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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Обработка изображений

Студентка гр. 3341 _____ Кузнецова С.Е.

Преподаватель _____ Глазунов С.А.

Санкт-Петербург

2024

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студентка Кузнецова С.Е.

Группа 3341

Тема работы: Обработка изображений

Вариант 4.13

Программа обязательно должна иметь CLI (опционально дополнительное использование GUI). Более подробно тут: http://se.moevm.info/doku.php/courses:programming:rules_extra_kurs

Программа должна реализовывать весь следующий функционал по обработке png-файла

Общие сведения

Формат картинки PNG (рекомендуем использовать библиотеку libpng) без сжатия

файл может не соответствовать формату PNG, т.е. необходимо проверка на PNG формат. Если файл не соответствует формату PNG, то программа должна завершиться с соответствующей ошибкой.

обратите внимание на выравнивание; мусорные данные, если их необходимо дописать в файл для выравнивания, должны быть нулями.

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

Программа должна иметь следующую функции по обработке изображений:

(1) Рисование окружности. Флаг для выполнения данной операции: `--circle`. Окружность определяется:

координатами ее центра и радиусом. Флаги `--center` и `--radius`. Значение флаг `--center` задаётся в формате `x.y`, где x – координата по оси x, y – координата по оси y. Флаг `--radius` На вход принимает число больше 0

толщиной линии окружности. Флаг `--thickness`. На вход принимает число больше 0

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

окружность может быть залитой или нет. Флаг `--fill`. Работает как бинарное значение: флага нет – false, флаг есть – true.

цветом которым залита сама окружность, если пользователем выбрана залитая окружность. Флаг `--fill_color` (работает аналогично флагу `--color`)

(2) Отражение заданной области. Флаг для выполнения данной операции: `--mirror`. Этот функционал определяется:

выбором оси относительно которой отражать (горизонтальная или вертикальная). Флаг `--axis`, возможные значения `x` и `y`

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

(3) Копирование заданной области. Флаг для выполнения данной операции: `--copy`. Функционал определяется:

Координатами левого верхнего угла области-источника. Флаг `--left_up`, значение задаётся в формате `left.up`, где left – координата по x, up – координата по y

Координатами правого нижнего угла области-источника. Флаг `--right_down`, значение задаётся в формате `right.down`, где right – координата по x, down – координата по y

Координатами левого верхнего угла области-назначения. Флаг `--dest_left_up`, значение задаётся в формате `left.up`, где left – координата по x, up – координата по y

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

Содержание пояснительной записки:
Аннотация, содержание, введение, ход выполнения, заключение, приложения.

Предполагаемый объем пояснительной записки:
Не менее 15 страниц.
Дата выдачи задания: 18.03.2024

Дата сдачи реферата: 21.05.2024

Дата защиты реферата: 23.05.2024

Студентка _____ Кузнецова С.Е.

Преподаватель _____ Глазунов С.А.

АННОТАЦИЯ

В данной курсовой работе была реализована программа, обрабатывающая PNG изображения, не имеющие сжатия. Программа проверяет тип изображения, его версию, при соответствии требованиям в дальнейшем обрабатывает его и подаёт на выход изменённую копию изображения. Взаимодействие с программой осуществляется с помощью CLI (интерфейс командной строки).

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Формат файлов PNG широко используется на веб-сайтах для отображения высококачественных цифровых изображений. Созданный для того, чтобы превзойти по производительности файлы GIF, PNG обеспечивает не только сжатие без потерь, но и более широкую и яркую цветовую палитру.

PNG – это сокращение от Portable Network Graphic, тип файла растрового изображения. Этот тип файлов особенно популярен среди вебдизайнеров, поскольку он позволяет работать с графикой с прозрачным или полупрозрачным фоном.

Целью курсовой работы является изучение формата файлов PNG, а также реализация функций для работы с этим форматом файлов.

Для достижения поставленной цели требуется решить следующие задачи:

1) изучить PNG формат изображений;

2) получить информацию об изображении: размеры, содержимое и др.;

3) обработать массив пикселей в соответствии с заданием;

4) обработать исключительные случаи;

5) сохранить итоговое изображение в новый файл.

1. РАБОТА С ФАЙЛАМИ

Работа с файлом происходит при помощи библиотеки png.h. Функции по считыванию файла, проверки его и заполнения соответствующей структуры, а также функция по созданию PNG файла и записыванию в него полученную структуру описаны в мануале libpng.

В работе используется сигнатура Png, которая содержит в себе информацию об изображении, основные поля:

– ширина и высота изображения;

– тип цвета, который используется в изображении (например, используется ли палитра, монохромное изображение, присутствует ли альфа канал);

– глубина цвета, т. е. сколько компонент (в данном случае байтов) описывают цвет (для RGBA – это 4, а для RGB – 3);

– количество проходов, которое необходимо, чтобы полностью обработать изображение. Данный параметр необходим для быстроты визуализации изображения.

Функция read_png_file отвечает за чтение данных из файла. Она читает файл стандартной функцией fopen и проверяет по возможной сигнатуре, является ли это изображение Png, заполняет структуру Png, в случае ошибки осуществляется выход из программы.

Функция write_png_file записывает данные в файл. Она открывает файл в для записи, записывает заголовки и пиксели изображения с учетом выравнивания данных, очищает память, выделенную под изображение и закрывает файл, в случае ошибки на каком-либо шаге осуществляется выход из программы.

2. ВВОД АРГУМЕНТОВ

Для разбора аргументов командной строки используется специальная функция getopt() из заголовочного файла getopt.h. Чтение аргументов строки используется в цикле, а также с конструкцией switch-case, которая делает различные действия в зависимости от переданного аргумента (определяет параметры или вызывает функции).

