

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

**КУРСОВАЯ РАБОТА**  
**по дисциплине «Программирование»**  
**Тема: Обработка PNG файлов**

Студент гр. 0383

Бояркин Н.А.

Преподаватель

Шевская Н.В.

Санкт-Петербург

2021

## ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

Студент Бояркин Н.А.

Группа 0383

Тема работы: обработка PNG файлов

Исходные данные:

Вариант 23.

Программа должна реализовывать весь следующий функционал по обработке png-файла

Общие сведения

- Формат картинки PNG (рекомендуем использовать библиотеку libpng)
- без сжатия
- файл всегда соответствует формату PNG
- обратите внимание на выравнивание; мусорные данные, если их необходимо дописать в файл для выравнивания, должны быть нулями.
- все поля стандартных PNG заголовков в выходном файле должны иметь те же значения что и во входном (разумеется кроме тех, которые должны быть изменены).

Программа должна реализовывать следующий функционал по обработке PNG-файла

(1) Рисование окружности. Окружность определяется:  
либо координатами левого верхнего и правого нижнего угла квадрата, в который она вписана, либо координатами ее центра и радиусом  
толщиной линии окружности  
цветом линии окружности

окружность может быть залитой или нет

цветом которым залита сама окружность, если пользователем выбрана залитая окружность

(2) Фильтр rgb-компонент. Этот инструмент должен позволять для всего изображения либо установить в 0 либо установить в 255 значение заданной компоненты. Функционал определяется

Какую компоненту требуется изменить

В какой значение ее требуется изменить

(3) Разделяет изображение на  $N \times M$  частей. Реализация: либо провести линии заданной толщины, тем самым разделив изображение либо сохранение каждой части в отдельный файл. -- по желанию студента (можно и оба варианта).

Функционал определяется:

Количество частей по "оси" Y

Количество частей по "оси" X

Толщина линии

Цвет линии

Либо путь куда сохранить кусочки

Содержание пояснительной записки:

«Содержание», «Введение», «Цель и задачи», «Ход выполнения работы», «Заключение», «Список использованных источников»

Предполагаемый объем пояснительной записки:

Не менее 28 страниц.

Дата выдачи задания: 05.04.2021

Дата сдачи реферата: 28.05.2021

Дата защиты реферата: 31.05.2021

Студент		Бояркин Н.А.
Преподаватель		Шевская Н.В.

## АННОТАЦИЯ

В курсовой работе реализована обработка PNG файлов с использованием библиотеки `libpng`. Реализованные опции обработки файла соответствуют заданиям.

Использовалось несколько стандартных библиотек: `stdio.h`, `stdlib.h`, `getopt.h`, `string.h`, `math.h`, `unistd.h`, `stdarg.h`. Программа реализована в виде утилиты, подобной стандартным linux утилитам: управление осуществляется посредством аргументов командной строки.

Инструкция компиляции и запуску приложения: скомпилировать файл `main.c` из папки, в которой он находится, командой “`gcc main.c -lpng -lm`”. Использовать ключи, описанные в справке (которую можно получить, вызвав утилиту без аргументов или ключом `-h` (`--help`), `-?`).

Примеры работы программы см. в приложении Б, разработанный программный код см. в приложении А.

## СОДЕРЖАНИЕ

Введение	4
1. Цель и задачи	8
1.1. Цель	8
1.2. Основные задачи	8
2. Ход выполнения работы	9
2.1. Чтение PNG-файла	9
2.2. Запись информации в выходной файл	9
2.3. Фильтр RGB-компонент	9
2.4. Рисование окружности	9
2.5. Разделение изображения на $N \times M$ частей	10
2.6. Функция печати подробной информации о PNG-файле	10
2.7. Функция main	10
Заключение	12
Список использованных источников	13
Приложение А. Название приложения	14
Приложение Б. Пример работы приложения	24

## ВВЕДЕНИЕ

Цель работы: написать стабильно работающую программу для обработки PNG файлов.

Программа разработана на операционной системе Linux Ubuntu в среде разработки CLion. Также был разработан консольный интерфейс с помощью библиотеки getopt.

В результате была создана программа, выполняющие команды пользователя: рисование окружности, фильтр RGB-компонент, разделение изображения на  $N \times M$  частей.

