входного файла, имя выходного файла, команду пользователя и параметры этой команды.

2.8 Функция main.

В функции main осуществляется проверка корректности введенных пользователем параметров: количество символов, ключевые слова, целочисленный тип и тому подобное. Также производится проверка типа цвета исходного изображения(программа работает только с цветом типа RGBA).

Если проверка прошла успешно, вызывается одна из функций по обработке изображения, в противном случае выводится соответствующее сообщение об ошибке.

ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы была создана программа для работы с изображением PNG формата. Реализован Command Line Interface, для управления программой. Произведена обработка ряда ошибок возникающих при выполнении программы.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Интернет ресурс http://www.libpng.org/pub/png/libpng-1.2.5-manual.html#section-2

2. Грег Перри, Дин Миллер Программирование на C для начинающих, Эксмо 2014 369 с.

3. Стив Оолайн C Elements of Style M.: M&T books, 1992 456 с.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

название файла: png_work.h

```
#ifndef CW_2_2_PNG_WORK_H
#define CW_2_2_PNG_WORK_H
#include <stdlib.h>
#include <png.h>
#include <stdbool.h>

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

    int channels;

    png_structp png_ptr;
    png_infop info_ptr;
    int number_of_passes;
    png_bytep *row_pointers;
    png_byte *row_data;
} PNG;

bool read_png_file(char *, PNG *);
bool write_png_file(char *, PNG * );
void free_png(PNG * );
#endif
```

название файла: png_work.c

```
#include "png_work.h"

bool read_png_file(char *file_name, PNG *image)
{
```

```
    char header[8];

    /* open file and test for it being a png */
    FILE *fp = fopen(file_name, "rb");
    if (!fp)
    {
        puts("Some error handling: file could not be opened");
        return false;
    }

    fread(header, 1, 8, fp);
    if (png_sig_cmp(header, 0, 8))
    {
        puts("Some error handling: file is not recognized as a PNG");
        fclose(fp);
        return false;
    }

    /* initialize stuff */
      image->png_ptr = png_create_read_struct(PNG_LIBPNG_VER_STRING, NULL,
NULL, NULL);

    if (!image->png_ptr)
    {
        puts("Some error handling: png_create_read_struct failed");
        fclose(fp);
        png_destroy_read_struct(&image->png_ptr, NULL, NULL);
        return false;
    }

    image->info_ptr = png_create_info_struct(image->png_ptr);
    if (!image->info_ptr)
    {
        puts("Some error handling : png_create_info_struct failed");
        fclose(fp);
        png_destroy_read_struct(&image->png_ptr, &image->info_ptr, NULL);
        return false;
    }
```

```
    if (setjmp(png_jmpbuf(image->png_ptr)))
    {
        puts("Some error handling : error during init_io");
        fclose(fp);
        png_destroy_read_struct(&image->png_ptr, &image->info_ptr, NULL);
        return false;
    }

    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->channels = png_get_channels(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);

    /* read file */
    if (setjmp(png_jmpbuf(image->png_ptr)))
    {
        puts("Some error handling : error during read image");
        fclose(fp);
        png_destroy_read_struct(&image->png_ptr, &image->info_ptr, NULL);
        return false;
    }

     image->row_pointers = (png_bytep *) malloc(sizeof(png_bytep) * image-
>height);
    int row_byte_size = png_get_rowbytes(image->png_ptr, image->info_ptr);
    image->row_data = (png_byte *) malloc(image->height * row_byte_size);

    for (int i = 0; i < image->height; i++)
```

```
    {
            image->row_pointers[i] = (png_byte *) (image->row_data + i *
row_byte_size);
    }

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

    fclose(fp);
    return true;

}
bool write_png_file(char *file_name, PNG *image)
{
    FILE *fp = fopen(file_name, "wb");
    if (!fp)
    {
        puts("Some error handling: file could not be opened");
        return false;
    }
      image->png_ptr = png_create_write_struct(PNG_LIBPNG_VER_STRING, NULL,
NULL, NULL);

    if (!image->png_ptr)
    {
        puts("Some error handling: png_create_write_struct failed");
        fclose(fp);
        return false;
    }

    image->info_ptr = png_create_info_struct(image->png_ptr);
    if (!image->info_ptr)
    {
        puts("Some error handling: png_create_info_struct failed");
        fclose(fp);
        return false;
    }
    if (setjmp(png_jmpbuf(image->png_ptr)))
    {
        puts("Some error handling: error during init_io");
```

```
        fclose(fp);
        return false;
    }
    png_init_io(image->png_ptr, fp);

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

        puts("Some error handling: error during writing header");
        fclose(fp);
        return false;
    }

       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)))
    {
        puts("Some error handling: error during writing bytes");
        fclose(fp);
        return false;
    }
    png_write_image(image->png_ptr, image->row_pointers);

    if (setjmp(png_jmpbuf(image->png_ptr)))
    {
        puts("Some error handling: error during end of write");
        fclose(fp);
        return false;
    }
    png_write_end(image->png_ptr, NULL);
    fclose(fp);
    return true;
}
void free_png(PNG *png)
```

```
{
    free(png->row_pointers);
    free(png->row_data);
}
```

