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. 3344 _____ Волков А.А.

Преподаватель _____ Глазунов С.А.

Санкт-Петербург

2024

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студент Волков А.А.

Группа 3344

Тема работы: обработка PNG файлов.

Исходные данные:

Создать консольную утилиту для взаимодействия с файлами PNG формата на языке C. Реализовать интерфейс при помощи getopt.

Содержание пояснительной записки:

«Содержание», «Введение», «Задание работы», «Ход выполнения работы», «Заключение», «Список использованных источников», «Приложение А. Тестирование», «Приложение Б. Исходный код программы»

Предполагаемый объем пояснительной записки:

Не менее 20 страниц.

Дата выдачи задания: 18.03.2024

Дата сдачи реферата: 21.05.2024

Дата защиты реферата: 23.05.2024

Студент _____ Волков А.А.

Преподаватель _____ Глазунов С.А.

АННОТАЦИЯ

Программа представляет из себя консольную утилиту по взаимодействию с PNG изображениями, выбор требуемых действий происходит посредством опций в виде флагов и аргументов. Интерфейс реализуется с использованием библиотеки getopt. Используемый язык программирования C.

SUMMARY

The program is a console utility for interacting with PNG images, the selection of the required action implemented by options in the form of flags and arguments. The interface is implemented using the getopt library. The programming language used is C.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель: создать программу на языке программирования C, способную считывать, хранить, записывать PNG файлы, а также реализующую консольный интерфейс.

Для достижения поставленной цели необходимо выполнить следующие задачи:

1. Разработать логику считывания, хранения, записи файлов.
2. Реализовать требуемые в заданиях функции обработки полученных данных из изображения.
3. Разобраться в работе с библиотекой getopt и реализовать интерфейс с ее помощью.
4. Протестировать работоспособность программы на разных тестовых сценариях.
5. Написать Makefile для удобной сборки проекта.

1. ЗАДАНИЕ РАБОТЫ

Вариант 11

Программа обязательно должна иметь CLI.

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

Общие сведения:

- Формат картинки PNG
- 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. Разделяет изображение на **N*M** частей. Флаг для выполнения данной операции: `**--split**`. Реализация: провести линии заданной толщины. Функционал определяется:

  - Количество частей по “оси” Y. Флаг `**--number_x**`. На вход принимает число больше 1
  - Количество частей по “оси” X. Флаг `**--number_y**`. На вход принимает число больше 1
  - Толщина линии. Флаг `**--thickness**`. На вход принимает число больше 0

- Цвет линии. Флаг **`--color`** (цвет задаётся строкой `rrr.ggg.bbb`, где rrr/ggg/bbb – числа, задающие цветовую компоненту. пример `--color 255.0.0` задаёт красный цвет)

2. 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`)

3. Сделать рамку в виде узора. Флаг для выполнения данной операции: **`--ornament`**. Рамка определяется:

- Узором. Флаг **`--pattern`**. Обязательные значения: rectangle и circle, semicircles. Также можно добавить свои узоры (красивый узор можно получить используя фракталы). Подробнее здесь: https://se.moevm.info/doku.php/courses:programming:cw_spring_ornament

- Цветом. Флаг `**--color**` (цвет задаётся строкой `rrr.ggg.bbb`, где rrr/ggg/bbb – числа, задающие цветовую компоненту. пример `--color 255.0.0` задаёт красный цвет)
- Шириной. Флаг `--thickness`. На вход принимает число больше 0
- Количеством. Флаг `--count`. На вход принимает число больше 0
- При необходимости можно добавить дополнительные флаги для необозначенных узоров

4. Поворот изображения (части) на 90/180/270 градусов. Флаг для выполнения данной операции: `**--rotate**`. Функционал определяется

- Координатами левого верхнего угла области. Флаг `**--left_up**`, значение задаётся в формате `left.up`, где left – координата по x, up – координата по y
- Координатами правого нижнего угла области. Флаг `**--right_down**`, значение задаётся в формате `right.down`, где right – координата по x, down – координата по y
- Углом поворота. Флаг `**--angle**`, возможные значения: `90`, `180`, `270`

Каждую подзадачу следует вынести в отдельную функцию, функции сгруппировать в несколько файлов (например, функции обработки текста в один, функции ввода/вывода в другой). Сборка должна осуществляться при помощи make и Makefile или другой системы сборки

2. ХОД ВЫПОЛНЕНИЕ РАБОТЫ

2.1. Считывание данных

Для считывания данных изображения используется функция **load_image()**. Эта функция отвечает за открытие файла изображения в бинарном режиме, чтение его содержимого с использованием библиотеки **libpng**, и создание структуры данных изображения, содержащей путь к файлу, его размеры, тип цвета, глубину цвета и массив байтов, представляющий пиксели изображения.

Функция **load_image()** начинает с открытия указанного файла в бинарном режиме с помощью стандартной библиотеки ввода-вывода C (stdio.h). Затем она создаёт структуры **png_struct и png_info**, которые используются библиотекой libpng для чтения PNG-изображений.

Затем функция инициализирует чтение данных PNG с помощью **png_init_io()**, указывая файл, который был открыт, и использует **png_read_png()** для чтения информации о PNG-изображении.

Информация, прочитанная из файла PNG, включает в себя ширину, высоту, тип цвета и глубину цвета изображения. После чтения этой информации выделяется память для хранения массива байтов, представляющего пиксели изображения.

2.2. Структуры данных

В файле **image.h** объявлены следующие структуры данных:

- **color**: Структура для представления цвета RGB.
- **point**: Структура для представления координат точки.
- **pixel**: Структура, объединяющая координаты пикселя и указатель на его байты в массиве изображения.

- **image**: Структура, описывающая изображение, включая его размеры, тип цвета, глубину цвета и массив байтов, представляющий пиксели.
- **selection**: Структура, представляющая выделенную область изображения с указанием её начальной и конечной точек.

2.3. Функции рисования и обработки пикселей

1) put_pixel(pixel* pix, color* clr)

Эта функция устанавливает цвет указанного пикселя. Принимает на вход указатель на структуру пикселя (pixel) и указатель на структуру цвета (color). Функция просто присваивает значения компонентам цвета из структуры color байтам пикселя.

2) get_pixel(image *img, double x, double y)

Данная функция возвращает указатель на пиксель изображения по указанным координатам. Принимает на вход указатель на структуру изображения (image) и координаты x и y. Функция проверяет, что координаты находятся в пределах изображения, затем вычисляет индексы массива байтов изображения и возвращает указатель на соответствующий пиксель.

3) pixel_to_color(pixel *pix)

Эта функция преобразует структуру пикселя (pixel) в структуру цвета (color). Принимает на вход указатель на структуру пикселя и возвращает указатель на новую структуру цвета, содержащую компоненты цвета из пикселя.

2.4. Основные функции работы с изображением

1) Рисование прямоугольника (rectangle()): функция принимает выделенную область, ширину обводки, цвет обводки и цвет заливки. Внутри выделенной области функция проходит через каждый пиксель. Если

пиксель находится внутри границы прямоугольника, то красим его цветом заливки (если он передан), иначе – цветом обводки.

2) Рисование окружности (draw_circle()): функция рисует окружность на изображении с заданными параметрами: центром, радиусом и толщиной линии. Она проходит через каждый пиксель в квадрате, содержащем окружность и проверяет расстояние от текущего пикселя до центра окружности: если оно находится в пределах толщины линии, закрашивает пиксель.

3) Разделение изображения на N M частей (split()): функция разделяет на N горизонтальных и M вертикальных частей, отрисовывая границы сетки с заданной шириной линии.

4) Поворот изображения на 90 / 180 / 270 градусов (rotate()): функция поворачивает изображение на заданный угол вокруг центра выделенной области. С помощью функции turn() поворачивается заданная область, затем после этого происходит копирование пикселей из исходного изображения в новое в соответствии с новыми координатами, которые рассчитываются на основе угла поворота с помощью матрицы поворота.

2.5. Функция main

В начале функции main определены строки short_options и long_options, которые описывают короткие и длинные опции соответственно. Эти строки используются в функции getopt_long, чтобы определить, какие опции приняты и какие аргументы им переданы.

В этом цикле while происходит разбор аргументов командной строки. Функция getopt_long в каждой итерации возвращает значение одной опции.

Это значение затем используется в switch для определения, какая опция была передана, и соответствующего действия.

После обработки всех опций проверяется наличие ошибок. В случае обнаружения ошибки вызывается функция raise_error, которая выводит сообщение об ошибке и завершает программу.