Для перемещения по массиву аргументов, программа использует глобальную переменную optind. По умолчанию она установлена в 1, тем самым указывая на индекс первого аргумента командной строки, идущего после аргумента с названием запущенной программы.

Перед каждым вызовом такой функции осуществляется проверка аргументов. Если какой-либо из аргументов принимает недопустимое значение, работа программы прекращается.

3. ОБРАБОТКА ИЗОБРАЖЕНИЙ

3.1 Рисование окружности и ее заливка

Осуществляется при помощи функции draw_circle(). На вход функции подаётся центр окружности, радиус, толщина, флаг, определяющий наличие заливки, цвет заливки и цвет линии окружности. Для изменения изображения осуществляется перебор всех пикселей картинки, для каждого пикселя проверяется, лежит ли он на окружности или внутри нее (функции check_in_circle() и check_on_circle_line()), и задает ему либо цвет заливки, если он лежит внутри и флаг заливки не 0, либо цвет линии окружности с помощью функции set_color(), если пиксель на линии окружности.

3.2 Отражение заданной области

Осуществляется с помощью функций mirror_y() и mirror_x() в зависимости от значения оси, относительно которой отражается часть изображения. Функции принимают значения левой верхней и правой нижней координаты и на их основе меняют значения цветов противоположных пикселей относительно оси с помощью функции swap().

3.3 Копирование заданной области

Осуществляется с помощью функции copy_pic(). На вход подаются значения левой верхней, правой нижней координаты и левой верхней координаты области, в которую надо осуществить вставку скопированной части. В функции создается двумерный массив, который в цикле заполняется цветами пикселей скопированной части, а затем этими цветами заполняется часть, в которую надо вставить скопированную область.

ЗАКЛЮЧЕНИЕ

Разработана программа на языке программирования Си, обрабатывающая PNG изображения и имеющая CLI. В ходе выполнения работы было изучено устройство PNG файлов.

- Изучены методы считывание и записи файлов.
- Получены навыки обработки изображений.
- Разработаны функции для рисования окружности и ее заливки; отражения части изображения; копирования заданной области.
- Изучены библиотеки libpng и getopt.h
- Изучена работа с аргументами командной строки.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. getopt(3) – Linux manual page. URL: https://man7.org/linux/man-pages/man3/getopt.3.html

2. Документация libpng. URL: http://www.libpng.org/pub/png/pngdocs.html

3. М. М. Заславский, А. А. Лисс, А. В. Гаврилов, С. А. Глазунов, Я. С. Государкин, С. А. Тиняков, В. П. Голубева, К. В. Чайка, В. Е. Допира. – Базовые сведения к выполнению курсовой работы по дисциплине «Программирование». Второй семестр, 2024.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: cw.c

```
#include <stdio.h>
#include <getopt.h>

#include "image_processing.h"
#include "args_parsing.h"
#include "png_io.h"

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

    printf("Course work for option 4.13, created by Svetlana Kuznetsova\n");

    opterr = 0;
    Png image;
    char* path_i = (char *)calloc(128, sizeof(char));
    char* path_o = (char *)calloc(128, sizeof(char));
    int flag;
    OPTS options = init_options();
    path_o = strdup("out.png");

    args_parsing(argc, argv, path_i, path_o, &options, &flag);
    printf("%s\n", path_i);

    read_png_file(path_i, &image);
    choose_func(image, flag, &options);
    write_png_file(path_o, &image);

    return 0;
}
```

Название файла: structs.h

```
#pragma once
#include <png.h>
#include <stdbool.h>

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

    png_structp png_ptr;
    png_infop info_ptr;
    int number_of_passes;
    png_bytep *row_pointers;
} Png;

typedef struct {
    int r;
    int g;
    int b;
} RGB;

typedef struct {
    int x;
    int y;
} COORD;
```

```
typedef struct{
    int flag;
    int circle;
    int copy;
    int mirror;
    int contr;

    int radius;
    int thickness;
    float alpha;
    int beta;

    bool fill_flag;
    bool fill;
    bool input_flag;
    bool info_flag;

    char* axis;

    COORD center_coords;
    COORD left_up_coords;
    COORD right_down_coords;
    COORD dest_left_up_coords;

    RGB color;
    RGB fill_color;
} OPTS;
```