название файла: picture_work.h

```
#ifndef CW_2_2_PICTURE_WORK_H
#define CW_2_2_PICTURE_WORK_H
#include <png.h>
#include <stdbool.h>
#include "png_work.h"

bool point_in_image(PNG * image, int x, int y);
void set_pixel(PNG * image, const int , const int , const png_byte *);
png_byte *get_pixel(PNG *image, const int x, const int y);
void resize_image(PNG *image, int new_h, int new_w);
void resize_image_for_draw_border(PNG *image, int border_size);
bool is_eq_color(const png_byte *source, const png_byte *dest, int
channels);

#endif
```

название файла: picture_work.c

```
#include "picture_work.h"

bool point_in_image(PNG *image, int x, int y)
{
     return !((x < 0) || (x >= image->height) || (y < 0) || (y >= image-
>width));
}
void set_pixel(PNG *image, const int x, const int y, const png_byte *color)
{
    if (!point_in_image(image, x, y))
    {
```

```
        puts("Some error handling: x or y out of range");
        return;
    }
    int pixel_channels = image->channels;
         png_byte  *pixel  =  (void  *)  (image->row_pointers[x]  +  y  *
pixel_channels);
    for (int i = 0; i < pixel_channels; i++)
        pixel[i] = color[i];

}
png_byte *get_pixel(PNG *image, const int x, const int y)
{
    if (!point_in_image(image, x, y))
    {
        puts("x or y out of range");
        return NULL;
    }
    int pixel_channels = image->channels;
    return (image->row_pointers[x] + y * pixel_channels);
}
void resize_image(PNG *image, int new_h, int new_w)
{
    if ((new_h <= 0) || (new_w <= 0))
    {
        puts("Some error handling: new_h <= 0 or new_w <= 0");
        return;
    }
    PNG img;
    img.height = new_h;
    img.width = new_w;
    img.bit_depth = image->bit_depth;
    img.channels = image->channels;
    img.color_type = image->color_type;
    img.info_ptr = image->info_ptr;

    img.row_pointers = (png_bytep *) malloc(sizeof(png_bytep) * img.height);
    int row_byte_size = img.width * img.channels * image->bit_depth / 8;
    img.row_data = (png_byte *) malloc(img.height * row_byte_size);
```

```
    for (int i = 0; i < img.height; i++)
                img.row_pointers[i] = (png_byte *) (img.row_data + i *
row_byte_size);

    for (int i = 0; (i < image->height) && (i < img.height); i++)
        for (int j = 0; (j < image->width) && (j < img.width); j++)
        {
            set_pixel(&img, i, j, get_pixel(image, i, j));
        }
    free_png(image);

    png_set_rows(img.png_ptr, img.info_ptr, img.row_pointers);
    *image = img;
}

void resize_image_for_draw_border(PNG *image, int border_size)
{
    PNG img;
    img.height = image->height + 2 * border_size;
    img.width = image->width + 2 * border_size;
    img.bit_depth = image->bit_depth;
    img.channels = image->channels;
    img.color_type = image->color_type;
    img.info_ptr = image->info_ptr;

    img.row_pointers = (png_bytep *) malloc(sizeof(png_bytep) * img.height);
    int row_byte_size = img.width * img.channels * image->bit_depth / 8;
    img.row_data = (png_byte *) malloc(img.height * row_byte_size);

    for (int i = 0; i < img.height; i++)
                img.row_pointers[i] = (png_byte *) (img.row_data + i *
row_byte_size);

    for (int i = border_size; i < img.height - border_size; i++)
        for (int j = border_size; j < img.width - border_size; j++)
        {
               set_pixel(&img, i, j, get_pixel(image, i - border_size, j -
border_size));
        }
```

```
    free_png(image);

    png_set_rows(img.png_ptr, img.info_ptr, img.row_pointers);
    *image = img;
}

bool is_eq_color(const png_byte *source, const png_byte *dest, int channels)
{
    for (int i = 0; i < channels; i++)
        if (source[i] != dest[i])
            return false;
    return true;
}
```