2.6. Makefile

С целью сделать код более читабельным и модульным был написан Makefile. Созданы заголовочные файлы image.h, draw.h, tools.h которые хранят основные декларации функций и структур. Остальной код разделен на image.c, draw.c, tools.c, main.c. При вызове make утилиты создается исполняемый файл с названием cw.

ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы была создана программа (исходный код в Приложении Б), обрабатывающая PNG файлы. Разработана логика их хранения. Реализовано взаимодействия с пользователем при помощи консольного интерфейса с использованием getopt. Реализованы и протестированы (Приложение А) собственные методы обработки изображения.

Кроме того, выполнение работы закрепило написания make-файла, который является мощным инструментом при работе с достаточно объемным массивом кода. Считаю, что поставленные цели были успешно достигнуты.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Алгоритм Брезенхэма [Wikipedia.org]. URL: https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%91%D1%80%D0%B5%D0%B7%D0%B5%D0%BD%D1%85%D1%8D%D0%BC%D0%B0 (Дата обращения: 21.05.2023)

2. Руководство по языку программирования C [Metanit.com]. URL: https://metanit.com/c/tutorial/ (Дата обращения: 15.05.2023)

3. libpng. URL: http://www.libpng.org/pub/png/libpng-1.2.5-manual.html (Дата обращения: 15.05.2023)

ПРИЛОЖЕНИЕ А
ТЕСТИРОВАНИЕ

Test 1 (используемое изображение 500 x 266)

Test 1.1 (рис.1): Пользователь запрашивает разделить изображение на 4 * 5 частей с заданным цветом и толщиной линии. Результат записывается в output.bmp.

Рисунок 1 - Test 1.1

Test 1.2 (рис.2): Если пользователь не вводит какой-то аргумент, то печатается сообщение об ошибке передачи аргументов.

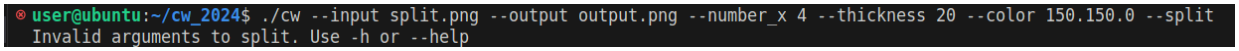

Рисунок 2 - Test 1.2

Test 2 (используемое изображение 500 x 375)

Test 2.1 (рис.4): Пользователь запрашивает повернуть выделенную область на 90 градусов. Результат сохраняется в файл output.png.

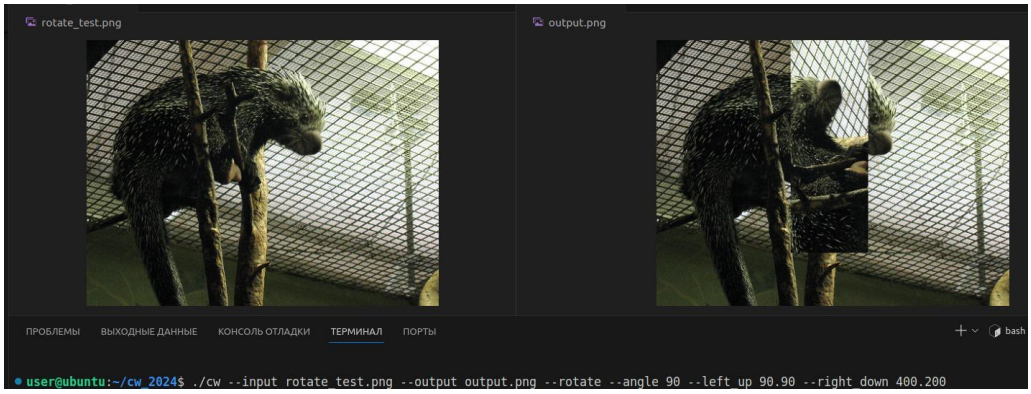

Рисунок 4 - Test 2.1

Test 2.2 (рис.5): Аналогично предыдущему примеру выбирается пользователь запрашивает повернуть выделенную область на 270 градусов, программа корректно работает.

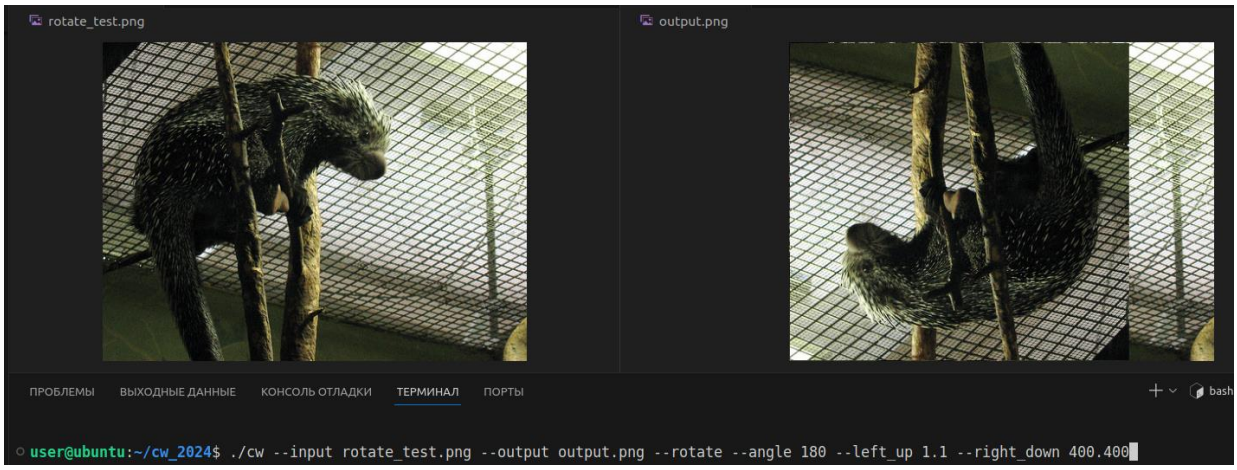

Рисунок 5 - Test 2.2

Test 2.3 (рис.6): Некорректно указывается угол поворота, программа выводит сообщение и завершает процесс.

Рисунок 6 - Test 2.3

Test 3 (используемое изображение 500 x 395)

Test 3.1 (рис.7): Выбирается рисование узора в виде прямоугольников, указывается количество, толщина линий и цвет. Результат сохраняется в файл output.png

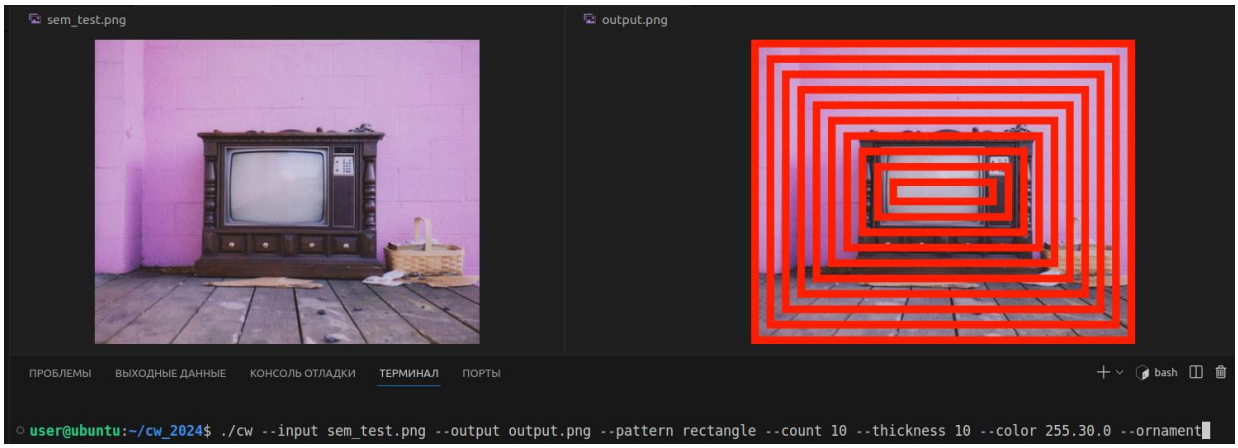

Рисунок 7 - Test 3.1

Test 3.2 (рис. 8): Узор в виде круга.

