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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Обработка изображений

Студентка гр. 3342 _____ Антипина В.А.

Преподаватель _____ Глазунов С.А.

Санкт-Петербург

2024

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студентка Антипина В.А.

Группа 3342

Тема работы: «Обработка PNG изображения»

Вариант 4.22

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

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

Общие сведения:

Формат картинки PNG (рекомендуем использовать библиотеку libpng), без сжатия, файл может не соответствовать формату PNG, т.е. необходимо проверка на PNG формат. Если файл не соответствует формату PNG, то программа должна завершиться с соответствующей ошибкой.

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

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

Программа должна иметь следующую функции по обработке изображений:

- Заменяет все пиксели одного заданного цвета на другой цвет. Флаг для выполнения данной операции: `--color_replace`. Функционал определяется: Цвет, который требуется заменить. Флаг `--old_color` (цвет задаётся строкой `rrr.ggg.bbb`, где rrr/ggg/bbb – числа, задающие цветовую компоненту. пример `--old_color 255.0.0` задаёт красный цвет).

Цвет на который требуется заменить. Флаг `--new_color` (работает аналогично флагу `--old_color`)

- 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). 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
- Поиск всех залитых прямоугольников заданного цвета. Флаг для выполнения данной операции: `--filled_rects`. Требуется найти все прямоугольники заданного цвета и обвести их линией. Функционал определяется: Цветом искомых прямоугольников. Флаг `--color` (цвет задаётся строкой `rrr.ggg.bbb`, где rrr/ggg/bbb – числа, задающие цветовую компоненту. пример `--color 255.0.0` задаёт красный цвет) Цветом линии для обводки. Флаг `--border_color` (работает аналогично флагу `--color`) Толщиной линии для обводки. Флаг `--thickness`. На вход принимает число больше 0

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

Содержание пояснительной записки:

«Содержание», «Введение», «Заключение», «Список использованных источников».

Предполагаемый объем пояснительной записки:

Не менее 15 страниц.

Дата выдачи задания: 18.03.2024

Дата сдачи реферата: 23.05.2024

Дата защиты реферата: 23.05.2024

Студентка _____ Антипина В.А

Преподаватель _____ Глазунов С.А.

АННОТАЦИЯ

В ходе курсовой работы реализована программа, осуществляющая обработку PNG изображения. Для взаимодействия с программой реализован интерфейс командной строки (CLI). Программа реализует следующие функции: перекрашивание пикселей заданного цвета в другой, создание узора вокруг изображения заданного вида, цвета и толщины, выделение прямоугольных областей заданного цвета.

SUMMARY

During the course work, a program is implemented that processes the PNG image. To interact with the program, a command line interface (CLI) is implemented. The program implements the following functions: recoloring pixels of a given color to another, creating a pattern around an image of a given appearance, color and thickness, selecting rectangular areas of a given color.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель работы: изучить структуру PNG изображения, научиться работать с PNG изображением на языке программирования C с помощью библиотеки libpng, реализовать функции для работы с этим форматом.

Достижение этой цели включает 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. РЕАЛИЗАЦИЯ ВВОДА

В функции read_png_file осуществляется чтение PNG изображения. Функция в качестве одного из аргументов принимает на вход указатель на структуру Png. Эта структура содержит поля для хранения значений ширины и высоты изображения, типа цвета, который используется в изображении, глубины цвета, количества проходов, необходимых, чтобы полностью обработать изображение, указатель на массив строк пикселей изображения, на struct_png, info_png — основные структуры библиотеки. В функции открывается файл , его первые 8 байт сравниваются с сигнатурой png-файла, и, если они совпадают, инициализируется структура PNG. Выделяется память под png_struct и png_info , затем заполняются поля описанной ранее структуры Png. Выделяется память под двумерный массив пикселей. В цикле выделяется память под каждую строку пикселей. Массив заполняется. Файл закрывается.

Чтобы вызвать определённую функцию, пользователь вводит соответствующие флаги в консоль. Обработка этих флагов реализована с помощью функции getopt_long. При этом предусмотрено выполнение только одной из функций, введённых пользователем — первой (то есть, к примеру, для флагов --filled_rects, --ornament с соответствующими параметрами будет вызвана только первая функция, вторая проигнорирована. Если у двух функций есть параметры с совпадающими именами, они также не перезаписываются).

2. РЕАЛИЗАЦИЯ ОСНОВНЫХ ФУНКЦИЙ ПРОГРАММЫ

2.1. Реализация первой функции

Функция color_replace вызывается, если был введён соответствующий флаг. Функция принимает на вход указатель на структуру Png и два массива чисел, содержащих информацию о цвете, который нужно заменить, и новом цвете. Цвета вводятся пользователем в консоль, поэтому в программе реализована функция, преобразовывающая строку в массив целых чисел. В функции find_colors проверяется корректность введённых данных, с помощью функции strtok осуществляется разделение строки на токены, функцией atoi они преобразовываются в числа, которые затем добавляются в массив. В функции color_replace последовательно проверяются пиксели каждого ряда массива. Если цвет очередного пикселя равен тому, что нужно заменить, его данные о цвете заменяются на новые.

2.2. Реализация второй функции

Функция ornament так же, как и предыдущая, запускается при вводе соответствующего флага. При этом необходимо ввести тип орнамента (в противном случае программа завершится с ошибкой) и параметры, обязательные для этого типа. В соответствии с выбранным типом функция вызывает одну из следующих функций: circle, rectangle и semicircles, которые обрабатывают изображение. Функция circle представляет из себя реализацию алгоритма Брезенхема для рисования окружности с дополнительным вызовом функций draw_before_and_after (которая «заливает» область над и под окружностью, если это необходимо) draw_except, которая закрашивает все пиксели слева и справа от пикселей, образующих окружность, расположенных в одном ряду. Каждый пиксель перед рисованием проверяется на принадлежность изображению. Функция rectangle одними из аргументов получает на вход значения толщины и количества рамок. Толщина определяет здесь количество рамок одинарной толщины, расположенных рядом, и ширину

незакрашенной области между двумя «толстыми» рамками. Количество — количество «толстых» рамок. Поэтому в данной функции count раз вызывается thickness функций draw_rectangle от угловых точек, координаты которых увеличиваются на единицу при каждой итерации, и после этого увеличиваются значения координат на thickness, чтобы оставить область между рамками незакрашенной. В функции draw_rectangle определяются координаты угловых точек, затем четыре раза вызывается функция draw_line — реализация алгоритма Брезенхема для рисования линии. Функция semicircles определяет центр первой окружности на каждой стороне изображения и рисует эту окружность, затем в цикле вычисляет координаты других центров (прибавляя к исходным длину промежутка, на что делятся значения длины и ширины количеством окружностей) и рисует и их. «Рисование» - вызов функции circle_thick, которая, в свою очередь, является реализацией алгоритма Мичнера рисования окружности (модифицированный алгоритм Брезенхема), где вместо пикселя ставится круг диаметра, равного толщине. Круг рисуется с помощью функции draw_circle_n — очередной реализации алгоритма Брезенхема, где вызывается функция draw_between для заливки ряда между двумя противоположными точками окружности.