Название файла: args_parsing.c

```
#include "args_parsing.h"

OPTS init_options(){
    OPTS new_opts;
    new_opts.flag = 0;
    new_opts.circle = 0;
    new_opts.copy = 0;
    new_opts.mirror = 0;
    new_opts.contr = 0;

    new_opts.radius = 0;
    new_opts.thickness = 0;
    new_opts.alpha = 0;
    new_opts.beta = 0;

    new_opts.fill_flag = false;
    new_opts.fill = false;
    new_opts.input_flag = false;
    new_opts.info_flag = false;

    return new_opts;
}

bool is_correct_component(int val) {
  if (val >= 0 && val <= 255)
    return true;

  return false;
}

int count_dots(char* arg){
  int i, j = 0;
  for (i = 0; i < strlen(arg); i++){
```

```
    if (arg[i] == '.'){
      j++;
    }
  }
  return j;
}

void are_coordinates(char *arg, COORD* coords) {
  char check[10];

  if (count_dots(arg) != 1){
    exit(Error_args);
  }

  char *tmp = strtok(arg, ".");
  coords->x= atoi(tmp);

  sprintf(check,"%d",coords->x);
  if (strcmp(tmp, check))
    exit(Error_args);

  tmp = strtok(NULL, ".");

  if (tmp == NULL)
    exit(Error_args);

  coords->y= atoi(tmp);

  sprintf(check,"%d",coords->y);
  if (strcmp(tmp, check))
    exit(Error_args);

  tmp = strtok(NULL, ".");
  if (tmp!=NULL)
    exit(Error_args);
}

int to_number(char* arg){
  int number;
  char check[10];
  number = atoi(arg);

  sprintf(check,"%d",number);
  if (strcmp(arg, check))
    exit(Error_args);

  return number;
}

void to_color(char* arg, RGB* color){
  char check[10];

  if (count_dots(arg) != 2){
    exit(Error_args);
  }

  char *tmp = strtok(arg, ".");
  color->r = atoi(tmp);

  sprintf(check,"%d",color->r);
  if (strcmp(tmp, check) || (is_correct_component(color->r)==0))
    exit(Error_args);
```

```
  tmp = strtok(NULL, ".");
  color->g = atoi(tmp);

  sprintf(check,"%d",color->g);
  if (strcmp(tmp, check) || (is_correct_component(color->g)==0))
    exit(Error_args);

  tmp = strtok(NULL, ".");

  if (tmp == NULL)
    exit(Error_args);

  color->b = atoi(tmp);

  sprintf(check,"%d",color->b);
  if (strcmp(tmp, check) || (is_correct_component(color->b)==0))
    exit(Error_args);

  tmp = strtok(NULL, ".");
  if (tmp!=NULL)
    exit(Error_args);
}

bool is_axis(char* arg) {
  if (strcasecmp(arg, "x") && strcasecmp(arg, "y")) {
    exit(Error_args);
  }
  return true;
}

bool invalid_file(char* path_i, char* path_o){
  return !strcmp(path_i, path_o);
}

void help(){
    printf("PNG image processing functions:\n\
    Drawing a circle. Flag to perform this operation: `--circle`.\n\
    Reflection of a specified area. Flag to perform this operation: `--
mirror`.\n\
    Copy a specified area. Flag to perform this operation: `--copy`.\n");
}

void args_parsing(int argc, char** argv, char* path_i, char* path_o, OPTS*
options, int* flag){

    char* last_argument;
    last_argument = strdup(argv[argc-1]);

    const char short_options[] = "hi:o:";
    const struct option long_options[] = {
        {"help", no_argument, NULL, 'h'},
        {"info", no_argument, NULL, 'I'},
        {"input", required_argument, NULL, 'i'},
        {"output", required_argument, NULL, 'o'},
        {"circle", no_argument, NULL, 'c'},
        {"mirror", no_argument, NULL, 'm'},
        {"copy", no_argument, NULL, 'k'},
        {"center", required_argument, NULL, 'R'},
        {"radius", required_argument, NULL, 'r'},
        {"color", required_argument, NULL, 'C'},
        {"thickness", required_argument, NULL, 't'},
        {"axis", required_argument, NULL, 'a'},
        {"left_up", required_argument, NULL, 'u'},
```

```
        {"right_down", required_argument, NULL, 'd'},
        {"fill", no_argument, NULL, 'f'},
        {"fill_color", required_argument, NULL, 'F'},
        {"dest_left_up", required_argument, NULL, 'U'},
        {"contrast", no_argument, NULL, 'K'},
        {"alpha", required_argument, NULL, 'A'},
        {"beta", required_argument, NULL, 'B'},
        {NULL, no_argument, NULL, 0}
    };

    int opt, index = -1;
    while ((opt = getopt_long(argc, argv, short_options, long_options, &index))
!= -1) {
        switch(opt) {
        case 'h':
            help();
            exit(0);
        case 'I':
            options->info_flag = true;
            break;
        case 'i':
            strcpy(path_i, optarg);
            break;
        case 'o':
            strcpy(path_o, optarg);
            break;

        case 'c':
            options->flag = 1;
            break;
        case 'm':
            options->flag = 2;
            break;
        case 'k':
            options->flag = 3;
            break;
        case 'K':
            options->flag = 4;
            break;

        case 'R':
            are_coordinates(optarg, &(options->center_coords));
            options->circle++;
            break;
        case 'r':
            options->radius = to_number(optarg);
            options->circle++;
            break;
        case 'C':
            to_color(optarg, &(options->color));
            options->circle++;
            break;
        case 't':
            options->thickness = to_number(optarg);
            options->circle++;
            break;
        case 'f':
            options->fill_flag = true;
            options->circle++;
            break;
        case 'F':
            to_color(optarg, &(options->fill_color));
            options->fill = true;
```

```
            options->circle++;
            break;

        case 'a':
            options->axis = optarg;
            if (is_axis(options->axis)) options->mirror++;
            break;
        case 'u':
            are_coordinates(optarg, &(options->left_up_coords));
            options->mirror++; options->copy++;
            break;
        case 'd':
            are_coordinates(optarg, &(options->right_down_coords));
            options->mirror++; options->copy++;
            break;

        case 'U':
            are_coordinates(optarg, &(options->dest_left_up_coords));
            options->copy++;
            break;

        case 'A':
            options->alpha = atof(optarg);
            options->contr++;
            break;
        case 'B':
            options->beta = to_number(optarg);
            options->contr++;
            break;

        default:
            exit(Invalid_flag);

        }
        index = -1;
    }
    *flag = options->flag;

    if (optind == argc-1 && strcmp(argv[optind], last_argument) == 0 &&
!options->input_flag) {
        strcpy(path_i, last_argument);
    } else if (optind <= argc-1) {
        exit(Too_many_args);
    }

    if (invalid_file(path_i, path_o)){
        exit(Invalid_file);
    }


}
```

Название файла: args_parsing.h

```
#pragma once

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

```
#include <math.h>
#include <stdbool.h>

#include "structs.h"

#define Too_many_args 40
#define Invalid_file 41
#define Error_read_png 42
#define Error_write_png 43
#define Error_args 44
#define Invalid_flag 45