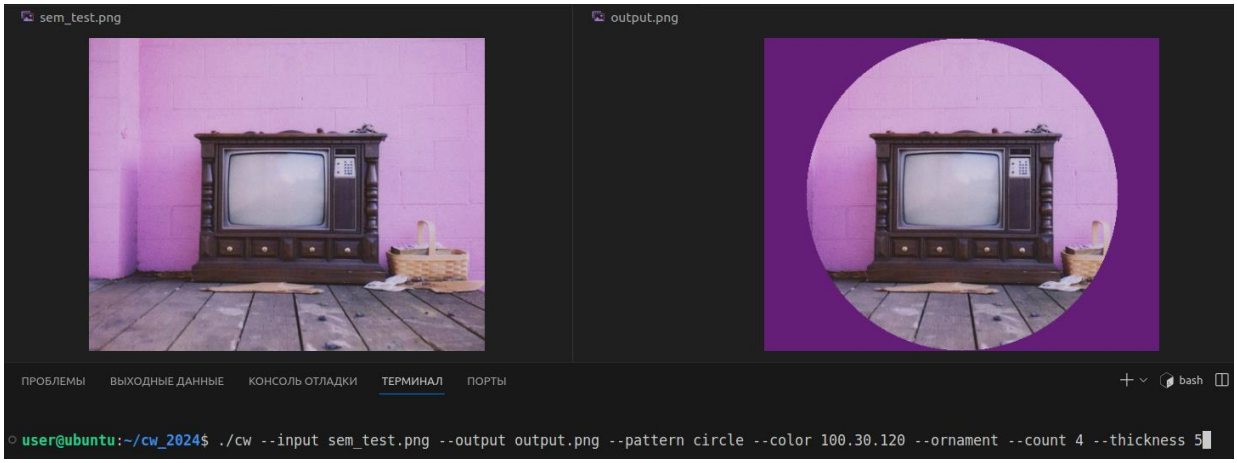

Рисунок 8 - Test 3.2

Test 3.3 (рис. 9): Узор в виде полуокружностей.

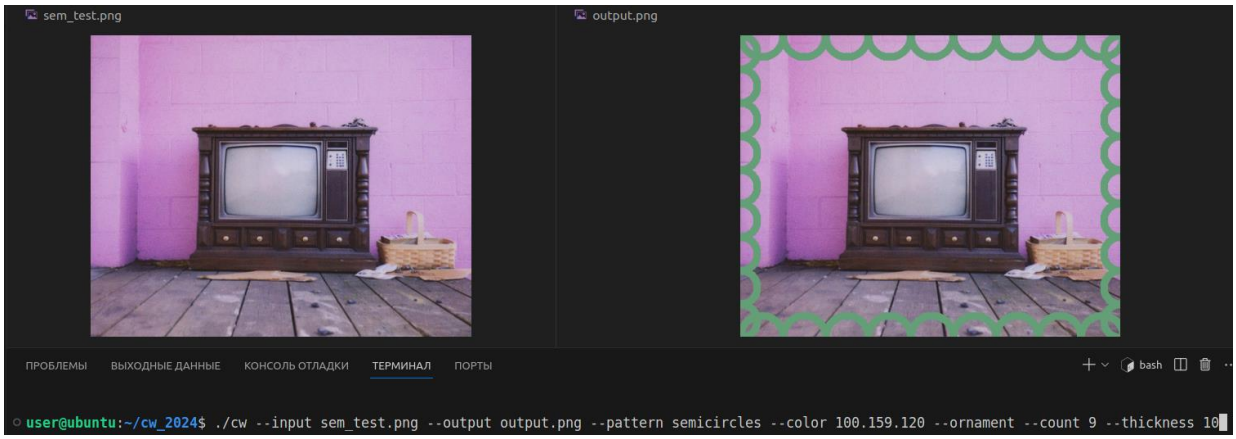

Рисунок 9 - Test 3.3

Test 4 (используемое изображение 500 x 335)

Test 4.1 (рис. 10): Выбирается построение прямоугольника с данной толщиной, цветом линии и координатами левого верхнего и правого нижнего угла, заливка не требуется. Результат записан в output.png.

Рисунок 10 - Test 4.1

Test 4.2 (рис. 11): Строится прямоугольник аналогичный прошлом тесту, но выбирается заливка нужного цвета. Программа работает корректно.

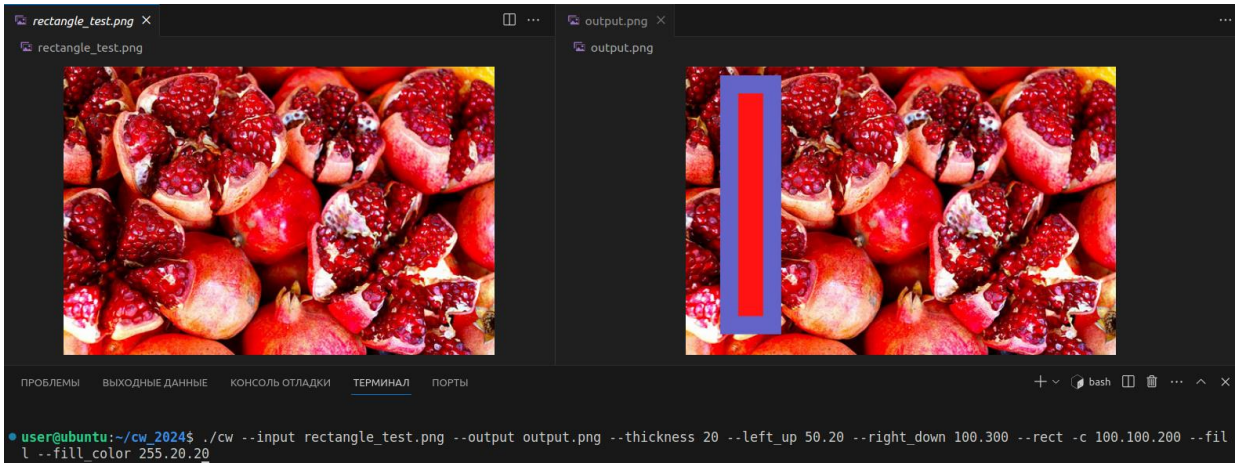

Рисунок 11 - Test 4.2

Test 4.3 (рис.12): Вводится некорректная толщина линии, печатается сообщение об ошибке.

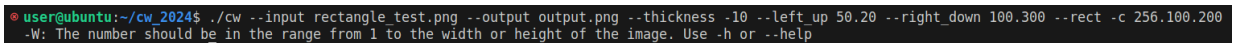

Рисунок 12 - Test 4.3

ПРИЛОЖЕНИЕ Б.
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: Makefile

```
all: main.o image.o draw.o tools.o
 gcc main.o image.o draw.o tools.o -o cw -lpng -lm

main.o: main.c image.h
 gcc -c main.c -o main.o

image.o: image.c image.h
 gcc -c image.c -o image.o

draw.o: draw.c draw.h image.h
 gcc -c draw.c -o draw.o

tools.o: tools.c tools.h draw.h image.h
 gcc -c tools.c -o tools.o

clear:
 rm -f main *.o
```

Название файла: image.h

```
#pragma once

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <png.h>

#define MAX(X, Y) (((X) > (Y)) ? (X) : (Y))
#define MIN(X, Y) (((X) < (Y)) ? (X) : (Y))

#define PIXSIZE 3 * sizeof(png_byte)

#define STR_ERR_OPEN "Invalid file\n"
#define STR_ERR_PNG_STRUCT "Failed to create png struct\n"
#define STR_ERR_READ_PNG "Failed to read png file\n"
#define STR_ERR_WRITE_PNG "Failed to write png file\n"
#define  STR_ERR_COLOR  "RGB  color  model  and  8-bit  depth  are
expected\n"

// Ошибки чтения-записи PNG файлов
#define ERR_OPEN 101
#define ERR_PNG_STRUCT 102
#define ERR_WRITE_PNG 103
#define ERR_PNGLIB_ERROR 104

// Ошибки изменения изображения
#define ERR_INVALID_SIZE 201

typedef png_uint_32 uint;
```

```
// Структура цвета RGB
typedef struct color_struct
{
    double R;
    double G;
    double B;
} color;

// Структура точки
typedef struct point_struct
{
    double x;
    double y;
} point;

// Структура пикселя (точка + указатель на место в массиве байт)
typedef struct pixel_struct
{
    point *pos;
    png_bytep byte;
} pixel;

// Структура изображения
typedef struct image_struct
{
    char *path;
    uint width;
    uint height;
    png_byte color_type;
    png_byte bit_depth;
    png_bytepp bytes;
} image;

// Структура рабочей области
typedef struct selection_struct
{
    point *start;
    point *end;
    image *img;
} selection;

color *init_color(double R, double G, double B);
point *init_point(double x, double y);
pixel *init_pixel(point *pos, png_bytep byte);
selection *init_selection(image *img, point *start, point *end);
image* init_image(uint width, uint height, color* clr);
selection *select_all(image *img);
image *load_image(char *path);
image *copy_image(image *img);
int save_image(image *img, char *path);
pixel *read_selection(selection *slct);
void free_pixel(pixel** pix);
void free_image(image **img);
void free_selection(selection** slct);
```