2.3. Реализация третьей функции

Функция filled_rects создаёт односвязный список — указатель на структуру ListXY, в которой будут храниться координаты верхнего левого и правого нижнего углов прямоугольной области и указатель на следующий элемент. В цикле последовательно проверяются пиксели в каждом ряду изображения. Если встретился пиксель нужного цвета, его координаты сохраняются, далее проверяются все пиксели этого же цвета далее в ряду до тех пор, пока не найден пиксель другого цвета. Тогда определяется абсцисса второй точки прямоугольной области. Далее от первой найденной точки осуществляется «движение вниз», то есть проверяются все пиксели в последующих рядах с этой же абсциссой. Когда находится первый пиксель

другого цвета, сохраняется ордината второй точки. Аналогичным образом обрабатываются все прямоугольные области заданного цвета на изображении. По завершении циклов для каждого элемента односвязного списка вызывается функция rectangle_in, которая своей реализацией несколько похожа на rectangle.

3. РЕАЛИЗАЦИЯ ВЫВОДА

3.1. Запись изображения

Создаётся файл с именем по умолчанию «res.png» (может быть изменено пользователем при вводе флага -i/--input). Инициализируется структура Png, записывается заголовок (иначе файл не откроется). Определяется чанк IHDR, записываются все данные из структуры. Очищается память из-под элементов двумерного массива.

ТЕСТИРОВАНИЕ

Рисунок 1 — исходное изображение

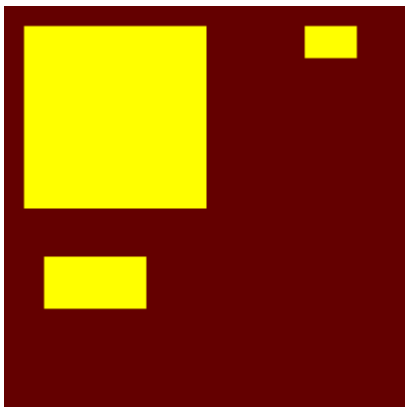

Рисунок 2 — результат работы функции color_replace

1. Тестирование функции color_replace:

Аргументы для запуска: ./a.out -i img.png --color_replace --old_color 255.255.255 --new_color 100.0.0

2. Тестирование функции ornament:

Аргументы для запуска: ./a.out --ornament --pattern rectangle --color 78.78.78 --thickness 15 --count 3 img.png

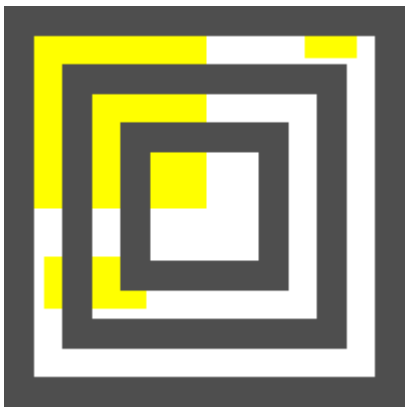

Рисунок 3 — результат работы функции ornament типа rectangle

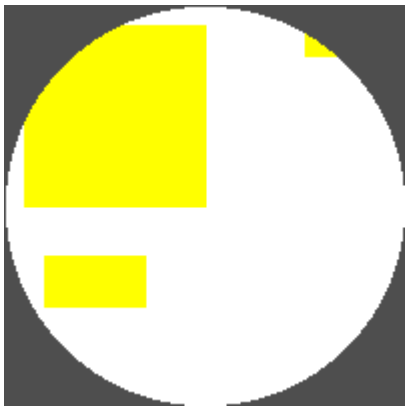

Рисунок 4 — результат работы функции ornament типа circle

Аргументы для запуска: ./a.out --ornament --pattern circle --color 78.78.78 img.png

Аргументы для запуска: ./a.out --ornament --pattern semicircles --color 78.78.78 --thickness 5 --count 7 img.png

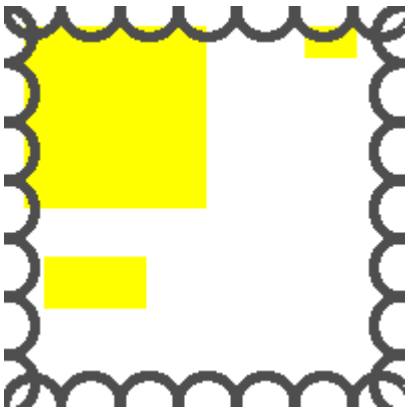

Рисунок 5 — результат работы функции ornament типа semicircles

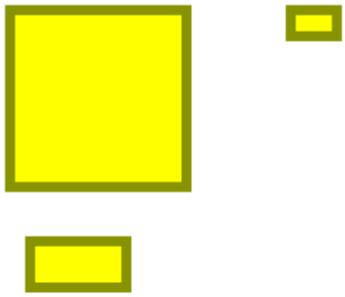

Рисунок 6 — результат работы функции filled_rects

3. Тестирование функции filled_rects:

Аргументы для запуска: ./a.out --ornament --filled_rects --color 255.255.0 --border_color 135.147.1 --thickness 5 img.png

4. Тестирование обработки ошибок:

- Проверка корректности введённого кода цвета

Аргументы для запуска: ./a.out --ornament --filled_rects --color 2o55.255.0 --border_color 250.21a8.221 --thickness 5 img.png -o r.png