OPTS init_options();
bool is_correct_component(int val);
int count_dots(char* arg);
void are_coordinates(char *arg, COORD* coords);
int to_number(char* arg);
void to_color(char* arg, RGB* color);
bool is_axis(char* arg);
bool invalid_file(char* path_i, char* path_o);
void help();
void args_parsing(int argc, char** argv, char* path_i, char* path_o, OPTS*
options, int* flag);
```

Название файла: image_processing.c

```
#include "image_processing.h"


void set_color(Png *image, int x, int y, RGB color) {
  if (x < 0 || y < 0 || x >= image->width || y >= image->height)
    return;

  png_bytep px = &(image->row_pointers[y][x * PX_SIZE]);
  px[RED] = color.r;
  px[GREEN] = color.g;
  px[BLUE] = color.b;
}

RGB get_color(Png *image, int x, int y) {
  RGB color;

  png_bytep px = &(image->row_pointers[y][x * PX_SIZE]);
  color.r = px[RED];
  color.g = px[GREEN];
  color.b = px[BLUE];

  return color;
}

int max(int a, int b){
  return a > b ? a : b;
}

int min(int a, int b){
  return a < b ? a : b;
}

void change_coords(COORD* left_up, COORD* right_down){
  int x_change, y_change;
  if (right_down->x < left_up->x){
    x_change = right_down->x;
    right_down->x = left_up->x;
```

```
    left_up->x = x_change;
  }
  if (right_down->y < left_up->y){
    y_change = right_down->y;
    right_down->y = left_up->y;
    left_up->y = y_change;
  }
}

int check_in_pic(Png* image, int x, int y){
  return (x >= 0 && x < image->width && y >= 0 && y < image->height);
}

int valid_circle(int radius, int thickness, bool fill_flag, RGB* color, bool
fill, int circle){
    if(radius <= 0 || thickness <= 0){
      return 0;
    }
    if ((circle == 4 && !fill_flag && !fill) || (circle == 6 && fill_flag &&
fill) || (circle == 5 && !fill_flag && fill)) {
        return 1;
    }
    return 0;
}

int check_on_circle_line(int x, int y, int x0, int y0, int radius, int
thickness){
    int flag1 = (x-x0)*(x-x0) + (y-y0)*(y-y0) <=
(radius+(thickness)/2)*(radius+(thickness)/2);
    int flag2 = (x-x0)*(x-x0) + (y-y0)*(y-y0) >= (max(0, radius-
(thickness)/2))*(max(0, radius-(thickness)/2));
    return flag1 && flag2;
}

int check_in_circle(int x, int y, int x0, int y0, int radius, int thickness)
{
    int flag = (x-x0)*(x-x0) + (y-y0)*(y-y0) <= (radius-thickness/2)*(radius-
thickness/2);
    return flag;
}

void draw_circle(Png* image, COORD center_coords, int radius, int thickness, int
fill_flag, RGB color, RGB fill_color){
  int x0 = center_coords.x;
  int y0 = center_coords.y;
    for (int y = max(0, y0-radius-thickness/2); y <= min(image->height-1,
y0+radius+thickness/2); y++) {
        for (int x = max(0, x0-radius-thickness/2); x <= min(image->width-1,
x0+radius+thickness/2); x++) {
            if (fill_flag && check_in_circle(x, y, x0, y0, radius, thickness)) {
                set_color(image, x, y, fill_color);
            }
            if (check_on_circle_line(x, y, x0, y0, radius, thickness)) {
                set_color(image, x, y, color);
            }
        }
    }
}

void swap(Png* image, int x1, int y1, int x2, int y2){
  RGB color1;
  RGB color2;
```

```
  color1 = get_color(image, x1, y1);
  color2 = get_color(image, x2, y2);

  set_color(image, x2, y2, color1);
  set_color(image, x1, y1, color2);
}

void set_area(int height, int width, COORD* left_up, COORD* right_down) {
  left_up->x = max(0, left_up->x);
  left_up->x = min(width-1, left_up->x);

  left_up->y = max(0, left_up->y);
  left_up->y = min(height-1, left_up->y);

  right_down->x = max(0, right_down->x);
  right_down->x = min(width-1, right_down->x);

  right_down->y = max(0, right_down->y);
  right_down->y = min(height-1, right_down->y);
}

void mirror_x(Png* image, COORD left_up, COORD right_down){
  set_area(image->height, image->width, &left_up, &right_down);

  int diff = right_down.x - left_up.x;

  for (int y = left_up.y; y < right_down.y; y++){
    int n = 0;
    for (int x = left_up.x; x <= left_up.x+(diff-1)/2; x++){
      if (check_in_pic(image, left_up.x+diff-n, y)){
        swap(image, x, y, left_up.x+diff-n, y);
      }
      n++;
    }
  }
}

void mirror_y(Png* image, COORD left_up, COORD right_down){
  set_area(image->height, image->width, &left_up, &right_down);
  RGB* color = (RGB *)malloc(sizeof(RGB) * (right_down.x - left_up.x));

  int n = 0;
  int diff = right_down.y - left_up.y;

  int i = 0;
  for (int x = left_up.x; x < right_down.x; x++){
    if (check_in_pic(image, x, right_down.y)){
      color[i]=get_color(image, x, right_down.y);
      i++;
    }
  }

  for (int y = left_up.y; y <= left_up.y+(diff-1)/2; y++){
    for (int x = left_up.x; x < right_down.x; x++){
      if (check_in_pic(image, x, left_up.y+diff-n)){
        swap(image, x, y, x, left_up.y+diff-n);
      }
    }
    n++;
  }

  i = 0;
  for (int x = left_up.x; x < right_down.x; x++){
```

```
    if (check_in_pic(image, x, right_down.y)){
      set_color(image, x, right_down.y, color[i]);
      i++;
    }
  }
}

void copy_pic(Png* image, COORD left_up, COORD right_down, COORD dest_left_up){
  set_area(image->height, image->width, &left_up, &right_down);