название файла: CW_tasks.h

```
#ifndef CW_2_2_CW_TASKS_H
#define CW_2_2_CW_TASKS_H

#include "png_work.h"
#include "picture_work.h"
#include <math.h>

typedef struct Rectangle
{
    int x_start;
    int y_start;
    int x_end;
    int y_end;
} RECTANGLE;

typedef struct Mask
{
    int height;
    int width;
    bool **data;
} MASK;
```

```
void print_info(PNG *image);
void replace_color(PNG *image, const png_byte *source, const png_byte
*dest);
void draw_rectangle_inner_border(PNG *image, RECTANGLE *rect, const png_byte
*color, int size);
MASK get_mask(PNG *image, const png_byte *color);
void free_mask(MASK *mask);
RECTANGLE find_rectangle(MASK *mask, int x, int y);
void find_rectangles(PNG *image, const png_byte *color, const png_byte
*color1, int border_size);
void draw_pattern(PNG *image, const int *pattern, int pattern_h, int
pattern_w);
bool is_border(PNG *image, int x, int y, int border_size);
void draw_border_pattern(PNG *image, const int *pattern, int pattern_h, int
pattern_w, int border_size);
void draw_border_gradient(PNG *image, int border_size);
int max(int a, int b);
void draw_border_fractal(PNG *image, int border_size);
#endif
```