# **1. ЦЕЛЬ И ЗАДАЧИ**

## **1.1. Цель**

Целью курсовой работы была реализация программы, обрабатывающую введенным пользователем PNG-файл в соответствии с заданием

## **1.2. Основные задачи**

- Реализация CLI;
- Чтение PNG-файла;
- Запись PNG-файла;
- Обработка ошибок;
- Рисование окружности;
- Фильтр RGB-компонент;
- Разделение изображения на  $N * M$  частей;



## 2. ХОД ВЫПОЛНЕНИЯ РАБОТЫ

### 2.1. Чтение PNG-файла

Для хранения данных PNG-файла была создана структура `Png`. Для чтения PNG-файла была реализована функция `int read_png_file(char *file_name, struct Png *image)`, которая принимает на вход два аргумента - имя файла и указатель на экземпляр структуры `Png`. Функция возвращает 0 если успешно завершается и 1 в случае ошибки.

### 2.2. Запись информации в выходной файл.

Для записи данных в выходной PNG-файл была реализована функция `void write_png_file(char *file_name, struct Png *image)`, которая принимает на вход два аргумента - имя файла и указатель на экземпляр структуры `Png`. Функция ничего не возвращает.

### 2.3. Фильтр RGB-компонент.

Функция `void process_file(struct Png *image, char* colour, int value)` для всего изображения позволяет либо установить в 0 либо установить в 255 значение заданной компоненты. На вход функция принимает три аргумента - структуру `Png`, цвет, значение (только 0 или 255). Функция проходит по всем пикселям и изменяет каждый пиксель в значение `value`. Функция ничего не возвращает.

### 2.4. Рисование окружности.

Для рисования окружности используется функция `void drawcircle(struct Png* image, int xc, int yc, int outer, int thickness, char* colour)`. На вход функция принимает несколько аргументов - указатель на экземпляр структуры `Png`, координаты центра окружности, радиус, толщина (по умолчанию 1), цвет. В случае если нужно закрасить окружность, то толщина окружности равна радиусу + 1. Для рисования окружности используется алгоритм средней точки.

Также в этой функции используются две функции: *void xLine(struct Png\* image, int x1, int x2, int y, char\* colour)* и *void yLine(struct Png\* image, int x, int y1, int y2, char\* colour)*, которые рисуют линии по оси X и Y соответственно. Также в этих функциях используется функция *void drawpixel(struct Png \*image, int x1, int y1, char\* colour)*, которая рисует пиксель в цвет, в зависимости от значения colour.

Все подзадачи были реализованы. Пользователь может выбрать толщину и цвет линии окружности, залить окружность, цвет, которым должна быть залита окружность.

## **2.5. Разделение изображения на N\*M частей.**

Для разделения изображения на части была выбрана реализация с использованием проведения линий заданной толщины (по умолчанию толщина равна одному пикселю). Для проведения линий заданной толщины используется функция *void plotLineWidth(struct Png\* image, int x0, int y0, int x1, int y1, float wd, char \*colour)*, которая принимает в качестве аргументов указатель на экземпляр структуры Png, координаты начала и конца линии, толщину и цвет линий. Для рисовании линий используется Алгоритм Брезенхэма.

## **2.6. Функция печати подробной информации о PNG-файле.**

Для печати информации о PNG-файле была написана функция *void print\_info\_image(struct Png\* image)*, которая принимает на вход указатель на структуру Png. Функция вызывается, если передать ключ -A или --information при вызове программы.

## **2.7. Функция main.**

В функции main был создан экземпляр структуры Configs, в который были переданы аргументы по умолчанию:

- цвет - чёрный;
- входной файл - NULL;

- файл на выход - NULL;
- значение для функции RGB-фильтра - 0;
- переменная для выбора опции функции - 0;
- флаг заливки - 0;
- радиус - 200;
- координаты центра окружности - 200, 200;
- толщина - 1;
- количество частей по “оси” Y - 3;
- количество частей по “оси” X - 5;

Дальше обрабатывается оператор switch и функция `getopt_long()`, которая изменяет значения экземпляра структуры `config` в зависимости от вызова ключа и аргументов пользователя.

Далее в зависимости от значения поля `config.action` обрабатывается изображение.