Название файла: image.c

```
#include "image.h"

// Создание структуры color
color *init_color(double R, double G, double B)
{
    if ((R < 0 || R > 255) || (G < 0 || G > 255) || (B < 0 || B
> 255))
        return NULL;
    color *clr = (color *)calloc(1, sizeof(color));
    clr->R = R;
    clr->G = G;
    clr->B = B;
    return clr;
}

// Создание структуры point
point *init_point(double x, double y)
{
    point *pnt = (point *)calloc(1, sizeof(point));
    pnt->x = x;
    pnt->y = y;
    return pnt;
}

// Создание структуры pixel
pixel *init_pixel(point *pos, png_bytep byte)
{
    pixel *pxl = (pixel *)calloc(1, sizeof(pixel));
    pxl->pos = pos;
    pxl->byte = byte;
    return pxl;
}

// Очистка памяти, выделенной под пиксель
void free_pixel(pixel **pix)
{
    if (pix == NULL || *pix == NULL)
        return;
    if ((*pix)->pos != NULL) {
        free((*pix)->pos);
    }
    free(*pix);
    *pix = NULL;
}

// Создание структуры selection
selection *init_selection(image *img, point *start, point *end)
{
    point  *new_start  =  init_point(MIN(MIN(start->x,  end->x),
img->width - 1),
                                     MIN(MIN(start->y,     end->y),
img->height - 1));
    point *new_end = init_point(MIN(MAX(start->x, end->x), img-
>width - 1),
```

```
                                        MIN(MAX(start->y,     end->y),
img->height - 1));
          selection *slct = (selection *)calloc(1, sizeof(selection));
          slct->start = new_start;
          slct->end = new_end;
          slct->img = img;
          return slct;
      }

      // Выделение всего изображения
      selection *select_all(image *img)
      {
          selection   *slct   =   init_selection(img,   init_point(0,0),
init_point(img->width - 1, img->height - 1));
          return slct;
      }

      // Освобождение памяти из под выделенной области
      void free_selection(selection **slct)
      {
          if (slct == NULL || *slct == NULL)
              return;

          if ((*slct)->start != NULL)
              free((*slct)->start);

          if ((*slct)->end != NULL)
              free((*slct)->end);
          free(*slct);
          *slct = NULL;
      }

      // Создание нового изображения
      image *init_image(uint width, uint height, color *clr)
      {
          image *img = malloc(sizeof(image));
          img->path = NULL;
          img->width = width;
          img->height = height;
          img->color_type = PNG_COLOR_TYPE_RGB;
          img->bit_depth = 8;

          // Выделение памяти под массив байтов
          png_bytepp      bytes      =      (png_bytepp)calloc(height,
sizeof(png_bytep));
          for (uint y = 0; y < height; y++) {
              // Выделение памяти под байты для каждого пикселя
              bytes[y]     =     (png_bytep)calloc(width     *     3,
sizeof(png_byte));
              for (uint x = 0; x < width; x++) {
                  png_bytep byte = bytes[y] + 3 * x;
                  if (clr) {
                      byte[0] = (png_byte)(MIN(MAX(0, clr->R), 255));
                      byte[1] = (png_byte)(MIN(MAX(0, clr->G), 255));
                      byte[2] = (png_byte)(MIN(MAX(0, clr->B), 255));
                  }
```

```
            }
        }
        img->bytes = bytes;
        return img;
    }

    // Выгрузка изображения из файла
    image *load_image(char *path)
    {
        // Открытие файла в двоичном режиме для чтения
        FILE *f = fopen(path, "rb");
        if (!f) {
            fprintf(stderr, STR_ERR_OPEN);
            return NULL;
        }

        // Создание структуры для чтения PNG изображения
        png_structp                    png_ptr                    =
png_create_read_struct(PNG_LIBPNG_VER_STRING, NULL, NULL, NULL);
        if (png_ptr == NULL) {
            fprintf(stderr, STR_ERR_PNG_STRUCT);
            fclose(f);
            return NULL;
        }

        //  Создание  структуры  для  хранения  информации  о  PNG
изображении
        png_infop info_ptr = png_create_info_struct(png_ptr);
        if (info_ptr == NULL) {
            fprintf(stderr, STR_ERR_PNG_STRUCT);
            // Освобождение памяти, занятой структурой png_structp
            png_destroy_read_struct(&png_ptr, NULL, NULL);
            fclose(f);
            return NULL;
        }

        // Установка точки возврата для обработки ошибок при чтении
        if (setjmp(png_jmpbuf(png_ptr))) {
            fprintf(stderr, STR_ERR_READ_PNG);
            png_destroy_read_struct(&png_ptr, &info_ptr, NULL);
            fclose(f);
            return NULL;
        }

        // Инициализация структуры png_structp для работы с файлом
        png_init_io(png_ptr, f);
        // Чтение информации о PNG изображении в структуру png_info
        png_read_info(png_ptr, info_ptr);

        uint width = png_get_image_width(png_ptr, info_ptr);
        uint height = png_get_image_height(png_ptr, info_ptr);
        png_byte color_type = png_get_color_type(png_ptr, info_ptr);
        png_byte bit_depth = png_get_bit_depth(png_ptr, info_ptr);

        if (color_type != PNG_COLOR_TYPE_RGB || bit_depth != 8) {
            fprintf(stderr, STR_ERR_COLOR);
```

```
            png_destroy_read_struct(&png_ptr, &info_ptr, NULL);
            fclose(f);
            return NULL;
        }

        // Выделение памяти под массив байтов (указатели на строки)
        png_bytepp    bytes    =    (png_bytepp)malloc(height    *
sizeof(png_bytep));
        for (uint y = 0; y < height; y++) {
            // Выделение памяти для каждой строки
            bytes[y]  =  (png_bytep)malloc(png_get_rowbytes(png_ptr,
info_ptr));
        }

        // Чтение изображения в массив байтов
        png_read_image(png_ptr, bytes);
        png_destroy_read_struct(&png_ptr, &info_ptr, NULL);
        fclose(f);
        image *img = malloc(sizeof(image));
        img->path = path;
        img->width = width;
        img->height = height;
        img->color_type = color_type;
        img->bit_depth = bit_depth;
        img->bytes = bytes;
        return img;
    }

    // Запись изображения в файл
    int save_image(image *img, char *path)
    {
        // Открытие файла для записи в бинарном режиме
        FILE *f = fopen(path, "wb");
        if (!f) {
            fprintf(stderr, STR_ERR_OPEN);
            return ERR_OPEN;
        }

        // Создание структуры png_structp для записи PNG-изображения
        png_structp                    png_ptr                    =
png_create_write_struct(PNG_LIBPNG_VER_STRING, NULL, NULL, NULL);
        if (png_ptr == NULL) {
            fprintf(stderr, STR_ERR_PNG_STRUCT);
            fclose(f);
            return ERR_PNG_STRUCT;
        }

        // Создание структуры png_info для хранения информации о
PNG-изображении
        png_infop info_ptr = png_create_info_struct(png_ptr);
        if (info_ptr == NULL) {
            fprintf(stderr, STR_ERR_PNG_STRUCT);
            fclose(f);
            return ERR_PNG_STRUCT;
        }
```

```
        // Установка точки возврата для обработки ошибок при чтении
        if (setjmp(png_jmpbuf(png_ptr))) {
            fprintf(stderr, STR_ERR_WRITE_PNG);
            png_destroy_write_struct(&png_ptr, &info_ptr);
            fclose(f);
            return ERR_PNGLIB_ERROR;
        }

        // Инициализация структуры png_structp для работы с файлом
        png_init_io(png_ptr, f);

        // Установка заголовка PNG изображения
        png_set_IHDR(png_ptr,  info_ptr,  img->width,  img->height,
img->bit_depth,
                     PNG_COLOR_TYPE_RGB,         PNG_INTERLACE_NONE,
PNG_COMPRESSION_TYPE_DEFAULT,
                     PNG_FILTER_TYPE_DEFAULT);
        // Запись информации о PNG изображении
        png_write_info(png_ptr, info_ptr);
        // Запись пиксельных данных в массив
        png_write_image(png_ptr, img->bytes);
        // Завершение процесса записи изображения
        png_write_end(png_ptr, NULL);

        png_destroy_write_struct(&png_ptr, &info_ptr);
        fclose(f);
        return 0;

    }

    // Копирование изображения
    image *copy_image(image *img)
    {
        image *copy = malloc(sizeof(image));
        copy->path = img->path;
        copy->width = img->width;
        copy->height = img->height;
        copy->color_type = img->color_type;
        copy->bit_depth = img->bit_depth;

        png_bytepp   bytes   =   (png_bytepp)malloc(img->height   *
sizeof(png_bytep));
        for (uint y = 0; y < img->height; y++) {
            bytes[y] = (png_bytep)malloc(img->width * PIXSIZE);
            memcpy(bytes[y], img->bytes[y], img->width * PIXSIZE);
        }
        copy->bytes = bytes;
        return copy;
    }

    // Освобождение памяти изображения
    void free_image(image **img)
    {
        if (img == NULL || *img == NULL)
            return;
        for (uint y = 0; y < (*img)->height; y++) {
```

```
            free((*img)->bytes[y]);
        }
        free((*img)->bytes);
        free((*img));
        *img = NULL;
    }

    // Считывание по пикселям (подобие strtok)
    pixel *read_selection(selection *slct)
    {
        static uint iter;
        static selection *slct_img = NULL;
        if (slct != NULL) {
            iter = 0;
            slct_img = slct;
        }

        if (slct_img == NULL)
            return NULL;

        uint x1 = slct_img->start->x;
        uint y1 = slct_img->start->y;
        uint x2 = slct_img->end->x;
        uint y2 = slct_img->end->y;
        uint w = x2 - x1 + 1;
        uint h = y2 - y1 + 1;
        if (iter >= w * h)
            return NULL;

        point  *pos = init_point(x1 + iter % w, y1 + iter / w);
        png_bytep byte = (slct_img->img)->bytes[y1 + iter / w] + (x1
+ iter % w) * 3;
        pixel *pxl = init_pixel(pos, byte);
        iter++;
        return pxl;
    }
```