название файла: CW_tasks.c

```
#include "CW_tasks.h"

void replace_color(PNG *image, const png_byte *source, const png_byte *dest)
{
    for (int i = 0; i < image->height; i++)
    {
        for (int j = 0; j < image->width; j++)
        {
                    if (is_eq_color(get_pixel(image, i, j), source, image-
>channels))
            {
                set_pixel(image, i, j, dest);
            }
        }
    }
```

```
}

void draw_rectangle_inner_border(PNG *image, RECTANGLE *rect, const png_byte
*color, int size)
{
    if ((rect->x_start < 0) || (rect->y_start < 0) || (rect->x_end >= image-
>height) || (rect->y_end >= image->width))
    {
        puts("Some error handling : rectangle out of range");
        return;
    }
    for (int k = 0; k < size; k++)
    {
        for (int i = rect->x_start + k; i <= rect->x_end - k; i++)
        {
            set_pixel(image, i, rect->y_start + k, color);
            set_pixel(image, i, rect->y_end - k, color);
        }
        for (int i = rect->y_start + k; i <= rect->y_end - k; i++)
        {
            set_pixel(image, rect->x_start + k, i, color);
            set_pixel(image, rect->x_end - k, i, color);
        }
    }
}

MASK get_mask(PNG *image, const png_byte *color)
{
    MASK mask;
    mask.data = (bool **) malloc(sizeof(bool *) * image->height);
    for (int i = 0; i < image->height; i++)
        mask.data[i] = calloc(image->width, sizeof(bool));
    mask.height = image->height;
    mask.width = image->width;

    for (int i = 0; i < image->height; i++)
    {
        for (int j = 0; j < image->width; j++)
            if (is_eq_color(get_pixel(image, i, j), color, image->channels))
```

```
                mask.data[i][j] = 1;
    }
    return mask;
}

void free_mask(MASK *mask)
{
    for (int i = 0; i < mask->height; i++)
        free(mask->data[i]);
    free(mask->data);
}

RECTANGLE find_rectangle(MASK *mask, int x, int y)
{
    int x_start = x;
    int y_start = y;
    int x_end = x;
    int y_end = y;
    int count = 0;

    while (count < 3)
    {
        if ((count != 1) && (count != 3))
        {
            for (int i = y_start; i <= y_end; i++)
            {
                if ((x_end + 1 >= mask->height) || (mask->data[x_end + 1][i]
!= 1))
                {
                    count += 1;
                    break;
                }
            }
            if ((count != 1) && (count != 3))
                x_end = x_end + 1;
        }

        if ((count != 2) && (count != 3))
        {
```

```
            for (int i = x_start; i <= x_end; i++)
            {
                   if ((y_end + 1 >= mask->width) || (mask->data[i][y_end +
1] != 1))
                {
                    count += 2;
                    break;
                }
            }
            if ((count != 2) && (count != 3))
                y_end = y_end + 1;
        }
    }
    RECTANGLE r = {x_start, y_start, x_end, y_end};
    return r;
}

void  find_rectangles(PNG  *image,  const  png_byte  *color,  const  png_byte
*color1, int border_size)
{
    RECTANGLE r;
    MASK mask = get_mask(image, color);

    for (int i = 0; i < image->height; i++)
        for (int j = 0; j < image->width; j++)
            if (mask.data[i][j] == 1)
            {
                r = find_rectangle(&mask, i, j);
                  if ((r.x_end - r.x_start > border_size * 2) && (r.y_end -
r.y_start > border_size * 2))
                {
                    for (int k = r.x_start; k <= r.x_end; k++)
                        for (int n = r.y_start; n <= r.y_end; n++)
                            mask.data[k][n] = 0;
                             draw_rectangle_inner_border(image,  &r,  color1,
border_size);
                }
            }
    free_mask(&mask);
```

```
}


void draw_pattern(PNG *image, const int *pattern, int pattern_h, int
pattern_w)
{
    png_byte color[4] = {0, 255, 0, 255};
    for (int i = 0; i < image->height; i++)
        for (int j = 0; j < image->width; j++)
            if (pattern[(i % pattern_h) * pattern_h + j % pattern_w] == 1)
            {
                set_pixel(image, i, j, color);
            }
}

bool is_border(PNG *image, int x, int y, int border_size)
{
     if ((x <= border_size) || (x >= image->height - border_size) || (y <=
border_size) ||
        (y >= image->width - border_size))
        return true;
    return false;
}

void draw_border_pattern(PNG *image, const int *pattern, int pattern_h, int
pattern_w, int border_size)
{
    png_byte color_black[4] = {0,0,0,0};

    resize_image_for_draw_border(image, border_size);