## **ЗАКЛЮЧЕНИЕ**

В результате выполнения данной работы была на практике применена библиотека `libpng` для работы с PNG-файлами и библиотека `getopt` для построение консольного интерфейса. Была создана стабильная программа, которая обрабатывает изображение в зависимости от команды пользователя. Также произведена обработка всех возможных исключительных ситуаций.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Сайт-справочник по Си, <http://cplusplus.com/>
2. Информация по libpng и самому формату: <http://www.libpng.org/>
3. Информация по алгоритмам Брезенхэма: [The Beauty of Bresenham's Algorithm](#)
4. Stack Overflow — система вопросов и ответов о программировании: <https://stackoverflow.com/>

## ПРИЛОЖЕНИЕ А

### MAIN.C

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

struct Configs{
    char* colour;
    char* file_in;
    char* file_out;
    int value;
    int action;
    int flag;
    int radius;
    int xc, yc;
    int thickness;
    int partsX, partsY;
};

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

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

int read_png_file(char *file_name, struct Png *image) {
    int x,y;
    char header[8];    // 8 is the maximum size that can be checked

    /* open file and test for it being a png */
    FILE *fp = fopen(file_name, "rb");
    if (!fp){
        // Some error handling: file could not be opened
        fprintf(stderr, "file could not be opened\n");
        return 1;
    }

    fread(header, 1, 8, fp);
    if (png_sig_cmp(header, 0, 8)){
        // Some error handling: file is not recognized as a PNG
        fprintf(stderr, "file is not recognized as a PNG\n");
        return 1;
    }
}
```

```

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

if (!image->png_ptr){
    // Some error handling: png_create_read_struct failed
    fprintf(stderr, "png_create_read_struct failed\n");
    return 1;
}

image->info_ptr = png_create_info_struct(image->png_ptr);
if (!image->info_ptr){
    // Some error handling: png_create_info_struct failed
    fprintf(stderr, "png_create_info_struct failed\n");
    return 1;
}

if (setjmp(png_jmpbuf(image->png_ptr))){
    // Some error handling: error during init_io
    fprintf(stderr, "error during init_io\n");
    return 1;
}

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

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

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

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

/* read file */
if (setjmp(png_jmpbuf(image->png_ptr))){
    // Some error handling: error during read_image
    fprintf(stderr, "error during read_image\n");
    return 1;
}

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

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

```

```

        fclose(fp);
        return 0;
    }

void write_png_file(char *file_name, struct Png *image) {
    int x,y;
    /* create file */
    FILE *fp = fopen(file_name, "wb");
    if (!fp){
        // Some error handling: file could not be opened
        fprintf(stderr, "file could not be opened\n");
        return;
    }

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

    if (!image->png_ptr){
        // Some error handling: png_create_write_struct failed
        fprintf(stderr, "png_create_write_struct failed\n");
        return;
    }

    image->info_ptr = png_create_info_struct(image->png_ptr);
    if (!image->info_ptr){
        // Some error handling: png_create_info_struct failed
        fprintf(stderr, "png_create_info_struct failed\n");
        return;
    }

    if (setjmp(png_jmpbuf(image->png_ptr))){
        // Some error handling: error during init_io
        fprintf(stderr, "error during init_io\n");
        return;
    }

    png_init_io(image->png_ptr, fp);

    /* write header */
    if (setjmp(png_jmpbuf(image->png_ptr))){
        // Some error handling: error during writing header
        fprintf(stderr, "error during writing header\n");
        return;
    }

    png_set_IHDR(image->png_ptr, image->info_ptr, image->width,
    image->height,
                                image->bit_depth, image->color_type,
    PNG_INTERLACE_NONE,
                                PNG_COMPRESSION_TYPE_BASE, PNG_FILTER_TYPE_BASE);

    png_write_info(image->png_ptr, image->info_ptr);
}

```



```

/* write bytes */
if (setjmp(png_jmpbuf(image->png_ptr))) {
    // Some error handling: error during writing bytes
    fprintf(stderr, "error during writing bytes\n");
    return;
}

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

/* end write */
if (setjmp(png_jmpbuf(image->png_ptr))) {
    // Some error handling: error during end of write
    fprintf(stderr, "error during end of write\n");
    return;
}

png_write_end(image->png_ptr, NULL);