Название файла: draw.h

```
#include "image.h"

void put_pixel(pixel* pix, color* clr);
pixel* get_pixel(image* img, double x, double y);
color* pixel_to_color(pixel* pix);
```

Название файла: draw.c

draw.c

```
#include "draw.h"


// Устанавливает цвет указанного пикселя
void put_pixel(pixel* pix, color* clr)
{
```

```
    if (pix == NULL || clr == NULL)
        return;

    pix->byte[0] = (png_byte)clr->R;
    pix->byte[1] = (png_byte)clr->G;
    pix->byte[2] = (png_byte)clr->B;
}

// Возвращает указатель на пиксель изображения по указанным координатам
pixel *get_pixel(image *img, double x, double y)
{
    if (img == NULL || img->bytes == NULL)
        return NULL;

    int ix = (int)x;
    int iy = (int)y;
    if ((ix < 0) || (ix >= img->width) || (iy < 0) || (iy >= img->height))
        return NULL;

    point *pos = init_point(ix, iy);
    png_bytep byte = img->bytes[iy] + ix * 3;
    return init_pixel(pos, byte);
}

// Преобразует структуру пикселя в структуру цвета
color *pixel_to_color(pixel *pix)
{
    return init_color(pix->byte[0], pix->byte[1], pix->byte[2]);
}
```

Название файла: tools.h

```
#include "image.h"
#include "draw.h"
#include <math.h>
```

```
#define FRAME_RECTANGLE 1
#define FRAME_CIRCLE 2
#define FRAME_SEMICIRCLES 3

image *split(image* img, uint N, uint M, uint stroke_width, color* stroke);
image *rectangle(selection* slct, uint stroke_width, color* stroke, color* fill);
image* rectangle_for_ornament(selection* slct, uint stroke_width, color* stroke, color* fill);
image *frame(image* img, color* color, uint thickness, uint count, int pattern_type);
void draw_circle(image* img, int centerX, int centerY, int radius, int thickness, color* color);
image* turn(selection* slct, int angle);
image* rotate(image* img, selection* slct, int angle);
```