    png_byte color[4] = {0, 255, 0, 255};
    for (int i = 0; i < image->height; i++)
        for (int j = 0; j < image->width; j++)
             if ((pattern[(i % pattern_h) * pattern_h + j % pattern_w] == 1)
&& (is_border(image, i, j, border_size)))
            {
                set_pixel(image, i, j, color);
            }
```

```
}

void draw_border_gradient(PNG *image, int border_size)
{
    resize_image_for_draw_border(image, border_size);

    png_byte color1[4] = {108, 73, 3, 255};
    png_byte color2[4] = {238, 160, 8, 255};

    for (int i = 0; i < image->height; i++)
        for (int j = 0; j <= border_size; j++)
        {
             double delta = fabs((double) border_size / 2 - j) / ((double)
border_size / 2);

               png_byte color[4] = {color1[0] + (color2[0] - color1[0]) *
delta,
                                    color1[1] + (color2[1] - color1[1]) *
delta,
                                    color1[2] + (color2[2] - color1[2]) *
delta,
                                255};
            set_pixel(image, i, j, color);
        }

    for (int i = 0; i < image->height; i++)
        for (int j = image->width - border_size; j < image->width; j++)
        {
                  double delta = fabs((double) (image->width - (double)
border_size / 2) - j) / ((double) border_size / 2);

               png_byte color[4] = {color1[0] + (color2[0] - color1[0]) *
delta,
                                    color1[1] + (color2[1] - color1[1]) *
delta,
                                    color1[2] + (color2[2] - color1[2]) *
delta,
                                255};
            set_pixel(image, i, j, color);
```

```
        }

    for (int i = 0; i < border_size; i++)
        for (int j = i; j < image->width - i; j++)
        {
             double delta = fabs((double) ((double) border_size / 2) - i) /
((double) border_size / 2);

                png_byte color[4] = {color1[0] + (color2[0] - color1[0]) *
delta,
                                     color1[1] + (color2[1] - color1[1]) *
delta,
                                     color1[2] + (color2[2] - color1[2]) *
delta,
                                255};
            set_pixel(image, i, j, color);
        }

    for (int i = image->height - border_size; i < image->height; i++)
        for (int j = image->height - i; j < image->width - (image->height -
i); j++)
        {
                   double delta = fabs((double) (image->height - (double)
border_size / 2) - i) / ((double) border_size / 2);


                png_byte color[4] = {color1[0] + (color2[0] - color1[0]) *
delta,
                                     color1[1] + (color2[1] - color1[1]) *
delta,
                                     color1[2] + (color2[2] - color1[2]) *
delta,
                                255};
            set_pixel(image, i, j, color);
        }
}

int max(int a, int b)
{
```

```
    if (a > b)
        return a;
    return b;
}

void draw_border_fractal(PNG *image, int border_size)
{
    resize_image_for_draw_border(image, border_size);

    png_byte color[4] = {0, 255, 0, 255};
    png_byte black[4] = {0, 0, 0, 228};
    int height = image->height - 1;
    int width = image->width - 1;
    if (height % 2 == width % 2)
        height--;

    for (int i = 0; i < image->height; i++)
        for (int j = 0; j < image->width; j++)
            if (is_border(image, i, j, border_size))
                set_pixel(image, i, j, black);
    int x = 1;
    int y = 1;
    int len = 10;
    int count = 0;
    int k = 1;
    int i = 1;
    int j = 1;

    while (1)
    {
        color[0] = (abs(height / 2 - x) - abs(width / 2 - y)) % 256;
        color[1] = (abs(height / 2 - x) + abs(width / 2 - y)) % 256;
        color[2] = max(abs(height / 2 - x) * 1.5, abs(width / 2 - y) * 1.5)
% 256;

        x += i;
        y += j;
        if (x == height)
        {
```

```
            i = -i;
            count++;
        }
        if (x == 0)
        {
            i = -i;
            count++;
        }
        if (y == width)
        {
            j = -j;
            count++;

        }
        if (y == 0)
        {
            j = -j;
            count++;
        }
         if (((x == 0) && (y == 0)) || ((x == 0) && (y == width)) || ((x ==
height) && (y == 0)) ||
            ((x == height) && (y == width)))
            break;
        if (count > 5000)
            break;
        len += k;
        if (abs(len) == 20)
            k = -k;
        if ((len > 7) && (is_border(image, x, y, border_size)))
            set_pixel(image, x, y, color);
    }
}
```