/* cleanup heap allocation */
for (y = 0; y < image->height; y++)
    free(image->row_pointers[y]);
free(image->row_pointers);

fclose(fp);
}

void setPixelColor(struct Png *image, int x1, int y1, int colour) {
    if (x1 < 0 || y1 < 0 || x1 >= image->width || y1 >= image->height)
        return;
    png_byte *row = image->row_pointers[y1];
    png_byte *ptr = &(row[x1 * 4]);
    if (png_get_color_type(image->png_ptr, image->info_ptr) ==
PNG_COLOR_TYPE_RGBA) {
        ptr[3] = 255;
    }
    ptr[0] = colour;
    ptr[1] = colour;
    ptr[2] = colour;
}

void drawpixel(struct Png *image, int x1, int y1, char* colour) {
    if (x1 < 0 || y1 < 0 || x1 >= image->width || y1 >= image->height)
        return;
    png_byte *row = image->row_pointers[y1];
    png_byte *ptr = &(row[x1 * 4]);
    if (png_get_color_type(image->png_ptr, image->info_ptr) ==
PNG_COLOR_TYPE_RGBA) {
        ptr[3] = 255;
    }
    if (!strcmp("black", colour)) {
        ptr[0] = 0;
        ptr[1] = 0;
        ptr[2] = 0;
    } else if (!strcmp("white", colour)) {

```

```

        ptr[0] = 255;
        ptr[1] = 255;
        ptr[2] = 255;
    } else if (!strcmp("red", colour)){
        ptr[0] = 255;
        ptr[1] = 0;
        ptr[2] = 0;
    } else if (!strcmp("green", colour)){
        ptr[0] = 0;
        ptr[1] = 255;
        ptr[2] = 0;
    } else if (!strcmp("blue", colour)){
        ptr[0] = 0;
        ptr[1] = 0;
        ptr[2] = 255;
    }
}

```

```

void process_file(struct Png *image, char* colour, int value) {
    int x,y;
    if (png_get_color_type(image->png_ptr, image->info_ptr) ==
    PNG_COLOR_TYPE_RGB){
        // Some error handling: input file is PNG_COLOR_TYPE_RGB but
        must be PNG_COLOR_TYPE_RGBA
        fprintf(stderr, "input file is PNG_COLOR_TYPE_RGB but must be
        PNG_COLOR_TYPE_RGBA\n");
        return;
    }

```

```

        if (png_get_color_type(image->png_ptr, image->info_ptr) !=
        PNG_COLOR_TYPE_RGBA){
            // Some error handling: color_type of input file must be
            PNG_COLOR_TYPE_RGBA
            fprintf(stderr, "color_type of input file must be
            PNG_COLOR_TYPE_RGBA\n");
            return;
        }

```

```

    for (y = 0; y < image->height; y++) {
        png_byte *row = image->row_pointers[y];
        for (x = 0; x < image->width; x++) {
            png_byte *ptr = &(row[x * 4]);
            if(!strcmp("red", colour)){
                ptr[0] = value;
            } else if (!strcmp("green", colour)){
                ptr[1] = value;
            } else if (!strcmp("blue", colour)){
                ptr[2] = value;
            }
        }
    }
}

```

```

void xLine(struct Png* image, int x1, int x2, int y, char* colour)
{
    while (x1 <= x2){

```

```

        drawpixel(image, x1++, y, colour);
    }
}

void yLine(struct Png* image, int x, int y1, int y2, char* colour)
{
    while (y1 <= y2){
        drawpixel(image, x, y1++, colour);
    }
}

void drawcircle(struct Png* image, int xc, int yc, int outer, int
thickness, char* colour)
{
    int inner = outer - thickness + 1;
    int xo = outer;
    int xi = inner;
    int y = 0;
    int erro = 1 - xo;
    int erri = 1 - xi;

    while(xo >= y) {
        xLine(image, xc + xi, xc + xo, yc + y, colour);
        yLine(image, xc + y, yc + xi, yc + xo, colour);
        xLine(image, xc - xo, xc - xi, yc + y, colour);
        yLine(image, xc - y, yc + xi, yc + xo, colour);
        xLine(image, xc - xo, xc - xi, yc - y, colour);
        yLine(image, xc - y, yc - xo, yc - xi, colour);
        xLine(image, xc + xi, xc + xo, yc - y, colour);
        yLine(image, xc + y, yc - xo, yc - xi, colour);

        y++;

        if (erro < 0) {
            erro += 2 * y + 1;
        } else {
            xo--;
            erro += 2 * (y - xo + 1);
        }