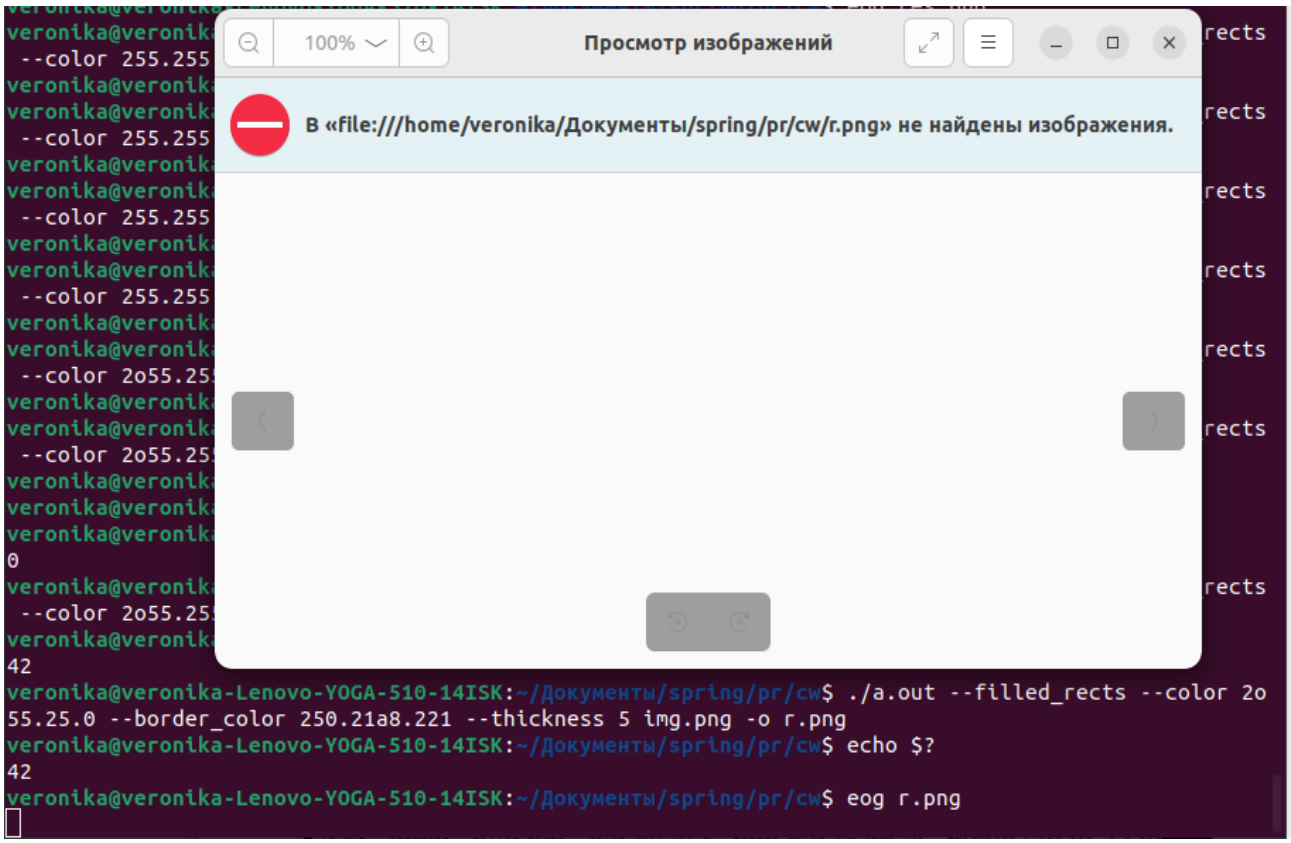

Рисунок 7 — демонстрация завершения с ошибкой

- Проверка обработки лишних аргументов

Аргументы для запуска: /a.out --ornament --pattern circle --thickness 5 --color 255.45.90 img.png

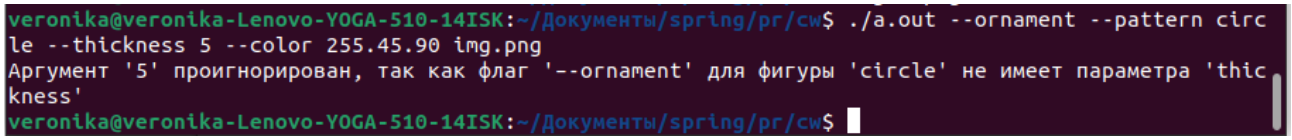

Рисунок 8 — вывод сообщения об ошибке

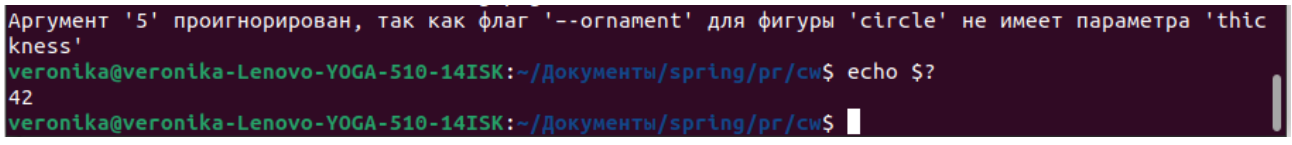

Рисунок 9 — программа завершилась с кодом ошибки 42

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы была создана программа на языке программирования C, осуществляющая обработку PNG изображения. В зависимости от выбранных опций, программа выполняет одну из поддерживаемых функций. С помощью функции getopt_long программа обрабатывает флаги, введённые пользователем.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Базовые сведения к выполнению курсовой работы по дисциплине "Программирование". Второй семестр: учеб.-метод. пособие. СПб.: Изд-во СПбГЭТУ "ЛЭТИ", 2024. 36 с.
2. Керниган Б., Ритчи Д. Язык программирования Си\ Пер. с англ., 3-е изд., испр. — СПб.: "Невский Диалект", 2001. - 352 с: ил.

ПРИЛОЖЕНИЕ А — ИСХОДНЫЙ КОД ПРОГРАММЫ

Имя файла: main.c

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

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

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

struct ListXY{
     int a;
     int b;
     int c;
     int d;
     struct ListXY* next;
};

struct ListXY* new_element(int a, int b, int c, int d){
        struct  ListXY*  cur  =  (struct  ListXY*)malloc(sizeof(struct
ListXY));
     if(!cur)
          exit(45);
        cur->a = a;
     cur->b = b;
     cur->c = c;
     cur->d = d;
        cur->next = NULL;
        return cur;
}

void push(int a, int b, int c, int d, struct ListXY* head){
     struct ListXY* cur = head;
     while(cur->next!=NULL)
          cur = cur->next;
     cur->next = new_element(a, b, c, d);
}

void read_png_file(char* file_name, struct Png* image){
     int x,y;
     char header[8];
     FILE *fp = fopen(file_name, "rb");
     if(!fp){
          printf("Cannot read file: %s\n", file_name);
          exit(42);
```