название файла: main.c

```
#include <string.h>
#include <getopt.h>
#include "png_work.h"
```

```
#include "picture_work.h"
#include "CW_tasks.h"

int pixel_pattern[64] = {
        0, 0, 0, 1, 1, 0, 0, 0,
        0, 0, 0, 1, 1, 0, 0, 0,
        0, 0, 0, 1, 1, 0, 0, 0,
        1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1,
        0, 0, 0, 1, 1, 0, 0, 0,
        0, 0, 0, 1, 1, 0, 0, 0,
        0, 0, 0, 1, 1, 0, 0, 0
};
enum METHOD
{
    NONE = -1, DRAW_BORDER, REPLACE_COLOR, FIND_RECTANGLES, HELP
};

typedef struct arg_p
{
    char *input_file;
    char *output_file;
    char *method_optarg;
    enum METHOD method;
} ARG_P;

bool is_RGBA(PNG *image)
{
            if   (png_get_color_type(image->png_ptr,   image->info_ptr)   ==
PNG_COLOR_TYPE_RGB)
    {
        puts("Some error handling: input file is PNG_COLOR_TYPE_RGB but must
be PNG_COLOR_TYPE_RGBA");
        return false;
    }

            if   (png_get_color_type(image->png_ptr,   image->info_ptr)   !=
PNG_COLOR_TYPE_RGBA)
    {
```

```
            puts("Some error handling: color_type of input file must be
PNG_COLOR_TYPE_RGBA");
        return false;
    }
    return true;
}

void print_help()
{
                                                                    puts("-h
Display this information\n");
                                                                puts("--help
Display this information\n");
                                          puts("-i          <source_file_name>
Enter source file\n");
                                            puts("-o          <dest_file_name>
Enter destination file\n");
                                  puts("-B        \"<fill_type>        <size>\"
Draw border\n");
                         puts("--draw_border      \"<fill_type>      <size>\"
Draw border\n");
        puts("-S  \"<source_color:[r  g  b  a]>  <final_color[r  g  b  a]>\"
Replace color1 with color2\n");
     puts("--substitute_color \"<source_color:[r g b a]> <final_color[r g b
a]>\"     Replace color1 with color2\n");
               puts("-R    \"<fill_color:[r    g    b    a]>    border_size\"
Find and outline rectangles filled with a given color\n");
        puts("--find_rectangles  \"<fill_color:[r  g  b  a]>  border_size\"
Find and outline rectangles filled with a given color\n");
}

void draw_border_p(PNG *image, char *arg)
{
    char fill_type[20] = {0};
    int border_size = 0;
    if (sscanf(arg, "%19s %d", fill_type, &border_size) != 2)
    {
        printf("<%s>", fill_type);
        puts("Some error handling: incorrect fill_type or bordr_size");
```

```
        return;
    }


        if  (((strcmp(fill_type,  "fractal")  !=  0)  &&  (strcmp(fill_type,
"gradient") != 0) &&
         (strcmp(fill_type, "pattern") != 0)))
    {
        puts("Some error handling: no such fill_type");
        return;
    }

    if ((border_size > 1000))
    {
        puts("Some error handling: border_size too big");
        return;
    }
    if (border_size <= 0)
    {
        puts("Some error handling: border_size <= 0");
        return;
    }

    if (strcmp(fill_type, "fractal") == 0)
    {
        draw_border_fractal(image, border_size);
    }
    if (strcmp(fill_type, "gradient") == 0)
    {
        draw_border_gradient(image, border_size);
    }
    if (strcmp(fill_type, "pattern") == 0)
    {
        draw_border_pattern(image, pixel_pattern, 8, 8, border_size);
    }
}

void replace_color_p(PNG *image, char *arg)
{
```

```
    int color1[4];
    int color2[4];

     if (sscanf(arg, "[%d %d %d %d] [%d %d %d %d]", &color1[0], &color1[1],
&color1[2], &color1[3],
                &color2[0], &color2[1], &color2[2], &color2[3]) != 8)
    {
        puts("Some error handling: incorrect RGBA color");
        return;
    }

    png_byte clr1[4];
    png_byte clr2[4];

    for (int i = 0; i < 4; i++)
    {
          if ((color1[i] >= 256) || (color1[i] < 0) || (color2[i] < 0) ||
(color2[i] >= 256))
        {
            puts("Some error handling: RGBA parameters should be [0, 255]");
            return;
        }
        clr1[i] = (png_byte) color1[i];
        clr2[i] = (png_byte) color2[i];
    }
    replace_color(image, clr1, clr2);
}

void find_rectangles_p(PNG *image, char *arg)
{
    int rect_color[4];
    int border_color[4];
    int border_size = 0;
      if (sscanf(arg, "[%d %d %d %d] [%d %d %d %d] %d", &rect_color[0],
&rect_color[1], &rect_color[2], &rect_color[3],
                      &border_color[0], &border_color[1], &border_color[2],
&border_color[3], &border_size) != 9)
    {
        puts("Some error handling: Invalid color or border_size");
```

```
        return;
    }
    if (border_size <= 0)
    {
        puts("border_size <= 0");
    }
     if ((border_size * 2 >= image->width) || (border_size * 2 >= image-
>height))
    {
        puts("border_size too big");
        return;
    }

    png_byte rect_clr[4];
    png_byte border_clr[4];
    for (int i = 0; i < 4; i++)
    {
                if ((rect_color[i] >= 256) || (rect_color[i] < 0) ||