Название файла: tools.c

```
#include "tools.h"
#define M_PI 3.14159265358979323846

image *rectangle(selection *slct, uint stroke_width, color *stroke, color *fill)
{
    // Заполнение прямоугольника цветом fill
    pixel *pix = read_selection(slct);
    while (pix) {
        if (pix->pos->x > slct->start->x + stroke_width / 2 &&
        pix->pos->x < slct->end->x - stroke_width / 2 &&
        pix->pos->y > slct->start->y + stroke_width/2 &&
        pix->pos->y < slct->end->y - stroke_width / 2) {
            put_pixel(pix, fill);
            free_pixel(&pix);
        } else {
            put_pixel(pix, stroke);
            free_pixel(&pix);
        }
```

```
            pix = read_selection(NULL);
        }

        // Отрисовка границы прямоугольника
        for (uint i = 0; i < (stroke_width + 2) / 2; i++) {
            // Верхняя граница
            for (uint x = slct->start->x - i; x <= slct->end->x + i;
x++) {
                put_pixel(get_pixel(slct->img,  x,  slct->start->y  -
i), stroke);
                put_pixel(get_pixel(slct->img, x, slct->end->y + i),
stroke);
            }

            // Левая и правая границы
            for (uint y = slct->start->y - i; y <= slct->end->y + i;
y++) {
                put_pixel(get_pixel(slct->img,  slct->start->x  -  i,
y), stroke);
                put_pixel(get_pixel(slct->img, slct->end->x + i, y),
stroke);
            }
        }

        return slct->img;
    }

    image*     rectangle_for_ornament(selection*     slct,     uint
stroke_width, color* stroke, color* fill)
    {
        pixel* pix = read_selection(slct);
        while (pix){
            if (
                ((pix->pos->x - slct->start->x) < stroke_width)
              ||((pix->pos->y - slct->start->y) < stroke_width)
              ||((slct->end->x - pix->pos->x) < stroke_width)
              ||((slct->end->y - pix->pos->y) < stroke_width)
            ){
```

```
                put_pixel(pix, stroke);
            } else if (fill) {
                put_pixel(pix, fill);
            }
            free_pixel(&pix);
            pix = read_selection(NULL);
        }
        return slct->img;
    }


    // Разделение изображения на N M частей
    image *split(image *img, uint N, uint M, uint stroke_width,
color *stroke)
    {
        if (N > img->width || M > img->height)
            return NULL;

        uint part_width = img->width / N;
        uint part_height = img->height / M;

        // Вертикальные линии:
        for (int i = 1; i < N; i++) {
            uint x = i * part_width;
            if (x > img->width - stroke_width / 2) {
                x = img->width - stroke_width / 2;
            }

            point *start = init_point(x - stroke_width / 2, 0);
            point *end = init_point(x + stroke_width / 2, img-
>height);
            rectangle_for_ornament(init_selection(img, start, end),
stroke_width, stroke, stroke);
        }

        // Горизонтальные линии:
        for (int i = 1; i < M; i++) {
            uint y = i * part_height;
```

```
            if (y > img->height - stroke_width / 2) {
                y = img->height - stroke_width / 2;
            }
            point *start = init_point(0, y - stroke_width / 2);
            point *end = init_point(img->width, y + stroke_width / 2);
            rectangle_for_ornament(init_selection(img, start, end), stroke_width, stroke, stroke);
        }

        return img;
    }

    // Рисование окружности
    void draw_circle(image* img, int center_x, int center_y, int radius, int thickness, color* color) {
        int outer_radius = radius + thickness / 2; // Внешний радиус
        int inner_radius = radius - thickness / 2; // Внутренний радиус

        for (int x = center_x - outer_radius + 1; x < center_x + outer_radius; ++x) {
            for (int y = center_y - outer_radius + 1; y < center_y + outer_radius; ++y) {
                // Вычисляем квадрат расстояния от текущего пикселя до центра окружности
                int distance_squared = (x - center_x) * (x - center_x) + (y - center_y) * (y - center_y);

                // Проверяем, находится ли текущий пиксель в пределах толщины окружности
                if (distance_squared >= inner_radius * inner_radius && distance_squared <= outer_radius * outer_radius) {
                    pixel* pix = get_pixel(img, x, y);
                    if (pix) put_pixel(pix, color);
                }
            }
        }
```

```
    }

    // Рамка для изображения
    image *frame(image* img, color* color, uint thickness, uint
count, int pattern_type) {
        uint width = img->width;
        uint height = img->height;

        switch (pattern_type) {
            case FRAME_RECTANGLE: {
                // Начальные координаты для первого прямоугольника
                uint start_x = 0;
                uint start_y = 0;
                uint rect_width = width;
                uint rect_height = height;

                // Рисуем прямоугольники
                for (uint i = 0; i < count; i++) {
                    //   Инициализируем   выделенную   область   для
текущего прямоугольника
                    point *start = init_point(start_x, start_y);
                    point  *end  =  init_point(start_x  +  rect_width  -
1, start_y + rect_height - 1);
                    selection  *slct  =  init_selection(img,  start,
end);

                    // Рисуем прямоугольник
                    rectangle_for_ornament(slct,  thickness,  color,
NULL);

                    // Обновляем координаты и размеры для следующего
прямоугольника
                    start_x += thickness * 2;
                    start_y += thickness * 2;
                    rect_width -= 4 * thickness;
                    rect_height -= 4 * thickness;

                    free_selection(&slct);
```

```
                }
                break;
            }

            case FRAME_CIRCLE: {
                uint radius = MIN(width, height) / 2;
                uint center_x = width / 2;
                uint center_y = height / 2;

                for (uint y = 0; y < height; y++) {
                    for (uint x = 0; x < width; x++) {
                        if ((x - center_x) * (x - center_x) + (y -
center_y) * (y - center_y) > radius * radius) {
                            pixel* p = get_pixel(img, x, y);
                            put_pixel(p, color);
                        }
                    }
                }
                break;
            }

            case FRAME_SEMICIRCLES: {
                int radius_x = ceil(((double)(width / count) /
2.0));
                int radius_y = ceil((double)((height / count) /
2.0));

                // Верхняя граница
                for (int i = radius_x; i < width + thickness; i += 2
* radius_x) {
                    draw_circle(img, i, 0, radius_x, thickness + 1,
color);
                }

                // Нижняя граница
                for (int i = radius_x; i < width + thickness; i += 2
* radius_x) {
```

```
                    draw_circle(img, i, height, radius_x, thickness
+ 1, color);
                }

                // Левая граница
                for (int i = radius_y; i < height + thickness; i +=
2 * radius_y) {
                    draw_circle(img, 0, i, radius_y, thickness + 1,
color);
                }

                // Правая граница
                for (int i = radius_y; i < height + thickness; i +=
2 * radius_y) {
                    draw_circle(img, width, i, radius_y, thickness +
1, color);
                }
                break;
            }

            default:
                // Неверный тип узора
                break;
        }
        return img;
    }

    // Поворот выделенной области на 90, 180 и 270 градусов
    image* turn(selection* slct, int angle) {
        uint width = slct->end->x - slct->start->x;
        uint height = slct->end->y - slct->start->y;

        // Создание нового изображения с измененными размерами
        image* new_img = malloc(sizeof(image));
        if (new_img == NULL) {
            return NULL; // Ошибка выделения памяти
        }
        new_img->width = (angle == 180) ? width : height;
```

```
        new_img->height = (angle == 180) ? height : width;
        new_img->color_type = slct->img->color_type;
        new_img->bit_depth = slct->img->bit_depth;
        new_img->bytes       =       malloc(new_img->height       *
sizeof(png_bytep));
        if (new_img->bytes == NULL) {
            free(new_img);
            return NULL; // Ошибка выделения памяти
        }

        for (uint i = 0; i < new_img->height; i++) {
            new_img->bytes[i]     =     malloc(new_img->width     *
sizeof(png_byte) * 3);
            if (new_img->bytes[i] == NULL) {
                // Освобождение ранее выделенной памяти
                for (uint j = 0; j < i; j++) {
                    free(new_img->bytes[j]);
                }
                free(new_img->bytes);
                free(new_img);
                return NULL; // Ошибка выделения памяти
            }
        }

        // Поворот изображения
        for (uint y = slct->start->y; y < slct->end->y; y++) {
            for (uint x = slct->start->x; x < slct->end->x; x++) {
                uint src_x = x - slct->start->x;
                uint src_y = y - slct->start->y;
                uint dest_x, dest_y;

                // Вычисление новых координат пикселя после поворота
                switch (angle) {
                    case 90:
                        dest_x = src_y;
                        dest_y = width - src_x - 1;
                        break;
```

```
                    case 180:
                        dest_x = width - src_x - 1;
                        dest_y = height - src_y - 1;
                        break;

                    case 270:
                        dest_x = height - src_y - 1;
                        dest_y = src_x;
                        break;

                    default:
                        // Некорректный угол поворота
                        for (uint i = 0; i < new_img->height; i++) {
                            free(new_img->bytes[i]);
                        }
                        free(new_img->bytes);
                        free(new_img);
                        return NULL;
                }

                // Копирование пикселя в новое место
                memcpy(&new_img->bytes[dest_y][dest_x * 3], &slct-
>img->bytes[y][x * 3], 3);
            }
        }

        return new_img;
    }

    // Вставка повёрнутого изображения
    image* rotate(image* img, selection* slct, int angle) {
        // Поворот выбранной области
        image* rotated_region = turn(slct, angle);
        if (rotated_region == NULL) {
            return NULL;
        }

        // Создание временной копии изображения
```

```
image* temp_img = copy_image(img);
if (temp_img == NULL) {
    free_image(&rotated_region);
    return NULL;
}

uint start_x = slct->start->x;
uint start_y = slct->start->y;
uint end_x = slct->end->x;
uint end_y = slct->end->y;

uint p1_x = slct->start->x;
uint p1_y = slct->start->y;
uint p2_x = slct->end->x;
uint p2_y = p1_y;
uint p3_x = slct->end->x;
uint p3_y = slct->end->y;
uint p4_x = p1_x;
uint p4_y = p3_y;


// Определение границ для вставки повернутой области
uint new_start_x, new_start_y, new_end_x, new_end_y;
double center_x = ((start_x + end_x) / 2.0);
double center_y = ((start_y + end_y) / 2.0);

int dx = 0;
int dy = 0;
int dxx = 0;
int dyy = 0;
double intPartX, fracPartX;
fracPartX = modf((start_x + end_x) / 2.0, &intPartX);
double intPartY, fracPartY;
fracPartY = modf((start_y + end_y) / 2.0, &intPartY);

// Смещение
if (fracPartX == 0.0 && fracPartY == 0.0) {
    // Оба числа целые
```

```
            dx = 0;
            dy = 0;
        } else if (fracPartX == 0.0) {
            // Только X целый
            if (end_x - start_x < end_y - start_y) {
                dx = 0;
                dy = -1;
                dxx = 1;
                dyy = 1;
            } else if (end_x - start_x >= end_y - start_y) {
                dx = 0;
                dy = -1;
                dxx = 1;
                dyy = 1;
            }
        } else if (fracPartY == 0.0) {
            // Только Y целый
            if (img->width <= img->height) {
                dx = 0;
                dy = 1;
            } else if (img->width > img->height) {
                dx = -1;
                dy = 0;
                dxx = 1;
                dyy = 1;
            }
        } else {
            // Оба числа нецелые
            dy = 0;
            dx = 0;
        }

        switch (angle) {
            case 90:
                new_start_x = (uint)ceil((double)(-1 * (p4_y -
center_y) + center_x)) + dx;
                new_start_y = (uint)ceil((double)((p4_x - center_x)
+ center_y)) + dy;
```

```
                new_end_x = ((-1 * (p2_y - center_y) + center_x) >
img->width) ? img->width : (-1 * (p2_y - center_y) + center_x) + dx +
dxx;
                new_end_y = (((p2_x - center_x) + center_y) > img-
>height) ? img->height : ((p2_x - center_x) + center_y) + dy + dyy;
                break;

            case 180:
                new_start_x = start_x;
                new_start_y = start_y;
                new_end_x = end_x;
                new_end_y = end_y;
                break;

            case 270:
                new_start_x  =  (uint)ceil((double)(-1  *  (p4_y  -
center_y) + center_x)) + dx;
                new_start_y = (uint)ceil((double)((p4_x - center_x)
+ center_y)) + dy;
                new_end_x = ((-1 * (p2_y - center_y) + center_x) >
img->width) ? img->width: (-1 * (p2_y - center_y) + center_x) + dx +
dxx;
                new_end_y = (((p2_x - center_x) + center_y) > img-
>height) ? img->height: ((p2_x - center_x) + center_y) + dy + dyy;
                break;

            default:
                free_image(&rotated_region);
                free_image(&temp_img);
                return NULL;
        }

        // Вставка повернутой области в исходное изображение
        for (uint y = ((int)new_start_y < 0) ? 0 : new_start_y; y <
new_end_y; y++) {
            for (uint x = ((int)new_start_x < 0) ? 0 : new_start_x;
x < new_end_x; x++) {
                uint src_x = x - new_start_x;
```

```
                uint src_y = y - new_start_y;
                uint dest_x = x;
                uint dest_y = y;
                memcpy(&temp_img->bytes[dest_y][dest_x * 3],
&rotated_region->bytes[src_y][src_x * 3], 3);
            }
        }


        free_image(&rotated_region);
        free_image(&img);

        return temp_img;
    }
```