```
    }

    fread(header, 1, 8, fp);
    if(png_sig_cmp(header,0,8)){
        printf("probably, %s is not a png\n", file_name);
        exit(42);
    }

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


    if(!image->png_ptr){
        printf("error in png structure\n");
        exit(45);
    }

    image->info_ptr = png_create_info_struct(image->png_ptr);
    if(!image->info_ptr){
        printf("error in png info-structure\n");
        exit(45);
    }

    if(setjmp(png_jmpbuf(image->png_ptr))){
        printf("Error\n");
        fclose(fp);
        exit(42);
    }


    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);

    image->row_pointers  =  (png_bytep  *)malloc(sizeof(png_bytep)*image-
>height);
    if(!image->row_pointers)
        exit(45);
    for(y=0; y<image->height; y++){
        image->row_pointers[y]                                          =
(png_byte*)malloc(png_get_rowbytes(image->png_ptr,image->info_ptr));
        if(!image->row_pointers[y])
            exit(45);
    }
    png_read_image(image->png_ptr, image->row_pointers);
```

```
     if(png_get_color_type(image->png_ptr,                          image-
>info_ptr)!=PNG_COLOR_TYPE_RGB){
          printf("It's a RGBA! Error\n");
          exit(43);
     }



     fclose(fp);
}

void write_png_file(char* file_name, struct Png* image){
     int x,y;
     FILE* fp = fopen(file_name, "wb");
     if(!fp){
          printf("Could not open file\n");
          exit(45);
     }

     image->png_ptr    =    png_create_write_struct(PNG_LIBPNG_VER_STRING,
NULL, NULL, NULL);
     if(!image->png_ptr){
          printf("Could not create structure\n");
          fclose(fp);
          exit(45);
     }
     image->info_ptr = png_create_info_struct(image->png_ptr);
     if(!image->info_ptr){
          printf("Error\n");
          fclose(fp);
          exit(45);
     }

     if(setjmp(png_jmpbuf(image->png_ptr))){
          printf("Problem reading file\n");
          fclose(fp);
          exit(42);
     }

     png_init_io(image->png_ptr, fp);

     if(setjmp(png_jmpbuf(image->png_ptr))){
          printf("Header problem\n");
          fclose(fp);
          exit(42);
     }

     png_set_IHDR(image->png_ptr,  image->info_ptr,  image->width,  image-
>height,    image->bit_depth,    image->color_type,    PNG_INTERLACE_NONE,
PNG_COMPRESSION_TYPE_BASE, PNG_FILTER_TYPE_BASE);
     png_write_info(image->png_ptr, image->info_ptr);
     if(setjmp(png_jmpbuf(image->png_ptr))){
          printf("Error: header\n");
          fclose(fp);
          exit(42);
     }
     png_write_image(image->png_ptr, image->row_pointers);
```

```
     if(setjmp(png_jmpbuf(image->png_ptr))){
          printf("Error\n");
          fclose(fp);
          exit(42);
     }
     png_write_end(image->png_ptr, NULL);

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

void process_file(struct Png* image){
     int x,y;
     int color;
     if(png_get_color_type(image->png_ptr,                              image-
>info_ptr)==PNG_COLOR_TYPE_RGB){
          color=3;
     }

        if(png_get_color_type(image->png_ptr,                           image-
>info_ptr)==PNG_COLOR_TYPE_RGBA){
                color=4;
          exit(43);
        }

     for(y=0;y<image->height;y++){
          png_byte* row = image->row_pointers[y];
          for(x=0;x<image->width;x++){
               png_byte* ptr = &(row[x*color]);
               ptr[0] = 0;
               ptr[1] = 0;
          }

     }

}

int* find_colors(char* string){
     if(!strcmp(string,"rrr.ggg.bbb")){
     //   printf("Too few arguments! Type --help, -h.\n");
          exit(42);
     }

        int* array_of_nums = (int*)malloc(3*sizeof(int));
     if(!array_of_nums)
          exit(45);
     char* token = strtok(string, ".");

        if(token==NULL){
              //  printf("Some problems with colors\n");
                exit(42);
        }

        for(int i = 0; i<3; i++){
```

```
          if(token!=NULL){
               for(int i = 0;i<strlen(token);i++)
                    if(!isdigit(token[i])){
                         exit(42);
                    }
                    array_of_nums[i] = atoi(token);

                    if(i!=2)
                    token = strtok(NULL, ".");

                    if(array_of_nums[i]>255||array_of_nums[i]<0){
                //           printf("Some problems with colors\n");
                         exit(42);
                    }
          }
        }
        return array_of_nums;
}

int sgn(int x){
     if(x>0)
          return 1;
     if(x<0)
          return -1;
     return 0;
}

int is_correct(struct Png* image, int x, int y){
        if(x<0||y<0||x>image->width-1||y>image->height-1){
                return 0;
        }
        return 1;
}

void draw_pixel(struct Png* image, int x, int y, int* set_col){
     if(is_correct(image, x, y)){
          png_byte* row = image->row_pointers[y];
          png_byte* ptr = &(row[x*3]);
          ptr[0] = set_col[0];
          ptr[1] = set_col[1];
          ptr[2] = set_col[2];
     }
}

void draw_line(struct Png* image, int x0, int y0, int x1, int y1, int*
set_col){
     int x = x0;
     int y = y0;
     int error = 0;

     int A = y1-y0;
     int B = x0-x1;
     int dx = -sgn(B);
     int dy = sgn(A);

     if((A==0&&B!=0)||(A!=0&&B==0)){
          while(x!=x1||y!=y1){
```

```
                if(is_correct(image,x,y))
                    draw_pixel(image, x, y, set_col);
                if(A==0)
                    x+=dx;
                else
                    y+=dy;
            }
            if(is_correct(image,x,y))
                draw_pixel(image, x, y, set_col);
        }else{
            while(x!=x1||y!=y1){
                if(is_correct(image,x,y)){
                        draw_pixel(image, x, y, set_col);
                }

                int ex = error + A*dx;
                int ey = error + B*dy;

                if(ex*sgn(ex)<ey*sgn(ey)){
                    x+=dx;
                    error = ex;
                }else{
                    y+=dy;
                    error = ey;
                }
            }
            if(is_correct(image,x,y))
                draw_pixel(image, x, y, set_col);