(border_color[i] < 0) || (border_color[i] >= 256))
        {
            puts("Some error handling: RGBA parameters should be [0, 255]");
            return;
        }
        rect_clr[i] = (png_byte) rect_color[i];
        border_clr[i] = (png_byte) border_color[i];
    }
    find_rectangles(image, rect_clr, border_clr, border_size);
}

void get_method(int argc, char **argv, ARG_P *arg_p)
{
    int opt;

    char *opts = "i:o:hB:R:S:";

    struct option long_options[] = {
            {"help",            no_argument,       NULL, 'h'},
            {"draw_border",     required_argument, NULL, 'B'},
            {"input",           required_argument, NULL, 'i'},
```

```
        {"output",          required_argument, NULL, 'o'},
        {"find_rectangles", required_argument, NULL, 'R'},
        {"replace_color",   required_argument, NULL, 'S'},
        {NULL, 0,                              NULL, 0}
};


int long_index;

opt = getopt_long(argc, argv, opts, long_options, &long_index);
while (opt != -1)
{
    switch (opt)
    {
        case 'i':
            arg_p->input_file = optarg;
            break;
        case 'o':
            arg_p->output_file = optarg;
            break;
        case 'h':
            if (arg_p->method != NONE)
            {
                puts("-h INVALID_ARGUMENT");
                return;
            }
        case 'P':
            if (arg_p->method != NONE)
            {
                puts("-P INVALID_ARGUMENT");
                return;
            }
            arg_p->method = HELP;
            break;
        case 'B':
            if (arg_p->method != NONE)
            {
                puts("-B INVALID_ARGUMENT");
                return;
```

```
                }
                arg_p->method = DRAW_BORDER;
                arg_p->method_optarg = optarg;
                break;
            case 'S':
                if (arg_p->method != NONE)
                {
                    puts("-S INVALID_ARGUMENT");
                    return;
                }
                arg_p->method = REPLACE_COLOR;
                arg_p->method_optarg = optarg;
                break;
            case 'R':
                if (arg_p->method != NONE)
                {
                    perror("-R INVALID_ARGUMENT");
                    return;
                }
                arg_p->method = FIND_RECTANGLES;
                arg_p->method_optarg = optarg;
                break;
            default:
                puts("Some error handling: Invalid flag");
                break;
        }
        opt = getopt_long(argc, argv, opts, long_options, &long_index);
    }
}

void set_negativ(PNG *image)
{
    png_byte color[4] = {0, 0, 0, 0};
    for (int i = 0; i < image->height; i++)
        for (int j = 0; j < image->width; j++)
        {
            color[0] = 255 - get_pixel(image, i, j)[0];
            color[1] = 255 - get_pixel(image, i, j)[1];
            color[2] = 255 - get_pixel(image, i, j)[2];
```

```
            color[3] = get_pixel(image, i, j)[3];
            set_pixel(image, i, j, color);
        }

}

void set_BW(PNG *image, int brightness)
{

    if(brightness <= 0)
    {
        puts("Invalid brightness");
        return;
    }

    int separator = 255 / brightness * 3;

    png_byte color[4] = {0, 0, 0, 255};
    for (int i = 0; i < image->height; i++)
        for (int j = 0; j < image->width; j++)
        {
              if ((get_pixel(image, i, j)[0] + get_pixel(image, i, j)[1] +
get_pixel(image, i, j)[2]) >= separator)
            {
                color[0] = 0;
                color[1] = 0;
                color[2] = 0;
                color[3] = get_pixel(image, i, j);
            } else
            {
                color[0] = 255;
                color[1] = 255;
                color[2] = 255;
                color[3] = get_pixel(image, i, j);
            }
            set_pixel(image, i, j, color);
        }
}
```

```
int main(int argc, char **argv)
{
    ARG_P arg_p = {NULL, NULL, NULL, NONE};

    PNG image;

    get_method(argc, argv, &arg_p);

    if (arg_p.method == HELP)
    {
        print_help();
        return 0;
    }
    if (arg_p.input_file == NULL)
    {
        puts("Some error handling: input_file not indicated");
        return -1;
    }
    if (arg_p.output_file == NULL)
    {
        arg_p.output_file = "out.png";
    }

    if (!read_png_file(arg_p.input_file, &image))
        return -1;

    if (!is_RGBA(&image))
        return -1;

    switch (arg_p.method)
    {
        case DRAW_BORDER:
            draw_border_p(&image, arg_p.method_optarg);
            break;
        case REPLACE_COLOR:
            replace_color_p(&image, arg_p.method_optarg);
            break;
        case FIND_RECTANGLES:
```

```
            find_rectangles_p(&image, arg_p.method_optarg);
            break;
        default:
            break;
    }

    if (!write_png_file(arg_p.output_file, &image))
        return -1;

    free_png(&image);
    return 0;
}
```