        if (y > inner) {
            xi = y;
        } else {
            if (erri < 0) {
                erri += 2 * y + 1;
            } else {
                xi--;
                erri += 2 * (y - xi + 1);
            }
        }
    }
}

int max(int a, int b){
    if (a >= b)
        return a;
}

```

```

        return b;
    }

void plotLineWidth(struct Png* image, int x0, int y0, int x1, int y1,
float wd, char *colour){
    int dx = abs(x1-x0), sx = x0 < x1 ? 1 : -1;
    int dy = abs(y1-y0), sy = y0 < y1 ? 1 : -1;
    int err = dx-dy, e2, x2, y2; /* error
value e_xy */
    float ed = dx+dy == 0 ? 1 : sqrt((float)dx*dx+(float)dy*dy);

    for (wd = (wd+1)/2; ; ) { /*
pixel loop */
        drawpixel(image, x0, y0, colour);
        e2 = err; x2 = x0;
        if (2*e2 >= -dx) { /*
x step */
            for (e2 += dy, y2 = y0; e2 < ed*wd && (y1 != y2 || dx >
dy); e2 += dx)
                drawpixel(image, x0, y2 += sy, colour);
            if (x0 == x1) break;
            e2 = err; err -= dy; x0 += sx;
        }
        if (2*e2 <= dy) { /*
y step */
            for (e2 = dx-e2; e2 < ed*wd && (x1 != x2 || dx < dy); e2 +=
dy)
                drawpixel(image, x2 += sx, y0, colour);
            if (y0 == y1) break;
            err += dx; y0 += sy;
        }
    }
}

void print_info_image(struct Png* image){
    printf("Width: %d and height: %d\n", image->width, image->height);
    printf("Depth: %d\n", image->bit_depth);
    if (png_get_color_type(image->png_ptr, image->info_ptr) ==
PNG_COLOR_TYPE_RGB){
        printf("input file is PNG_COLOR_TYPE_RGB\n");
        return;
    }
    if (png_get_color_type(image->png_ptr, image->info_ptr) ==
PNG_COLOR_TYPE_RGBA){
        printf("color_type of input file is PNG_COLOR_TYPE_RGBA\n");
        return;
    }
}

void printHelp(){
    printf("main.c: picture must be a png file and RGBA.\n");
    printf("-h -? --help - help\n");
    printf("-A --information - the information about png file\n");
}

```

```

printf("-d --drawcircle - draw a circle\n");
printf("-f --filter - RGBA filter\n");
printf("-p --parts - Divides an image into N * M parts.\n");
printf("-c --colour - colour (black, white, red, green, blue)\n");
printf("-v --value - value (only 0 or 255)\n");
printf("-F --Fill - fills the circle\n");
printf("-r --radius - radius\n");
printf("-t --thickness - thickness\n");
printf("-C --coordinates - coordinates of the circle (X,Y)\n");
printf("-P --Parts - N and M parts (N,M)\n");
printf("-i --image - file for input\n");
printf("-I --Image - file for output\n");
printf("default values:\n");
printf("colour: black; input file: NULL; output file: NULL;\n");
printf("value: 0; action: 0; flag(filling circle): 0;\n");
printf("radius: 200; Coordinates of circle (200,200)\n");
printf("thickness: 1; partsX: 3; parts: 5\n");
}

```

```

int main(int argc, char **argv) {
    if (argc == 1){
        printHelp();
        return 0;
    }
    struct Png image;
    struct Configs config = {"black", NULL, NULL,
                             0, 0, 0,
                             200, 200, 200,
                             1, 3, 5};
    char* opts = "h?fdpAFP:C:c:v:i:I:r:t:";
    struct option longOpts[] = {
        {"help", no_argument, NULL, 'h'},
        {"information", no_argument, NULL, 'A'},
        {"filter", no_argument, NULL, 'f'},
        {"drawcircle", no_argument, NULL, 'd'},
        {"parts", no_argument, NULL, 'p'},
        {"Fill", no_argument, NULL, 'F'},
        {"colour", required_argument, NULL, 'c'},
        {"value", required_argument, NULL, 'v'},
        {"image", required_argument, NULL, 'i'},
        {"Image", required_argument, NULL, 'I'},
        {"radius", required_argument, NULL, 'r'},
        {"thickness", required_argument, NULL, 't'},
        {"coordinates", required_argument, NULL, 'C'},
        {"Parts", required_argument, NULL, 'P'},
        {NULL, 0, NULL, 0}
    };
    int flag_info = 0;
    int opt;
    int longIndex = 0;
    opt = getopt_long(argc, argv, opts, longOpts, &longIndex);
    while (opt != -1){
        switch (opt) {
            case 'h':
            case '?':
                printHelp();

```