  RGB pixel;
  int diff_x = dest_left_up.x - left_up.x;
  int diff_y = dest_left_up.y - left_up.y;

  RGB** pixel_row_pointer = (RGB **)malloc(sizeof(RGB *) * (right_down.y -
left_up.y + 1));
  for (int y = 0; y < right_down.y - left_up.y + 1; y++) {
    pixel_row_pointer[y] = (RGB *)malloc(sizeof(RGB) * (right_down.x -
left_up.x));
  }

  for (int y = left_up.y; y < right_down.y; y++){
    for (int x = left_up.x; x < right_down.x; x++){
      if (check_in_pic(image, x+diff_x, y+diff_y)){
        pixel = get_color(image, x, y);
        pixel_row_pointer[y - left_up.y][x - left_up.x] = pixel;
      }
    }
  }

  for (int y = left_up.y; y < right_down.y; y++){
    for (int x = left_up.x; x < right_down.x; x++){
      if (check_in_pic(image, x+diff_x, y+diff_y)){
        set_color(image, x+diff_x, y+diff_y, pixel_row_pointer[y - left_up.y][x
- left_up.x]);
      }
    }
  }
}

void contrast(Png* image, float alpha, int beta){
  RGB old;
  RGB new;

  for (int y = 0; y < image->height; y++){
    for (int x = 0; x < image->width; x++){
      old = get_color(image, x, y);
      new.r = (int)floorf(old.r*alpha + beta);
      new.g = (int)floorf(old.g*alpha + beta);
      new.b = (int)floorf(old.b*alpha + beta);
      if (new.r>255){
        new.r=255;
      }
      if (new.g>255){
        new.g=255;
      }
      if (new.b>255){
        new.b=255;
      }
      set_color(image, x, y, new);
    }
  }
}
```

```
void info(Png* image){
    printf("Info about PNG image:\n\
    width, height: %d, %d\n\
    color_type: %d\n\
    bit_depth: %d\n", image->width, image->height, image->color_type, image-
>bit_depth);
}

void choose_func(Png image, int flag, OPTS* options){
    switch(flag){
        case 1:
        if (!valid_circle(options->radius, options->thickness, options-
>fill_flag, &(options->color), options->fill, options->circle)) {
            exit(Error_args);
        }
        draw_circle(&image, options->center_coords, options->radius, options-
>thickness, options->fill_flag, options->color, options->fill_color);
        break;
        case 2:
        if (options->mirror != 3) {
            exit(Error_args);
        }
        change_coords(&(options->left_up_coords), &(options-
>right_down_coords));
        if (!strcmp(options->axis, "x")){
            mirror_x(&image, options->left_up_coords, options-
>right_down_coords);
        }
        else {
            mirror_y(&image, options->left_up_coords, options-
>right_down_coords);
        }
        break;
        case 3:
        if (options->copy != 3) {
            exit(Error_args);
        }
        change_coords(&(options->left_up_coords), &(options-
>right_down_coords));
        copy_pic(&image, options->left_up_coords, options->right_down_coords,
options->dest_left_up_coords);
        break;
        case 4:
        if (options->contr != 2){
            exit(Error_args);
        }
        contrast(&image, options->alpha, options->beta);
        break;
        default:
        if (options->info_flag){
            info(&image);
            exit(0);
        }
    }
}
```

Название файла: image_processing.h

```
#pragma once

#include <math.h>
```

```
#include "args_parsing.h"

#define PX_SIZE 3
#define RED 0
#define GREEN 1
#define BLUE 2

void set_color(Png *image, int x, int y, RGB color);
RGB get_color(Png *image, int x, int y);
int max(int a, int b);
int min(int a, int b);
void change_coords(COORD* left_up, COORD* right_down);
int check_in_pic(Png* image, int x, int y);
int valid_circle(int radius, int thickness, bool fill_flag, RGB* color, bool
fill, int circle);
int check_on_circle_line(int x, int y, int x0, int y0, int radius, int
thickness);
void draw_circle(Png* image, COORD center_coords, int radius, int thickness, int
fill_flag, RGB color, RGB fill_color);
void swap(Png* image, int x1, int y1, int x2, int y2);
void set_area(int height, int width, COORD* left_up, COORD* right_down);
void mirror_x(Png* image, COORD left_up, COORD right_down);
void mirror_y(Png* image, COORD left_up, COORD right_down);
void copy_pic(Png* image, COORD left_up, COORD right_down, COORD dest_left_up);
void contrast(Png* image, float alpha, int beta);
void info(Png* image);
void choose_func(Png image, int flag, OPTS* options);
```

Название файла: png_io.c

```
#include "png_io.h"

void read_png_file(char *file_name, Png *image) {
    int y;
    png_bytep header = (png_bytep)calloc(8, sizeof(png_bytep));

    FILE *fp = fopen(file_name, "rb");
    if (!fp){
        exit(Error_read_png);
    }

    fread(header, 1, 8, fp);
    if (png_sig_cmp(header, 0, 8)){
        exit(Error_read_png);
    }

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

    if (!image->png_ptr){
        exit(Error_read_png);

    }

    image->info_ptr = png_create_info_struct(image->png_ptr);
    if (!image->info_ptr){
        png_destroy_read_struct(&image->png_ptr, NULL, NULL);
        exit(Error_read_png);
    }

    if (setjmp(png_jmpbuf(image->png_ptr))){
        png_destroy_read_struct(&image->png_ptr, &image->info_ptr, NULL);
        exit(Error_read_png);
```