        }
}


void color_replace(struct Png* image, int* old_ones, int* new_ones){
        int x, y;
        for(y=0;y<image->height;y++){
            png_byte* row = image->row_pointers[y];

            for(x=0;x<image->width;x++){
                png_byte* ptr = &(row[x*3]);

        if(ptr[0]==old_ones[0]&&ptr[1]==old_ones[1]&&ptr[2]==old_ones[2]){
                    ptr[0] = new_ones[0];
                    ptr[1] = new_ones[1];
                    ptr[2] = new_ones[2];
                }
            }
        }
}

void draw_rectangle(struct Png* image, int x0, int y0, int* color){
        int x3 = x0;
        int y3 = image->height - y0 -1;
        int x1 = image->width - x0 - 1;
        int y1 = y0;
        int x2 = x1;
        int y2 = y3;
```

```
//оптимизация: цвет в массив чисел в инте -- Done
     draw_line(image,x0,y0,x1,y1,color);
     draw_line(image,x1,y1,x2,y2,color);
     draw_line(image,x2,y2,x3,y3,color);
     draw_line(image,x3,y3,x0,y0,color);
}

void rectangle(struct Png* image, int* color, int thickness, int count){
     int x = 0;
     int y = 0;
     for(int r_c = 0; r_c < count; r_c++){
             draw_rectangle(image,x,y,color);
          for(int i = 0; i<thickness-1;i++){
               x+=1;
               y+=1;
               draw_rectangle(image,x,y,color);
          }
          x+=thickness+1;
          y+=thickness+1;
     }
}
void draw_except(struct Png* image, int x0, int y0, int x1, int y1, int*
color){
        png_byte* row = image->row_pointers[y0];
     for(int i = 0; i<x0; i++){
          png_byte* ptr = &(row[i*3]);
          ptr[0] = color[0];
          ptr[1] = color[1];
          ptr[2] = color[2];
     }
     row = image->row_pointers[y1];
     for(int k = x1; k<image->width; k++){
          png_byte* ptr = &(row[k*3]);
          ptr[0] = color[0];
          ptr[1] = color[1];
          ptr[2] = color[2];
     }
}

void draw_before_and_after(struct Png* image, int y1, int y2, int*
color){
     for(int y0 = 0; y0<y1; y0++){
          png_byte* row = image->row_pointers[y0];
          for(int i = 0; i<image->width; i++){
               png_byte* ptr = &(row[i*3]);
               ptr[0] = color[0];
               ptr[1] = color[1];
               ptr[2] = color[2];
          }
     }
     for(int y0 = y2; y0<image->height; y0++){
          png_byte* row = image->row_pointers[y0];
          for(int i = 0; i<image->width; i++){
               png_byte* ptr = &(row[i*3]);
               ptr[0] = color[0];
               ptr[1] = color[1];
```

```
                ptr[2] = color[2];
            }
        }
}

void circle(struct Png* image, int x0, int y0, int r, int* color){//рамка
в виде большого круга
        int x = 0;
        int y = r;
        int delta = 1-2*r;
        int error = 0;
        draw_before_and_after(image, y0-y, y0+y, color);
        while(y>=x){
            draw_pixel(image, x0+x, y0+y, color);
            draw_pixel(image, x0-x, y0+y, color);
            if(is_correct(image,x0-x,y0+y)&&is_correct(image,x0+x,y0+y))
                draw_except(image, x0-x, y0+y, x0+x, y0+y, color);


            draw_pixel(image, x0+x, y0-y, color);
            draw_pixel(image, x0-x, y0-y, color);
                  if(is_correct(image, x0-x, y0-y)&&is_correct(image, x0+x,
y0-y))
                draw_except(image, x0-x, y0-y, x0+x, y0-y, color);

            draw_pixel(image, x0+y, y0+x, color);
            draw_pixel(image, x0-y, y0+x, color);
                  if(is_correct(image, x0-y, y0+x)&&is_correct(image, x0+y,
y0+x))
                draw_except(image, x0-y, y0+x, x0+y, y0+x, color);

            draw_pixel(image, x0+y, y0-x, color);
            draw_pixel(image, x0-y, y0-x, color);
                  if(is_correct(image, x0-y, y0-x)&&is_correct(image, x0+y,
y0-x))
                draw_except(image, x0-y, y0-x, x0+y, y0-x, color);

            error = 2*(delta+y)-1;
            if((delta<0)&&(error<=0)){
                delta+=2*(++x)+1;
                continue;
            }
            if((delta>0)&&(error>0)){
                delta-=2*(--y)+1;
                continue;
            }
            delta+=2*(++x-(--y));
        }
}

void draw_between(struct Png* image, int x0, int y0, int x1, int y1, int*
color){//заливка участка между пикселями
          png_byte* row = image->row_pointers[y0];
          for(int i = x0; i<x1; i++){
                  png_byte* ptr = &(row[i*3]);
                  ptr[0] = color[0];
                  ptr[1] = color[1];
```

```
                ptr[2] = color[2];
        }
}


void draw_circle_n(struct Png* image, int x0, int y0, int r, int*
color){//рисуем круг нужного цвета
        int x = 0;
        int y = r;
        int delta = 1-2*r;
        int error = 0;
        while(y>=x){

          draw_pixel(image, x0+x, y0+y, color);
          draw_pixel(image, x0+x, y0-y, color);//ставим точки параллельно
Oy на расстоянии радиуса от центра, симметрично.; ставим точки справа, на
расстоянии 1; диагональ справа два
          draw_pixel(image, x0-x, y0+y, color);
          draw_pixel(image, x0-x, y0-y, color);//на первой итерации те же
точки, на второй те же слева на единицу по Ox; на третьей диагональ слева
два
          draw_pixel(image, x0+y, y0+x, color);
          draw_pixel(image,  x0+y,  y0-x,  color);//на  второй  итерации
ставим по две точки над и под правой от радиуса
          draw_pixel(image, x0-y, y0+x, color);
          draw_pixel(image,  x0-y,  y0-x,  color);//на  первой  итерации
ставим  две  точки  параллельно  Ox  на  расстоянии  p,  симметрично
(дублирование)

                if(is_correct(image, x0-x, y0+y)&&is_correct(image, x0+x,
y0+y))
                        draw_between(image,  x0-x,  y0+y,  x0+x,  y0+y,
color);
                if(is_correct(image, x0-x, y0-y)&&is_correct(image, x0+x,
y0-y))
                        draw_between(image,  x0-x,  y0-y,  x0+x,  y0-y,
color);
                if(is_correct(image, x0-y, y0+x)&&is_correct(image, x0+y,
y0+x))
                        draw_between(image,  x0-y,  y0+x,  x0+y,  y0+x,
color);
                if(is_correct(image, x0-y, y0-x)&&is_correct(image, x0+y,
y0-x))
                        draw_between(image,  x0-y,  y0-x,  x0+y,  y0-x,
color);


          error = 2*(delta+y)-1;
                if((delta<0)&&(error<=0)){
                        delta+=2*(++x)+1;
                        continue;
                }
                if((delta>0)&&(error>0)){
                        delta-=2*(--y)+1;
                        continue;
                }
                delta+=2*(++x-(--y));
```

