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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Обработка изображения в формате PNG

Студент гр. 1303 Чубан Д.В.

Преподаватель Чайка К.В.

Санкт-Петербург
2022

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студент: Чубан Дмитрий

Группа 1303

Тема работы: Обработка изображения в формате **PNG**

Вариант 23

Условие задания:

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*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

Дата выдачи задания: 22.03.2022

Дата сдачи реферата: 01.03.2022

Дата защиты работы: 03.06.2022

Студент группы 1303 _____ Чубан Д.В.

Преподаватель _____ Чайка К.В.

АННОТАЦИЯ

Работа представляет собой программу, осуществляющую обработку изображений типа PNG с использованием терминального интерфейса. В программе реализованы следующие опции для обработки изображения: рисование окружности, изменение одного из RGB-каналов изображение в заданную величину, разделение изображения линиями заданной толщины на заданное количество N*M частей.

ВВЕДЕНИЕ

Целью работы являлось: научиться обрабатывать изображения в языке программирования Си, изучить и освоить передачу опций и аргументов к ним через командную строку. Написать программу, осуществляющую считывание изображения, обработку и сохранение полученного результата в новый файл.

1. ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ

1.1. Структуры

Для работы программы были написаны структуры *Png, Configs.* В структуру *Png* основная информация об изображении. *Configs* - структура, хранящая в себе флаги и аргументы, переданные с опциями.

1.2. Описание функции printInfo()

Функция выводит основную информацию об изображении (ширина, высота, битовая глубина и тип цветовой палитры).

1.3. Описание функций read_png_file() и write_png_file()

Данные функции обеспечивают считывание информации об изображении и ее обновлении после изменений, сделанных в ходе работы программы.

1.4. Описание функции drawPixel()

Функция принимает на вход ссылку на изображение Png* image, координаты пикселя, цвет которого надо изменить (int x, int y) и 3 значения для каждого канала RGB. После этого проверяется, находится ли данный пиксель в пределах изображения, и изменяются значения RGB.

1.5. Описание функции checkColor()

Функция принимает на вход 3 значения типа int, соответствующие значениям цвета, и проверяет, не выходят ли данные значения за допустимые границы (0-255).

1.6. Описание функции drawOutline()

Функция принимает на вход изображение, радиус рисуемого круга, координаты его центра, три RGB цвета, определяющие цвет линии и двумерный массив int mask, который нужен для определения заливаемых пикселей (если пользователь выбрал опцию заливки и/или толщины окружности).

Далее алгоритмом Брезенхема рисуется окружность с помощью функции drawPixel() и отмечает в mask перекрашенные пиксели.

1.7. Описание функции Fillment()

Функция принимает на вход изображение, маску заливки, координаты начала заливки и ее цвет.

Функция проверяет, находится ли текущий пиксель в границах изображения и равенство 0 соответствующему элементу из маски, после чего меняет цвет текущего пикселя на заливаемый с помощью drawPixel() и рекурсивно вызывает себя для соседних пикселей.

1.8. Описание функции strtokForTwo()

Функция принимает на вход строку и два значения, куда нужно будет записать разделенные значения.

Функция делит строку с помощью функции strtok() и, если значений в строке больше или меньше двух, уведомляет о неправильности введенных данных и завершает программу.

1.9. Описание функции strtokForThree()

Функция принимает на вход строку и три значения, куда нужно будет записать разделенные значения.

Функция делит строку с помощью функции strtok() и, если значений в строке больше или меньше трех, уведомляет о неправильности введенных данных и завершает программу.

1.10. Описание функции drawCircle()

Функция принимает на вход изображение и struct Configs cfg, откуда будут браться данные для рисования окружности.

Далее, инициализируется двумерный массив int mask, который нужен для выполнения заливки и создании толщины линии окружности. Если окружность была задана пользователем с помощью координат углов квадрата, в который вписана окружность, то функция просчитывает центр этой окружности и ее радиус.

Далее вызывается функция drawOutline() и если пользователь выбрал опцию толщины, то дополнительно вызываются еще одна drawOutline() и Fillment(). Если выбрана опция заливки, то выполняется функция Fillment().

1.11. Описание функции colorChanger()

Функция принимает на вход изображение, букву, соответствующую одному из каналов RGB и значение, на которое данный канал нужно изменить. Если значения входных данных не соответствуют правильному формату, выводится сообщение об ошибке введенных данных и программа завершается.

Функция проверяет на валидность введенные букву и значение цвета, после чего в цикле меняет значение RGB канала для каждого пикселя на изображении.