```
    }

    png_init_io(image->png_ptr, fp);
    png_set_sig_bytes(image->png_ptr, 8);

    png_read_info(image->png_ptr, image->info_ptr);

    image->width = png_get_image_width(image->png_ptr, image->info_ptr);
    image->height = png_get_image_height(image->png_ptr, image->info_ptr);
    image->color_type = png_get_color_type(image->png_ptr, image->info_ptr);
    image->bit_depth = png_get_bit_depth(image->png_ptr, image->info_ptr);

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

    if (setjmp(png_jmpbuf(image->png_ptr))){
        exit(Error_read_png);
    }

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

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

    fclose(fp);
}

void write_png_file(char *file_name, Png *image) {
    int y;
    FILE *fp = fopen(file_name, "wb");
    if (!fp){
        exit(Error_write_png);
    }

    image->png_ptr = png_create_write_struct(PNG_LIBPNG_VER_STRING, NULL, NULL,
NULL);

    if (!image->png_ptr){
        fclose(fp);
        exit(Error_write_png);
    }

    image->info_ptr = png_create_info_struct(image->png_ptr);
    if (!image->info_ptr){
        png_destroy_write_struct(&image->png_ptr, NULL);
        fclose(fp);
        exit(Error_write_png);
    }

    if (setjmp(png_jmpbuf(image->png_ptr))){
        fclose(fp);
        exit(Error_write_png);
    }

    png_init_io(image->png_ptr, fp);

    if (setjmp(png_jmpbuf(image->png_ptr))){
        fclose(fp);
        exit(Error_write_png);
    }
```

```
    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))){
        fclose(fp);
        exit(Error_write_png);
    }

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

    if (setjmp(png_jmpbuf(image->png_ptr))){
        fclose(fp);
        exit(Error_write_png);
    }

    png_write_end(image->png_ptr, NULL);

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

    fclose(fp);
}
```

Название файла: png_io.h

```
#pragma once

#include "args_parsing.h"

void read_png_file(char *file_name, Png *image);
void write_png_file(char *file_name, Png *image);
```

Название файла: Makefile

```
TARGET = cw
CC = gcc

LIBS = ./structs ./png_io ./args_parsing ./image_processing

CFLAGS = -Wall -std=gnu99 $(foreach lib, $(LIBS), -I$(lib))

CORE = $(wildcard *.c) $(wildcard **/*.c)
OBJ = $(patsubst */%.c, %.o, $(CORE))

all : $(TARGET)

$(TARGET) : $(OBJ)
                                        $(CC) $(CFLAGS) $^ -lm -lpng -o $@