Название файла: main.c

```
#include <getopt.h>
#include <ctype.h>
#include "image.h"
#include "draw.h"
#include "tools.h"

#define ERR_UNKNOWN_OPRION 300
#define ERR_TOOL_NOT_SELECTED 301
#define ERR_TO_MANY_TOOLS 302

#define ERR_LONG_FILENAME 303
#define ERR_TO_MANY_DOTS 304
#define ERR_INVALID_DOT 305
#define ERR_INVALID_COLOR 306
#define ERR_INVALID_INT 307
#define ERR_INVALID_RANGE 308
#define ERR_TO_MANY_ANGLES 309
#define ERR_INVALID_PATTERN 310
#define ERR_TO_MANY_FILES 311
#define ERR_NO_FILES 312
```

```
#define ERR_SPLIT 313
#define ERR_RECTANGLE 314
#define ERR_FRAME 315
#define ERR_TURN 316

#define ERR_WRONG_ARGV 317
#define ERR_FLAG_FILL 318


#define MAX_FILE_NAME 255
#define MAX_DOTS_COUNT 2


void raise_error(int error, int val)
{
     char error_text[100][1000] = {
        "The option is unknown or a required argument is not specified",
        "Use first one of this tools --split (-s), --rect (-r), --ornament (-o), --rotate (-r)",
        "Select one of the tools (two or more are specified)",
        "Filename is too long",
        "Too many dots (points) were passed as options",
        "Invalid dot (point). Write 2 unsigned integers separated by commas (without spaces)",
        "Invalid color. Write color in RGB: 3 unsigned integers [0;255] separated by commas (without spaces).",
        "Failed to convert string to number",
        "The number should be in the range from 1 to the width or height of the image",
        "Too many angles were passed as options",
        "Invalid pattern number",
        "Too many files",
        "Need input file",
        "Invalid arguments to split",
        "Invalid arguments to rectangle",
        "Invalid arguments to frame",
        "Invalid arguments to rotate",
```

```
            "Two input files are specified or an extra unknown argv
argument is specified",
            "Missing --fill flag"
        };

        if (error) {
            if (300 <= error <= 399) {
                if (val > 0) {
                    fprintf(stderr, "-%c: %s. Use -h or --help\n",
val, error_text[error-300]);
                } else {
                    fprintf(stderr,  "%s.  Use  -h  or  --help\n",
error_text[error-300]);
                }
            }
            exit(45);
        }
    }

    // Является ли строка числом
    int isNumeric(char *str)
    {
        if (str == NULL)
            return 0;
        int i = 0;
        while (str[i] != '\0' && (isdigit(str[i]) || (str[0] == '-
'))) {
            if (!(isdigit(str[i]) || (str[0] == '-'))) {
                return 0;
            }
            i++;
        }
        return 1;
    }

    // Преобразование строки в структуру point
    point *str_to_point(char *str)
    {
```

```
    if (str == NULL)
        return NULL;
    int count, x, y;
    count = sscanf(str, "%d.%d", &x, &y);
    if (count != 2)
        return NULL;
    if (x < 0 || y < 0)
        return NULL;
    return init_point(x, y);
}

// Преобразование строки в структуру color
color *str_to_color(char *str)
{
    if (str == NULL)
        return NULL;
    int count, r, g, b;
    count = sscanf(str, "%d.%d.%d", &r, &g, &b);
    if (count != 3)
        return NULL;
    return init_color(r, g, b);
}

int main(int argc, char *argv[])
{
    const char* short_options = "hsofti:n:m:c:p:a:";
    const struct option long_options[] = {
        { "help", no_argument, NULL, 'h' },
        { "split", no_argument, NULL, 'S' },
        { "rect", no_argument, NULL, 'R' },
        { "ornament", no_argument, NULL, 'F' },
        { "rotate", no_argument, NULL, 'T' },
        { "input", required_argument, NULL, 'i'},
        { "output", required_argument, NULL, 'o'},
        { "number_x", required_argument, NULL, 'n'},
        { "number_y", required_argument, NULL, 'm'},
        { "color", required_argument, NULL, 'c'},
        { "fill", no_argument, NULL, 'f'},
```

```
        { "fill_color", required_argument, NULL, 'G'},
        { "pattern", required_argument, NULL, 'p'},
        { "angle", required_argument, NULL, 'a'},
        { "count", required_argument, NULL, 'N'},
        { "thickness", required_argument, NULL, 'W'},
        { "left_up", required_argument, NULL, 'L'},
        { "right_down", required_argument, NULL, 'D'},
        { "info", required_argument, NULL, 'H'},
        { NULL, 0, NULL, 0 }
    };

    int val;
     int option_index = -1; // индекс текущей опции
    int options_count = 0; // кол-во обработанных опций

    int count = -1;
    int thickness = -1; // толщина линии
    int tool = 0; // 1 - split, 2 - rectangle, 3 - frame, 4 -
turn
    int input_flag = 0; // был ли указан входной файл
    char input_file[MAX_FILE_NAME + 1]; // путь к входному файлу
    int output_flag = 0; // был ли указан выходной файл
    char output_file[MAX_FILE_NAME + 1]; // путь к выходному
файлу
    point *new_dot = NULL; // указатель на точку
    point *dots[MAX_DOTS_COUNT]; // массив указателей на точки
    int dots_count = 0; // кол-во точек
    color *clr = NULL; // цвет
    int number;
    int n = -1; // число разделений по горизонтали
    int m = -1; // число разделений по вертикали
    // int width = -1; // ширина
    int flag_angle = 0; // был ли указан угол поворота
    int angle = -1; // угол поворота
    int pattern = 0; // узор для рамки
    char *pattern_str = NULL;
    int fill = 0;
    int flag_fill_color = 0;
```

```
        color *fill_color = NULL; // заполнение цветом
        point *left;
        point *right;

        int error = 0;
        while ((val = getopt_long(argc, argv, short_options,
long_options, &option_index)) != -1) {
            switch (val) {
                // Справка (help)
                case 'h':
                    puts(
                        "Course work for option 5.11, created by
Aleksandr Volkov\n\n"
                        "1) Split. Divides images into NxM parts.\n"
                        "Accepts the input numbers N and M, the
color and thickness of the stroke.\n"
                        "Key: --split or -s\n"
                        "Parameter Keys:\n"
                        "\t--number_x or -n: unsigned integer, range
[1; width of image]\n"
                        "\t--number_y or -m: unsigned integer, range
[1; height of image]\n"
                        "\t--thickness: unsigned integer, >= 1\n"
                        "\t--color or -c: RGB, 3 unsigned integers,
range [0; 255]\n\n"

                        "2) Rectangle. Draws a rectangle with an
outline and fill.\n"
                        "Accepts two dots, fill color (optinal), the
color and thickness of the stroke.\n"
                        "Key: --rect or -r\n"
                        "Parameter Keys:\n"
                        "\t--left_up: 2 unsigned integers through
the point, >= 0\n"
                        "\t--right_down: 2 unsigned integers through
the point, >= 0\n"
                        "\t--fill: true or false\n"
```

```
                        "\t--fill_color:  RGB, 3 unsigned integers,
range [0; 255]\n"
                        "\t--color or -c: RGB, 3 unsigned integers,
range [0; 255]\n\n"

                        "3) Ornament. Draws a frame around the
image.\n"
                        "Accepts pattern number, the color and
thickness of the stroke.\n"
                        "Key: --ornament or -o\n"
                        "Parameter Keys:\n"
                        "\t--pattern or -p: rectangle or circle or
semicircles\n"
                        "\t--thickness: unsigned integer, >= 1\n"
                        "\t--count: unsigned integer, >= 1\n"
                        "\t--color or -c: RGB, 3 unsigned integers,
range [0; 255]\n\n"

                        "4) Rotate. Rotates part of the image by the
specified number of degrees\n"
                        "Accepts two dots and angle (in degrees).\n"
                        "Key: --rotate or -r\n"
                        "Parameter Keys:\n"
                        "\t--left_up:  2  unsigned  integers  through
the point, >= 0\n"
                        "\t--right_down: 2 unsigned integers through
the point, >= 0\n"
                        "\t--angle or -a: integer\n\n"

                        "Also,  all  functions  require  an  input
file,\n"
                        "which can be specified via the --input or -
i key.\n"
                        "The  output  file  is  defined  by  a  similar
flag --output or -o.\n"
                    );

                    exit(0);
```

```
        break;

    case 'H': // --info
        if (strlen(optarg) >= MAX_FILE_NAME) {
            error = ERR_LONG_FILENAME;
            break;
        }
        strncpy(input_file, optarg, MAX_FILE_NAME);

        image *img = load_image(input_file);
        raise_error(!img, -1);

        printf("Image Information:\n");
        printf("Path: %s\n", img->path);
        printf("Width: %u\n", img->width);
        printf("Height: %u\n", img->height);
        printf("Color Type: %u\n", img->color_type);
        printf("Bit Depth: %u\n", img->bit_depth);
        input_flag = 1;
        exit(0);
        break;

    // Выбор инструмента
    case 'S': // --split or -s
        if (tool) {
            error = ERR_TO_MANY_TOOLS;
            break;
        }
        tool = 1;