```
        }
}

void draw_line_thick(struct Png* image, int x0, int y0, int x1, int y1,
int* color, int thickness){
        int x = x0;
        int y = y0;
        int error = 0;

        int A = y1-y0;
        int B = x0-x1;
        int dx = -sgn(B);
        int dy = sgn(A);

        if((A==0&&B!=0)||(A!=0&&B==0)){
                while(x!=x1||y!=y1){
                                draw_circle_n(image,  x,  y,  thickness/2,
color);
                        if(A==0)
                                x+=dx;
                        else
                                y+=dy;
                }
                        draw_circle_n(image, x, y, thickness/2, color);
        }else{
                while(x!=x1||y!=y1){
                        draw_circle_n(image, x, y, thickness/2, color);

                        int ex = error + A*dx;
                        int ey = error + B*dy;

                        if(ex*sgn(ex)<ey*sgn(ey)){
                                x+=dx;
                                error = ex;
                        }else{
                                y+=dy;
                                error = ey;
                        }
                }
                        draw_circle_n(image, x, y, thickness/2, color);

        }
}


void plot_circle(struct Png* image,int x, int y, int x_center, int
y_center, int* color, int thickness)//рисуем круги вместо пикселей для
толщины
{
    draw_circle_n(image,x_center+x,y_center+y,thickness/2,color);
    draw_circle_n(image,x_center-x,y_center+y,thickness/2,color);
    draw_circle_n(image,x_center+x,y_center-y,thickness/2,color);
    draw_circle_n(image,x_center-x,y_center-y,thickness/2,color);
}

/* Вычерчивание окружности с использованием алгоритма Мичнера */
```

```
void circle_thick(struct Png* image, int x_center, int y_center, int
radius, int* color, int thickness)//рисуем окружности заданной толщины
{
    int x,y,delta;
    x = 0;
    y = radius;
    delta=3-2*radius;
    while(x<y) {
        plot_circle(image,x,y,x_center,y_center,color,thickness);
        plot_circle(image,y,x,x_center,y_center,color,thickness);
        if (delta<0)
            delta+=4*x+6;
        else {
            delta+=4*(x-y)+10;
            y--;
        }
        x++;
    }

    if(x==y) plot_circle(image,x,y,x_center,y_center,color,thickness);
}

void semicircles(struct Png* image, int* color, int thickness, int
count){
     int parts_h = image->width/count;
     if(image->width%count!=0)
          parts_h++;
     int x0_h = parts_h/2;
     int r_x = parts_h/2;
     if(parts_h%2!=0)
          r_x++;
     int f_x = 0;

     circle_thick(image, x0_h, 0, x0_h, color, thickness);
     circle_thick(image, x0_h, image->height-1, x0_h, color, thickness);

     int parts_v = image->height/count;
     if(image->height%count!=0)
          parts_v++;
     int y0_v = parts_v/2;
     int r_y = parts_v/2;
     if(parts_v%2!=0)
          r_y++;
     int f_y = 0;

     circle_thick(image, 0, y0_v, y0_v, color, thickness);
     circle_thick(image, image->width-1, y0_v, y0_v, color, thickness);

     for(int i = 1; i < count; i++){
          x0_h+=parts_h;
          circle_thick(image, x0_h, 0, r_x, color, thickness);
          circle_thick(image,   x0_h,   image->height-1,   r_x,   color,
thickness);

          y0_v+=parts_v;
          circle_thick(image, 0, y0_v, r_y, color, thickness);
```

```
            circle_thick(image,    image->width-1,    y0_v,    r_y,    color,
thickness);
        }
}

void  ornament(struct  Png*  image,  int*  color,  char*  pattern,  int
thickness, int count){
        int centre_y = image->height/2;
        int centre_x = image->width/2;
        int min;
        if(centre_x<centre_y){
                min = centre_x;
        }else{
                min = centre_y;
        }

        if((thickness<=0||count<=0)&&strcmp(pattern, "circle"))
                exit(42);

    if(!strcmp(pattern,"circle")){
         circle(image, centre_x, centre_y, min, color);
    }else if(!strcmp(pattern, "rectangle")){
         rectangle(image, color, thickness, count);
    }else if(!strcmp(pattern, "semicircles")){
         semicircles(image, color, thickness, count);
    }else{
    //   printf("Incorrect type. Check --help/-h.\n");
         exit(42);
    }
}

void rectangle_f(struct Png* image, int x0, int y0, int x1, int y1, int*
color, int thickness){//для рисования рамки с толщиной (пиксели - круги)
    draw_line_thick(image, x0, y0, x1, y0, color, thickness);
    draw_line_thick(image, x0, y1, x1, y1, color, thickness);
    draw_line_thick(image, x0, y0, x0, y1, color, thickness);
    draw_line_thick(image, x1, y0, x1, y1, color, thickness);
}

void rectangle_in(struct Png* image, int x0, int y0, int x1, int y1, int*
border_color,  int  thickness){//для  рисования  толщины  рамки  внутрь
(пиксели - пиксели)
    for(int i = 0; i<thickness; i++){
         draw_line(image, x0, y0, x1, y0, border_color);
         draw_line(image, x0, y1, x1, y1, border_color);
         draw_line(image, x0, y0, x0, y1, border_color);
         draw_line(image, x1, y0, x1, y1, border_color);
         if(x0+1<=x1-1&&y0+1<=y1-1){
              x0++;
              x1--;
              y0++;
              y1--;
         }
         else
              return;
    }
}
```