clean :
                                        rm -rf $(TARGET)
```

ПРИЛОЖЕНИЕ Б
ТЕСТИРОВАНИЕ

Рисование окружности с заливкой: исходное изображение – рис. 1, обработанное – рис. 2.

Входные данные: `./cw --input 1pEgSsirnLM.png --output out1.png --circle --radius 20 --thickness 5 --center 400.350 --color 255.151.187 --fill --fill_color 255.151.187`

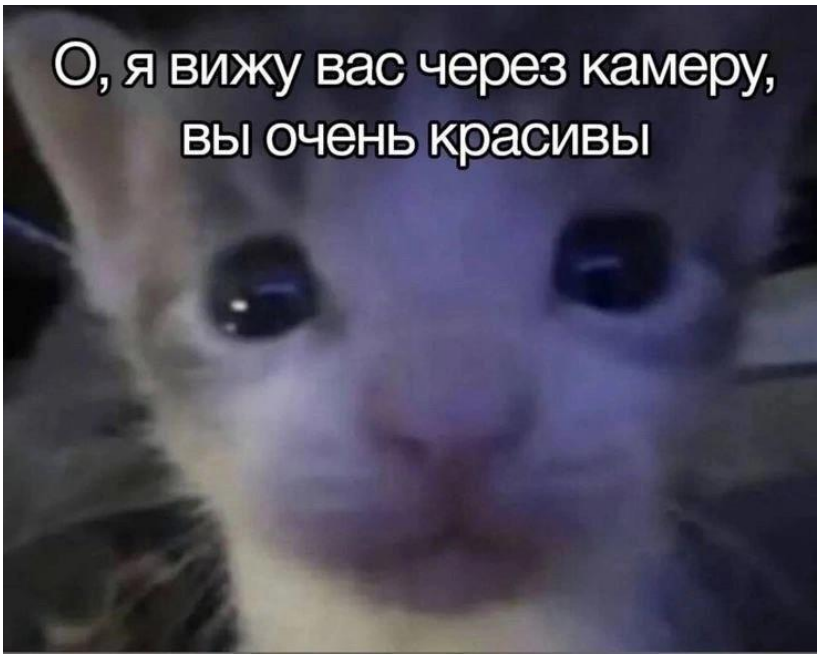

Рисунок 1 – исходное изображение

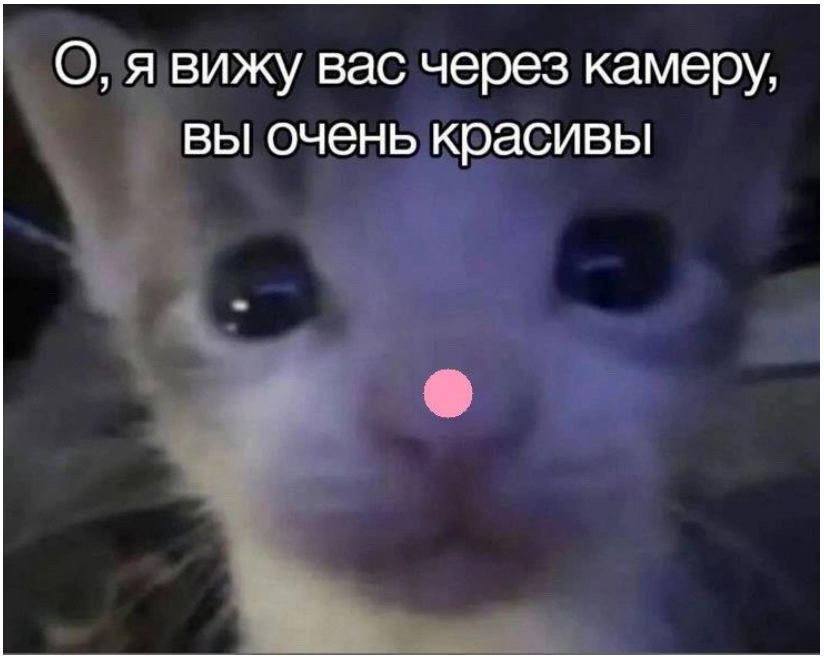

Рисунок 2 – результат рисования круга с заливкой

Рисование окружности без заливки: исходное изображение – рис. 3, обработанное – рис. 4.

Входные данные: `./cw --input ptjaf0Z9lDs.png --output out2.png --circle --radius 100 --thickness 5 --center 260.260 --color 0.0.0`

Рисунок 3 – исходное изображение

Рисунок 4 – результат рисования окружности без заливки

Копирование заданной области: исходное изображение – рис. 5, обработанное – рис. 6.

Входные данные: `./cw --input GvrRuWuMxjo.png -output out3.png --copy --left_up 20.320 --right_down 180.350 --dest_left_up 320.367`

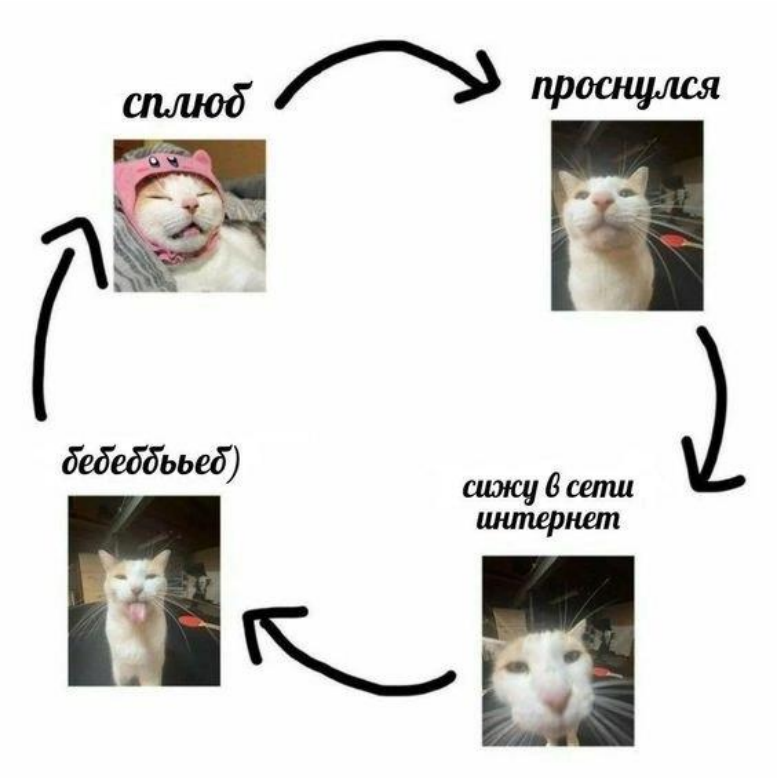

Рисунок 5 – исходное изображение

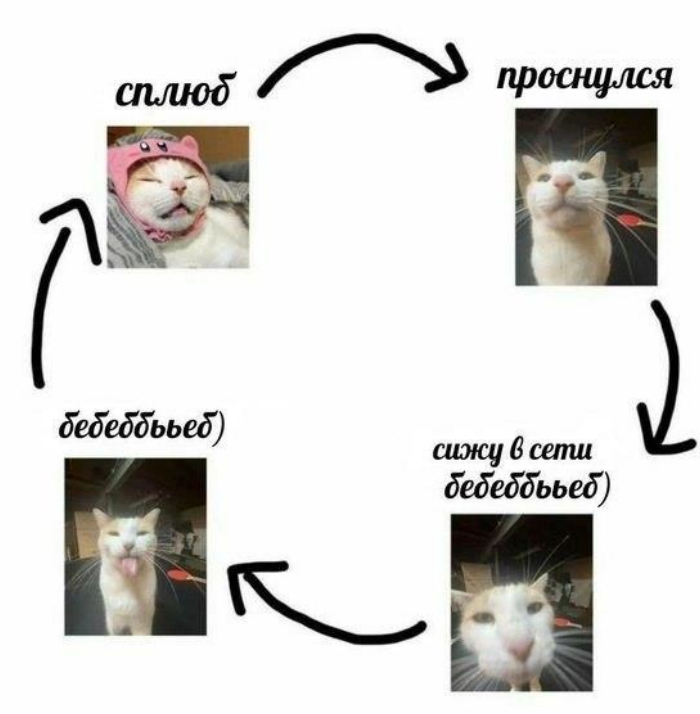

Рисунок 6 – результат копирования заданной области

Копирование заданной области: исходное изображение – рис. 7, обработанное – рис. 8.