        break;

    case 'R': // --rect or -r
        if (tool) {
            error = ERR_TO_MANY_TOOLS;
            break;
        }
        tool = 2;
        break;
```

```
case 'F': // --ornament or -o
    if (tool) {
        error = ERR_TO_MANY_TOOLS;
        break;
    }
    tool = 3;
    break;

case 'T': // --rotate or -t
    if (tool) {
        error = ERR_TO_MANY_TOOLS;
        break;
    }
    tool = 4;
    break;

// Ввод параметров инструментов
case 'i': // --input or -i
    if (input_flag) {
        error = ERR_TO_MANY_FILES;
        break;
    }

    if (strlen(optarg) >= MAX_FILE_NAME) {
        error = ERR_LONG_FILENAME;
        break;
    }
    input_flag = 1;
    strncpy(input_file, optarg, MAX_FILE_NAME);
    break;

case 'o': // --output or -o
    if (output_flag) {
        error = ERR_TO_MANY_FILES;
        break;
    }
```

```
        if (strlen(optarg) >= MAX_FILE_NAME) {
            error = ERR_LONG_FILENAME;
            break;
        }
        output_flag = 1;
        strncpy(output_file, optarg, MAX_FILE_NAME);
        break;

    case 'c': // --color or -c
        clr = str_to_color(optarg);
        if (!clr) {
            error = ERR_INVALID_COLOR;
            break;
        }
        break;


    case 'f': // --fill
        fill = 1;
        break;

    case 'G': // --fill_color
        fill_color = str_to_color(optarg);
        if (!fill_color) {
            error = ERR_INVALID_COLOR;
            break;
        }
        flag_fill_color = 1;
        break;

    case 'n': // --number_x
        if (!isNumeric(optarg)) {
            error = ERR_INVALID_INT;
            break;
        }

        number = atoi(optarg);
        if (1 > number) {
```

```
            error = ERR_INVALID_RANGE;
            break;
        }
        n = number;
        break;

    case 'm': // --number_y
        if (!isNumeric(optarg)) {
            error = ERR_INVALID_INT;
            break;
        }

        number = atoi(optarg);
        if (1 > number) {
            error = ERR_INVALID_RANGE;
            break;
        }
        m = number;
        break;

    case 'a': // --angle or -a
        if (!isNumeric(optarg)) {
            error = ERR_INVALID_INT;
            break;
        }

        if (flag_angle) {
            error = ERR_TO_MANY_ANGLES;
            break;
        }
        number = atoi(optarg);
        angle = number;
        flag_angle = 1;
        break;

    case 'p': // --pattern or -p
        pattern_str = optarg;
```

```
                    if (strcmp(pattern_str, "rectangle") != 0 &&
strcmp(pattern_str,   "circle")   !=   0   &&   strcmp(pattern_str,
"semicircles") != 0) {
                        error = ERR_INVALID_PATTERN;
                        break;
                    }

                    if (strcmp(pattern_str, "rectangle") == 0) {
                        pattern = 1;
                    } else if (strcmp(pattern_str, "circle") == 0) {
                        pattern = 2;
                    } else if (strcmp(pattern_str, "semicircles") ==
0) {
                        pattern = 3;
                    }

                    break;

                case 'N': // --count
                    if (!isNumeric(optarg)) {
                        error = ERR_INVALID_INT;
                        break;
                    }

                    number = atoi(optarg);
                    if (1 > number) {
                        error = ERR_INVALID_RANGE;
                        break;
                    }
                    count = number;
                    break;

                case 'W': // --thickness
                    if (!isNumeric(optarg)) {
                        error = ERR_INVALID_INT;
                        break;
                    }
```

```
            number = atoi(optarg);
            if (1 > number) {
                error = ERR_INVALID_RANGE;
                break;
            }
            thickness = number;
            break;

        case 'L': // --left_up
            if (dots_count >= MAX_DOTS_COUNT) {
                error = ERR_TO_MANY_DOTS;
                break;
            }

            new_dot = str_to_point(optarg);
            if (!new_dot) {
                error = ERR_INVALID_DOT;
                break;
            }
            dots[dots_count++] = new_dot;
            left = new_dot;
            break;

        case 'D': // --right_down
            if (dots_count >= MAX_DOTS_COUNT) {
                error = ERR_TO_MANY_DOTS;
                break;
            }

            new_dot = str_to_point(optarg);
            if (!new_dot) {
                error = ERR_INVALID_DOT;
                break;
            }
            dots[dots_count++] = new_dot;
            right = new_dot;
            break;
```

```
                // Неизвестный флаг
                case '?':
                default: {
                    error = ERR_UNKNOWN_OPRION;
                    break;
                }
            }

            if (error)
                break;
            option_index = -1;
            options_count++;
        }

        options_count -= output_flag + input_flag + (tool > 0);

        raise_error(error, val);
        if (tool == 0)
            error = ERR_TOOL_NOT_SELECTED;

        if (optind < argc){
            if (!input_flag) {
                if (strlen(argv[argc - 1]) > MAX_FILE_NAME) {
                    error = ERR_LONG_FILENAME;
                } else {
                    input_flag = 1;
                    strncpy(input_file,     argv[argc     -     1],
MAX_FILE_NAME);
                }
            } else {
                error = ERR_WRONG_ARGV;
            }
        }
        if (!input_flag) {
            error = ERR_NO_FILES;
        }
        raise_error(error, -1);
```

```
        image *img = load_image(input_file);
        raise_error(!img, -1);

        image *out = NULL;
        switch (tool) {
            case 1:
                //SPLIT
                if ((n == -1) || (m == -1) || (clr == NULL) ||
(thickness < 0) || (options_count != 4)) {
                    error = ERR_SPLIT;
                    break;
                }

                out = split(img, n, m, thickness, clr);
                if (!out) {
                    error = ERR_SPLIT;
                    break;
                }
                break;

            case 2:
                //RECTANGLE
                if ((dots_count != 2) || (clr == NULL) || (thickness
< 0) || (options_count < 3) || (options_count > 6)) {
                    error = ERR_RECTANGLE;
                    break;
                }

                if (left->y > img->height && right->y > img->height)
{
                    out = img;
                    break;
                }

                if (left->x > img->width && right->x > img->width) {
                    out = img;
                    break;
                }
```

```
                if (left->x > right->x) {
                    double tmp = left->x;
                    left->x = right->x;
                    right->x = tmp;
                }

                if (left->y > right->y) {
                    double tmp = left->y;
                    left->y = right->y;
                    right->y = tmp;
                }

                if (fill == 1 && flag_fill_color == 1) {
                    out    =    rectangle(init_selection(img,    left,
right), thickness, clr, fill_color);
                } else if (fill == 0 && flag_fill_color == 0) {
                    out    =    rectangle(init_selection(img,    left,
right), thickness, clr, NULL);
                } else if (fill == 1 && flag_fill_color == 0) {
                    error = ERR_RECTANGLE;
                } else if (fill == 0 && flag_fill_color == 1) {
                    out    =    rectangle(init_selection(img,    left,
right), thickness, clr, NULL);
                }

                if (!out) {
                    error = ERR_RECTANGLE;
                    break;
                }
                break;

            case 3:
                //FRAME
                if ((pattern < 0) || (clr == NULL) || (count < 0) ||
(thickness < 0) || (options_count > 4)) {
                    error = ERR_FRAME;
                    break;
```

```
                }

                out = frame(img, clr, thickness, count, pattern);

                if (!out) {
                    error = ERR_FRAME;
                    break;
                }
                break;

            case 4:
                //ROTATE
                if ((dots_count != 2) || (flag_angle == 0) ||
(options_count != 3)) {
                    error = ERR_TURN;
                    break;
                }

                out = rotate(img, init_selection(img, left, right),
angle);
                if (!out) {
                    error = ERR_TURN;
                    break;
                }
                break;

            default:
                error = ERR_TOOL_NOT_SELECTED;
                break;
        }

        raise_error(error, -1);
        if (!output_flag) {
            error = save_image(out, input_file);
        } else {
            error = save_image(out, output_file);
        }
        raise_error(error, -1);
```

```
    return 0;
}
```