```
void filled_rects(struct Png* image, int* color, int* border_color, int
thickness){
        int x, y;
     int x0, y0, x1, y1, x_h;
     int a = -1;
     int b = -1;
     int c = -1;
     int d = -1;
     int y_s;

     struct ListXY* list;
     int count = 0;

        for(y=0;y<image->height;y++){
                png_byte* row = image->row_pointers[y];
          y_s = y;
                for(x=0;x<image->width;x++){
                        png_byte* ptr = &(row[x*3]);

if(ptr[0]==color[0]&&ptr[1]==color[1]&&ptr[2]==color[2]){
                    count++;
                    x0 = x;
                    y0 = y;
                    x1 = x;
                    y1 = y;
                    x_h = x0;


     while(ptr[0]==color[0]&&ptr[1]==color[1]&&ptr[2]==color[2]){
                         x1++;
                         x++;
                         ptr = &(row[x1*3]);
                    }
                    if(y0>0){
                         row = image->row_pointers[y0-1];
                         ptr = &(row[x0*3]);

     if((ptr[0]==color[0]&&ptr[1]==color[1]&&ptr[2]==color[2])||(ptr[0]==
border_color[0]&&ptr[1]==border_color[1]&&ptr[2]==border_color[2]))
                                        continue;
                    }
                    if(x0>0){
                         row = image->row_pointers[y0];
                         ptr = &(row[(x0-1)*3]);

if((ptr[0]==color[0]&&ptr[1]==color[1]&&ptr[2]==color[2])||(ptr[0]==borde
r_color[0]&&ptr[1]==border_color[1]&&ptr[2]==border_color[2]))
                                                continue;
                    }

                    row = image->row_pointers[y0];
                    ptr = &(row[x0*3]);

     while(ptr[0]==color[0]&&ptr[1]==color[1]&&ptr[2]==color[2]){
                         if(y1<image->height-1)
                              y1++;
```

```
                        else
                             return;

                        row = image->row_pointers[y1];
                        ptr = &(row[x0*3]);
                        x_h = x0;

     while(ptr[0]==color[0]&&ptr[1]==color[1]&&ptr[2]==color[2]){
                             x_h++;//!
                             ptr = &(row[x_h*3]);
                        }

                        x_h = x0;
                        ptr = &(row[x_h*3]);
                   }

                   a = x0-1;
                   b = x1;
                   c = y0-1;
                   d = y1;
                   if(count==1)
                        list = new_element(a,b,c,d);
                   else
                        push(a,b,c,d, list);

                   row = image->row_pointers[y];
                   ptr = &(row[x*3]);
                        }
              }
        }
     struct ListXY* cur = list;
     if(a!=-1){
          while(cur->next!=NULL){
               rectangle_in(image,   cur->a,   cur->c,   cur->b,   cur->d,
border_color, thickness);
               cur = cur->next;
          }
          rectangle_in(image,    cur->a,    cur->c,    cur->b,    cur->d,
border_color, thickness);
     }
}
void  setPixel(png_byte*  row,  int  x,  png_byte  red,  png_byte  green,
png_byte blue){
        png_byte* ptr = &(row[x*3]);
        ptr[0] = red;
        ptr[1] = green;
        ptr[2] = blue;
}
void print_help(){
     printf("Course   work   for   option   4.22,   created   by   Veronika
Antipina.\n");
     printf("Здравствуйте! Вы можете использовать следующие флаги:\n");
     printf("--color_replace (параметры --old_color, --new_color, после
каждого цвет в формате rrr.ggg.bbb, где rrr, ggg, bbb - число в диапозоне
от 0 до 255), чтобы заменить один выбранный цвет на другой.\n");
```

```
    printf("--ornament (параметры --pattern (circle, rectangle, semicircles), --color, --thickness - цвет и толщина узора, для всех типов, кроме circle, --count. Рисует красивую рамку:)\n");
    printf("--filled_rects (параметры --color, --border_color, --thickness) - обводит все прямоугольники заданного цвета.\n");
    printf("--help, -h - информация о флагах.\n");
    printf("--output, -o, --input, -i. Изменение названий файлов для записи результата и подаваемых на вход.\n");
}

void create(struct Png* image, char* filename, int width, int height){

        FILE* fp = fopen(filename,"wb");
        if(!fp){
                printf("\n");
                exit(42);
        }

        image->png_ptr = png_create_write_struct(PNG_LIBPNG_VER_STRING, NULL, NULL, NULL);
        if(!image->png_ptr)
                exit(42);

        image->info_ptr = png_create_info_struct(image->png_ptr);
        png_init_io(image->png_ptr, fp);

        png_set_IHDR(image->png_ptr, image->info_ptr, width, height, 8, PNG_COLOR_TYPE_RGB, PNG_INTERLACE_NONE, PNG_COMPRESSION_TYPE_DEFAULT, PNG_FILTER_TYPE_DEFAULT);

        png_write_info(image->png_ptr, image->info_ptr);
        png_bytep row = (png_bytep)malloc(3*width*sizeof(png_byte));
    if(!row)
      exit(45);

        for(int y=0; y<height; y++){
                for(int x=0; x<width; x++){
                        setPixel(row,x,255,0,0);
                }
                png_write_row(image->png_ptr, row);
        }

        png_write_end(image->png_ptr, NULL);
        png_destroy_write_struct(&(image->png_ptr), &(image->info_ptr));

        fclose(fp);
        free(row);

}

void create_file(char*filename, struct Png* image, int width, int height){
        create(image, filename, width, height);
        read_png_file(filename, image);
}
```

```
void blur(struct Png* image,struct Png* image_new, int size, char*
filename){
     int count = size*size;
     int x_st = 0;
     int x_end = 0;
     int y_st = 0;
     int y_end = 0;
     int co = size/2;
     int sum_r = 0;
     int sum_b = 0;
     int sum_g = 0;
     int dc = 0;
     create_file(filename, image_new, image->width, image->height);

     for(int y=0; y<image->height; y++){
          png_byte* row = image_new->row_pointers[y];
          y_st = y-co;
          y_end = y+co;
          if(y_st<0)
               y_st = 0;
          if(y_end>=image->height)
               y_end = image->height;
          for(int x = 0; x<image->width; x++){
               x_st = x-co;
               x_end = x+co;
               if(x_st<0)
                    x_st = 0;
               if(x_end>=image->width)
                    x_end = image->width;
               count = (x_end-x_st)*(y_end-y_st);
               for(int j = y_st; j<y_end; j++){
                    png_byte* row_cur = image->row_pointers[j];
                    for(int i = x_st; i<x_end; i++){
                         png_byte* cur = &(row_cur[i*3]);
                         sum_r += cur[0];
                         sum_g += cur[1];
                         sum_b += cur[2];
                    }
               }
               png_byte* ptr = &(row[x*3]);
               dc = count/2;
               if(sum_r%count >= dc)
                    ptr[0] = sum_r/count+1;
               else
                    ptr[0] = sum_r/count;
               if(sum_g%count >= dc)
                    ptr[1] = sum_g/count+1;
               else
                    ptr[1] = sum_g/count;
               if(sum_b%count >= dc)
                    ptr[2] = sum_b/count+1;
               else
                    ptr[2] = sum_b/count;
               sum_r = 0;
               sum_g = 0;
               sum_b = 0;
          }
```