Входные данные: `./cw --input gxKIrS0CrEs.png --output out4.png --copy --left_up 250.100 --right_down 321.200 --dest_left_up 160.60`

Рисунок 7 – исходное изображение

Рисунок 8 – результат копирования заданной области

Отражение заданной области по оси y: исходное изображение – рис. 9, обработанное – рис. 10.

Входные данные: `./cw --input wWp7NQbMn-Y.png --output out5.png --mirror --axis y --left_up 120.230 --right_down 630.510`

Рисунок 9 – исходное изображение

Рисунок 10 – результат отражения заданной области по y

Отражение заданной области по оси x (граничный случай, правая нижняя координата выходит за пределы изображения): исходное изображение – рис. 11, обработанное – рис. 12.

Входные данные: `./cw --input vmfgKypLljA.png --output out6.png --mirror --axis x --left_up 850.620 --right_down 980.10000`

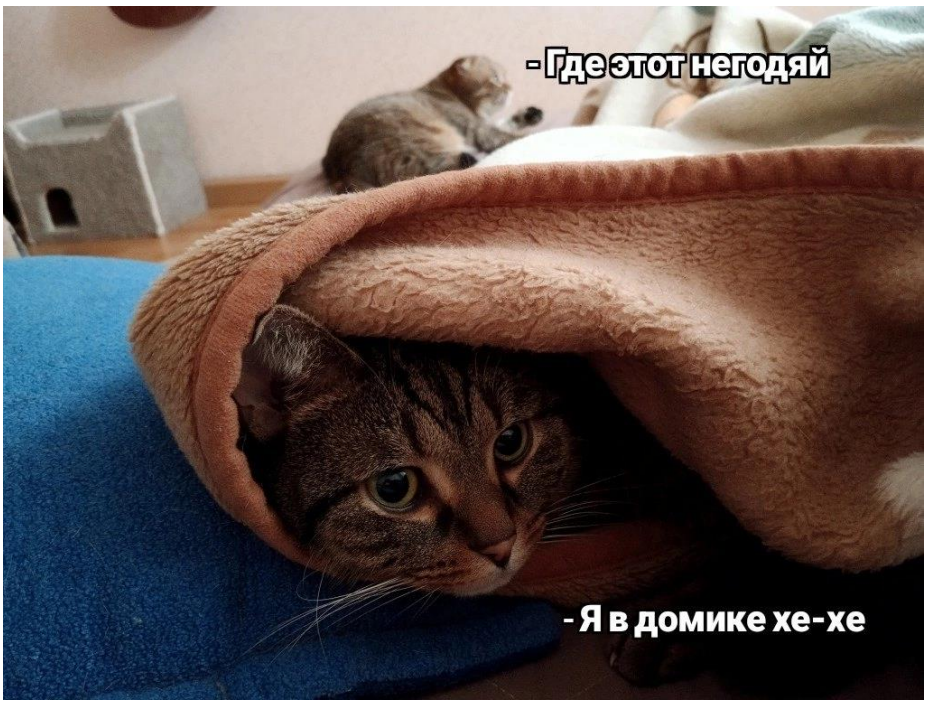

Рисунок 11 – исходное изображение

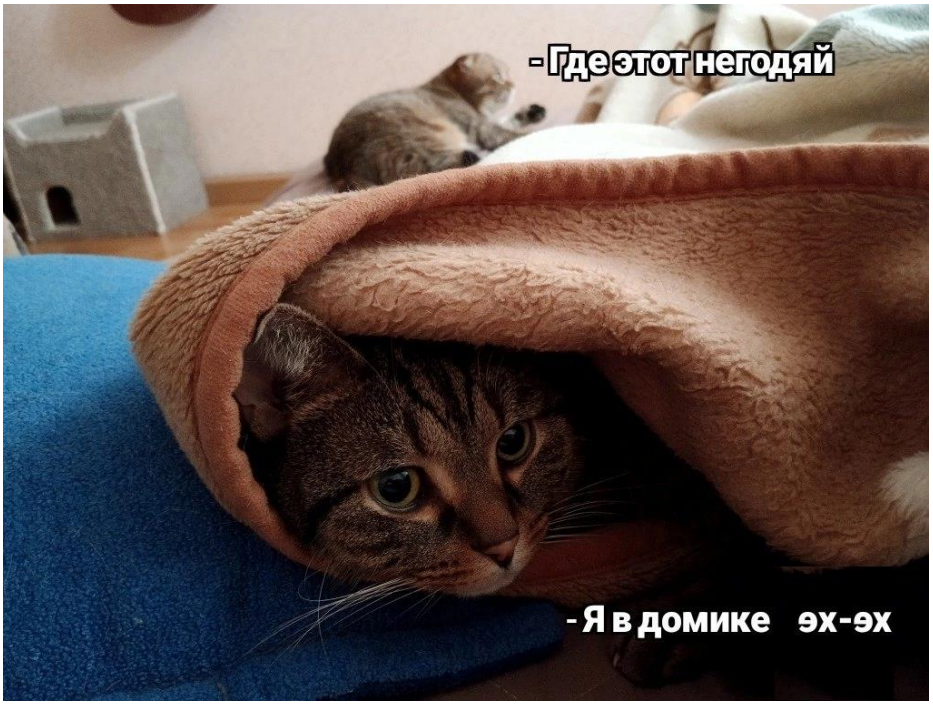

Рисунок 12 – результат отражения заданной области по x

Вывод информации об изображении: рис. 13.

Входные данные: `./cw --input vmfgKypLljA.png --info`

Рисунок 13 – результат введения флага --info

Вывод справки: рис. 14.

Входные данные: `./cw --help`

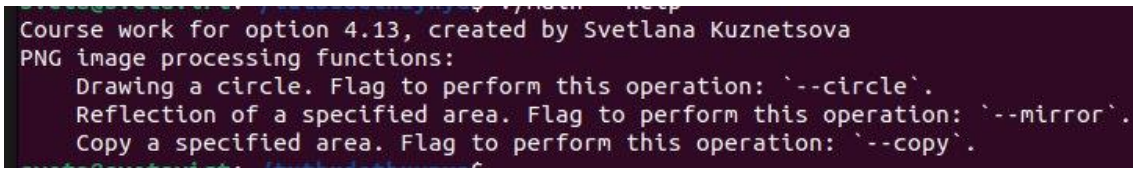

Рисунок 14 – результат введения флага --help