```

        return 0;
    case 'A':
        flag_info = 1;
        break;
    case 'd':
        config.action = 1;
        break;
    case 'f':
        config.action = 2;
        break;
    case 'p':
        config.action = 3;
        break;
    case 'c':
        config.colour = optarg;
        break;
    case 'v':
        config.value = atoi(optarg);
        break;
    case 'F':
        config.flag = 1;
        break;
    case 'i':
        config.file_in = optarg;
        break;
    case 'I':
        config.file_out = optarg;
        break;
    case 'r':
        config.radius = atoi(optarg);
        break;
    case 't':
        config.thickness = atoi(optarg);
        break;
    case 'C':
        sscanf(optarg, "%d,%d", &config.xc, &config.yc);
        break;
    case 'P':
        sscanf(optarg, "%d,%d", &config.partsX,
&config.partsY);
        break;
    case 0:
        printf("->%s\n", longOpts[longIndex].name);
    }
    opt = getopt_long(argc, argv, opts, longOpts, &longIndex);
}
argc -= optind;
argv += optind;
for(int i=0; i<argc; i++)
    printf(">%s\n", argv[i]);
float wd = (float)config.thickness;
int res;
// ДОПОЛНИТЕЛЬНЫЕ ПРОВЕРКИ (условие if)
if (config.file_in != NULL && config.file_out != NULL){
    res = read_png_file(config.file_in, &image);
    if (config.action == 2 && res == 0 && (config.value == 0 ||
config.value == 255)) {

```

```

        // res = read_png_file(config.file_in, &image);
        process_file(&image, config.colour, config.value);
        write_png_file(config.file_out, &image);
    } else if (config.action == 1 && res == 0) {
        // res = read_png_file(config.file_in, &image);
        if (config.flag == 1) {
            config.thickness = config.radius + 1;
            drawcircle(&image, config.xc, config.yc, config.radius,
config.thickness, config.colour);
        } else {
            drawcircle(&image, config.xc, config.yc, config.radius,
config.thickness, config.colour);
        }
        write_png_file(config.file_out, &image);
    } else if (config.action == 3 && res == 0) {
        // res = read_png_file(config.file_in, &image);

        // plotLineWidth(&image, 0, 0, 0, image.height, wd,
config.colour);
        for (int i = image.width / config.partsX; i < image.width;
i += image.width / config.partsX) {
            if (image.width % config.partsX == 0){
                plotLineWidth(&image, i, 0, i, image.height, wd,
config.colour);
            } else{
                if (i + image.width / config.partsX < image.width)
                    plotLineWidth(&image, i, 0, i, image.height,
wd, config.colour);
            }
        }

        plotLineWidth(&image, image.width, 0, image.width,
image.height, wd, config.colour);
        // plotLineWidth(&image, 0, 0, image.width, 0, wd,
config.colour);

        for (int i = image.height / config.partsY; i <
image.height; i += image.height / config.partsY) {
            if (image.width % config.partsY == 0){
                plotLineWidth(&image, 0, i, image.width, i, wd,
config.colour);
            } else{
                if (i + image.height / config.partsY < image.width)
                    plotLineWidth(&image, 0, i, image.width, i, wd,
config.colour);
            }
        }

        plotLineWidth(&image, 0, image.height, image.width,
image.height, wd, config.colour);
        write_png_file(config.file_out, &image);
    }
    } else{
        fprintf(stderr, "Usage: main.c <file_in> <file_out>\n");
    }
    if (config.file_in != NULL && config.file_out != NULL && flag_info
== 1){
        print_info_image(&image);
    }
    return 0;