```
        }
        for(int y=0; y<image->height; y++){
            png_byte* cur_row = image->row_pointers[y];
            png_byte* row = image_new->row_pointers[y];
            for(int x = 0; x<image->width; x++){
                png_byte* cur = &(cur_row[x*3]);
                png_byte* ptr = &(row[x*3]);
                cur[0] = ptr[0];
                cur[1] = ptr[1];
                cur[2] = ptr[2];
            }

        }
    }

    void contrast(struct Png* image, float a, int b){
        int f = 0;
        int s = 0;
        int t = 0;
        for(int y=0; y<image->height; y++){
            png_byte* row = image->row_pointers[y];
            for(int x = 0; x<image->width; x++){
                png_byte* ptr = &(row[x*3]);
                f =(int)(ptr[0]*a+b);
                s = (int)(ptr[1]*a +b);
                t = (int)(ptr[2]*a+b);
                if(f<=255)
                    ptr[0] = f;
                if(s<=255)
                    ptr[1] = s;
                if(t<=255)
                    ptr[2] = t;
            }
        }
    }

    int main(int argc, char** argv){
        struct Png image;

        const char* short_options = "CO:N:up:c:t:n:fb:ahi:o:sz:xl:B:";

        const struct option long_options[] = {
            { "color_replace", no_argument, NULL, 'C'},
            { "old_color", required_argument, NULL, 'O'},
            { "new_color", required_argument, NULL, 'N'},
            { "ornament", no_argument, NULL, 'u'},
            { "pattern", required_argument, NULL, 'p'},
            { "color", required_argument, NULL, 'c'},
            { "thickness", required_argument, NULL, 't'},
            { "count", required_argument, NULL, 'n'},
            { "filled_rects", no_argument, NULL, 'f'},
            { "border_color", required_argument, NULL, 'b'},
            { "info", no_argument, NULL, 'a'},
            { "help", no_argument, NULL, 'h'},
            { "input", required_argument, NULL, 'i'},
            { "output", required_argument, NULL, 'o'},
            { "blur", no_argument, NULL, 's'},
```

```
            { "size", required_argument, NULL, 'z'},
            { "contrast", no_argument, NULL, 'x'},
            { "alpha", required_argument, NULL, 'l'},
            { "beta", required_argument, NULL, 'B'},
            { NULL, 0, NULL, 0}
        };

        int rez;
        int option_index = -1;
        char* new_filename = "res.png";
        char* old_filename = argv[argc-1];
        int option = 0;
        char* old_color = "rrr.ggg.bbb";
        char* new_color = "rrr.ggg.bbb";
        char* pattern = "shape";
        char* color = "rrr.ggg.bbb";
        int thickness = 0;
        int count = 0;
        int count_control = 0;
        int color_control = 0;
        int is_read = 0;
        int  thickness_control  =  0;//если вызывают несколько функций с
одинаковыми параметрами, значение не должно перезаписываться
        char* border_color = "rrr.ggg.bbb";
        int size = 0;
        float alpha = 0.0;
        int beta = 0;

        while((rez  =  getopt_long(argc,  argv,  short_options,  long_options,
&option_index))!=-1){
             switch(rez){
                  case 'i':
                       old_filename = optarg;
                       break;
                  case 'h':
                       print_help();
                       return(0);
                       break;
                  case 'o':
                       new_filename = optarg;
                       break;
                  case 'a':
                       if(argc<=2)
                            exit(42);
                       printf("Image  filename:  %s,  image  width:  %d,  image
height: %d\n",old_filename, image.width, image.height);
                       return(0);
                       break;
                  case 'C':
                       if(!option)
                            option = 1;
                       break;
                  case 'O':
                       old_color = optarg;
                       break;
                  case 'N':
                       new_color = optarg;
```

```
                    break;
               case 'u':
                    if(!option)
                         option = 2;
                    break;
               case 'p':
                    if(!strcmp(pattern,"shape"))
                         pattern = optarg;
                    break;
               case 'c':
                    if(!color_control)
                         color = optarg;
                    color_control = 1;
                    break;
               case 't':
                    if(!thickness_control)
                         thickness = atoi(optarg);
                    thickness_control = 1;
                    break;
               case 'n':
                    if(!count_control)
                         count = atoi(optarg);
                    count_control = 1;
                    break;
               case 'f':
                    if(!option)
                         option = 3;
                    break;
               case 'b':
                    border_color = optarg;
                    break;
               case 's':
                    if(!option)
                         option = 4;
                    break;
               case 'z':
                    size = atoi(optarg);
                    break;
               case 'x':
                    if(!option)
                         option = 5;
                    break;
               case 'l':
                    alpha = atof(optarg);

                    break;
               case 'B':
                    beta = atoi(optarg);
                    break;
               case '?': default:
     //             printf("Found unknown option! Type --help or -h to
check.\n");
                    break;
          };
     }
     if(!strcmp(old_filename, new_filename))
          return(42);
```

```
    read_png_file(old_filename, &image);
    int* str_1;
    int* str_2;
    struct Png image_new;

    switch(option){
        case 1:
            str_1 = find_colors(old_color);
            str_2 = find_colors(new_color);
            color_replace(&image, str_1, str_2);
            free(str_1);
            free(str_2);
            break;
        case 2:
                    str_2 = find_colors(color);
            ornament(&image, str_2, pattern, thickness, count);
            free(str_2);
            break;
        case 3:
            str_1 = find_colors(color);
            str_2 = find_colors(border_color);
            if(thickness<=0)
                return(42);
            filled_rects(&image, str_1, str_2, thickness);
            free(str_1);
            free(str_2);
            break;
        case 4:
            if(size<=0)
                return(42);
            if(size%2==0)
                size++;
            blur(&image, &image_new, size, new_filename);
            break;
        case 5:
            if(alpha<=0)
                return(42);
            contrast(&image, alpha, beta);
            break;
        default:
            printf("You didn't choose any functions! Type -h or --
help.\n");
            return(42);
            break;
    };

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

Имя файла: Makefile

```
all: main.o
    gcc main.o -o ./cw -std=c99 -lpng -Wall -Werror
main.o: main.c
    gcc -c main.c -std=c99
```