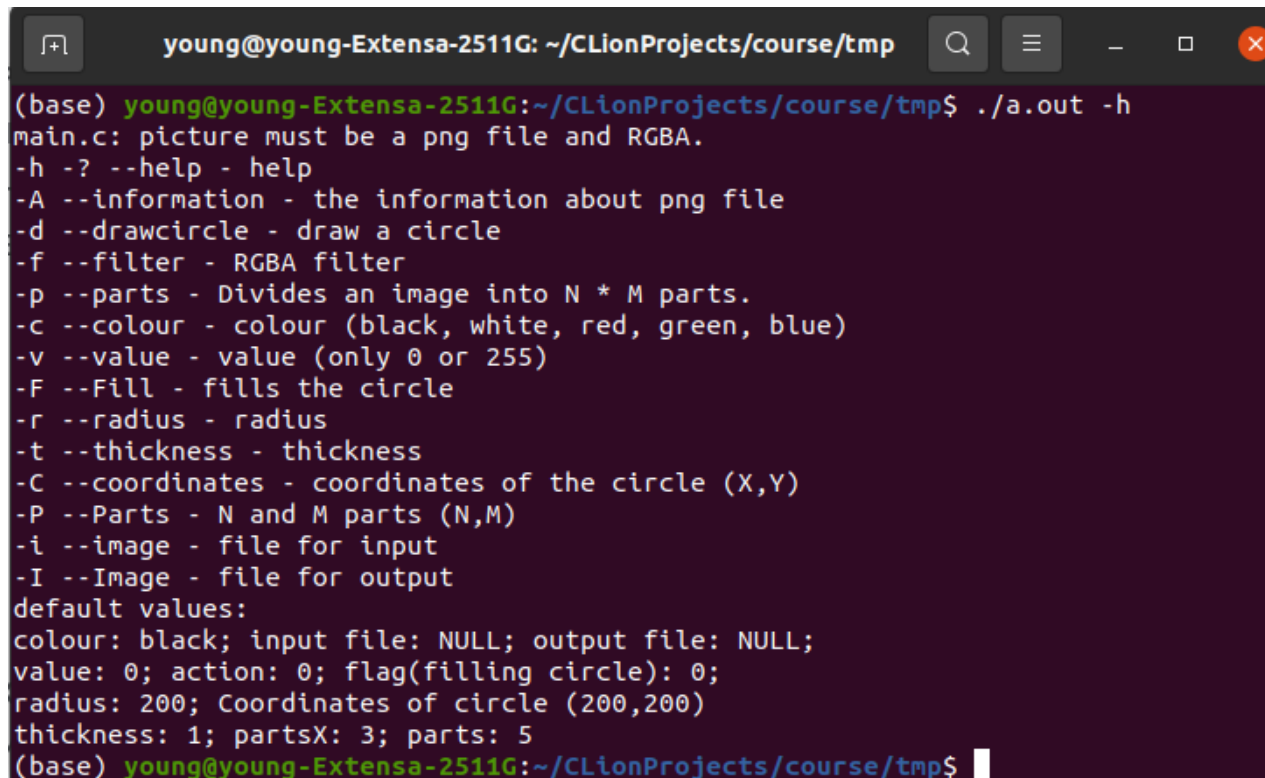
```

}

## ПРИЛОЖЕНИЕ Б

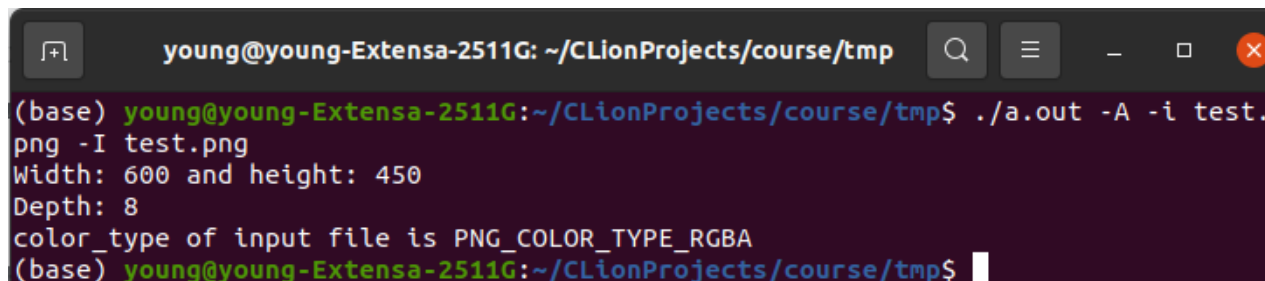
### ПРИМЕРЫ РАБОТЫ ПРОГРАММЫ MAIN.C

1. Вызов справки, которая распечатывается при вызове программы без аргументов, ключом  `-?`,  `-h` и  `--help`.



```
young@young-Extensa-2511G: ~/CLionProjects/course/tmp
(base) young@young-Extensa-2511G:~/CLionProjects/course/tmp$ ./a.out -h
main.c: picture must be a png file and RGBA.
-h -? --help - help
-A --information - the information about png file
-d --drawcircle - draw a circle
-f --filter - RGBA filter
-p --parts - Divides an image into N * M parts.
-c --colour - colour (black, white, red, green, blue)
-v --value - value (only 0 or 255)
-F --Fill - fills the circle
-r --radius - radius
-t --thickness - thickness
-C --coordinates - coordinates of the circle (X,Y)
-P --Parts - N and M parts (N,M)
-i --image - file for input
-I --Image - file for output
default values:
colour: black; input file: NULL; output file: NULL;
value: 0; action: 0; flag(filling circle): 0;
radius: 200; Coordinates of circle (200,200)
thickness: 1; partsX: 3; parts: 5
(base) young@young-Extensa-2511G:~/CLionProjects/course/tmp$
```

2. Вызов функции подробной информации о PNG-файле.



```
young@young-Extensa-2511G: ~/CLionProjects/course/tmp
(base) young@young-Extensa-2511G:~/CLionProjects/course/tmp$ ./a.out -A -i test.
png -I test.png
Width: 600 and height: 450
Depth: 8
color_type of input file is PNG_COLOR_TYPE_RGBA
(base) young@young-Extensa-2511G:~/CLionProjects/course/tmp$
```



### 3. Рисование окружности (без заливки).

```
$ ./a.out -d -C 300,225 -r 150 -t 10 -c blue -i test.png -I  
res.png
```

Рисование окружности с центром координат (300, 225), радиусом 150 и толщиной линии 10 пикселей с синим цветом. Входной файл test.png, файл на выход res.png



#### 4. Рисование окружности (с заливкой).

```
$ ./a.out -d -C 300,225 -c green -F -i test.png -I res.png
```

Рисование окружности с центром координат (300,225) с заливкой с зелёным цветом.



#### 5. Фильтр RGB-компонент (Пример №1).

```
$ ./a.out --filter -c red -v 0 -i test.png -I res.png
```

Изменение красного компонента для всего изображения в значение 0.



## 6. Фильтр RGB-компонент (Пример №2).

```
$ ./a.out --filter -c blue -v 255 -i test.png -I res.png
```

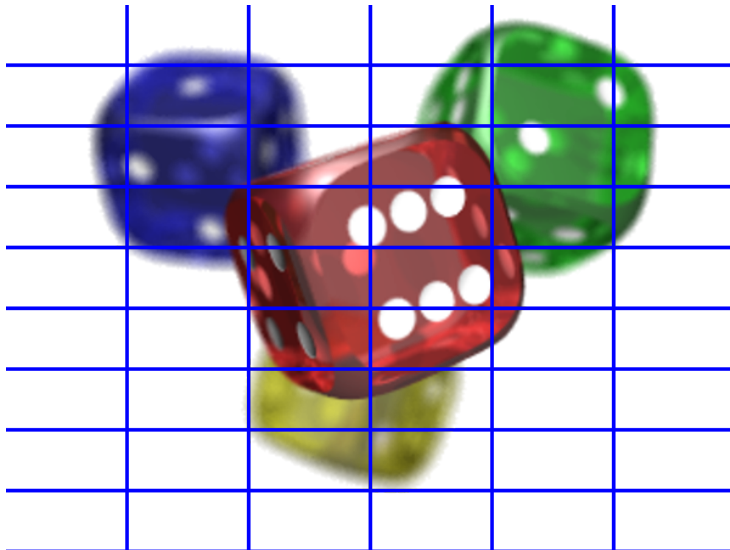
Изменение синего компонента для всего изображения в значение 255.



## 7. Разделение изображения на $N * M$ частей (Пример №1).

```
$ ./a.out -p -P 6,9 -c blue -t 5 -i test.png -I res.png
```

Разделение изображения на  $6 * 9$  частей с линиями синего цвета, толщиной в 5 пикселей.



## 8. Разделение изображения на $N * M$ частей (Пример №2).

```
$ ./a.out -p -P 7,7 -c red -t 1 -i test.png -I res.png
```

Разделение изображения на 7\*7 частей с линиями красного цвета, толщиной в 1 пиксель.