1.12. Описание функции divideImage()

Функция принимает на вход изображение и структуру конфигураций.

После чего в цикле на изображении с помощью функции drawPixel() рисуются линии-разделители заданной толщины (если она была задана).

1.13. Описание функции main()

В главной функции происходит инициализация структуры конфигураций стандартными значениями, потом вызывается функция optsProcessing(), которая заполняет структуру данными. Далее, если в программу при запуске не было передано значение входного, выводится ошибка и программа завершается. После проверки вызывается функция read_png_file() и ее результат передается в переменную image. Далее в зависимости от введенных опций вызывается одна из функций printInfo(), drawCircle(), divideImage() или colorChanger().

После выполнения одной из функций, результат преобразований записывается в новый файл с помощью функции write_png_file() либо с именем, введенным пользователем, либо со стандартным output.png.

1.14. Описание функции printHelp()

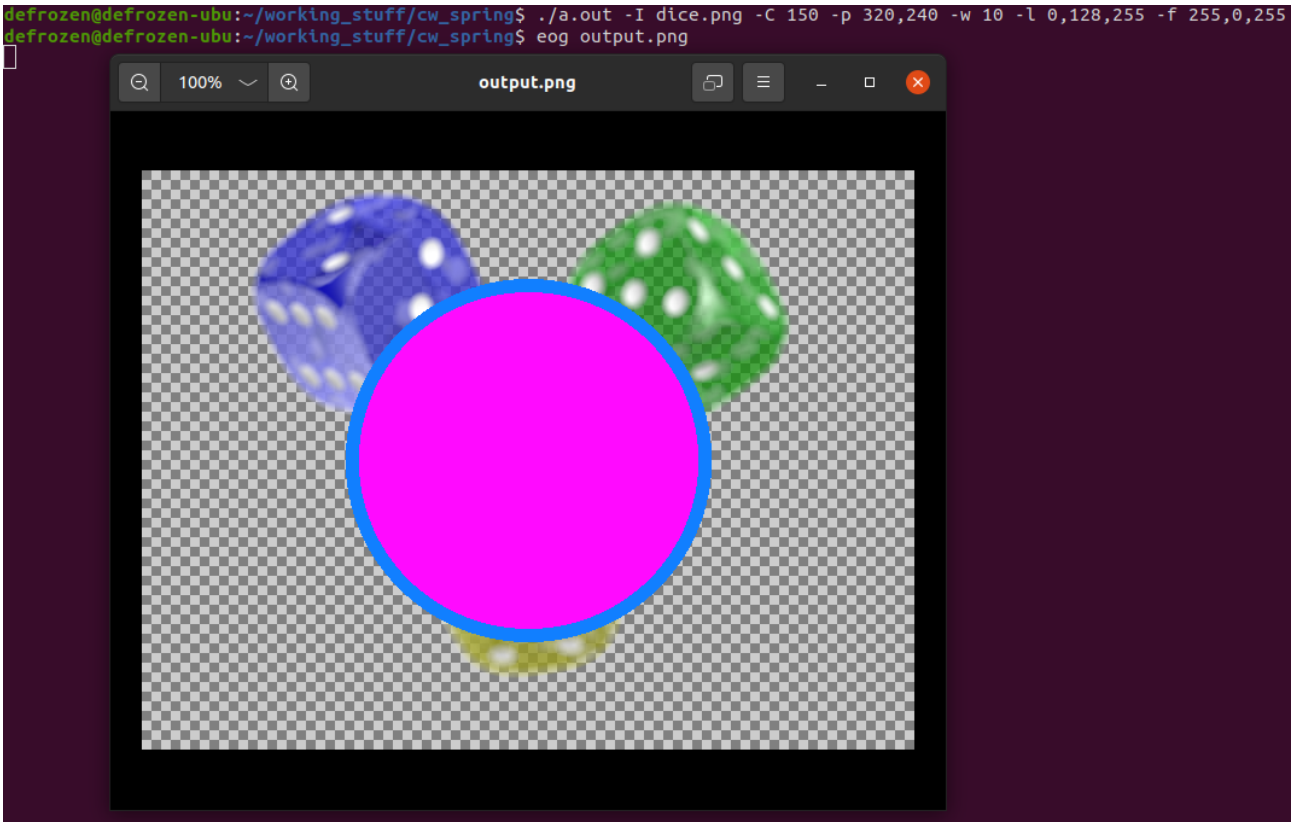
Функция выводит справку об использовании программы со всеми возможными ключами.

1.15. Описание функции optsProcessing()

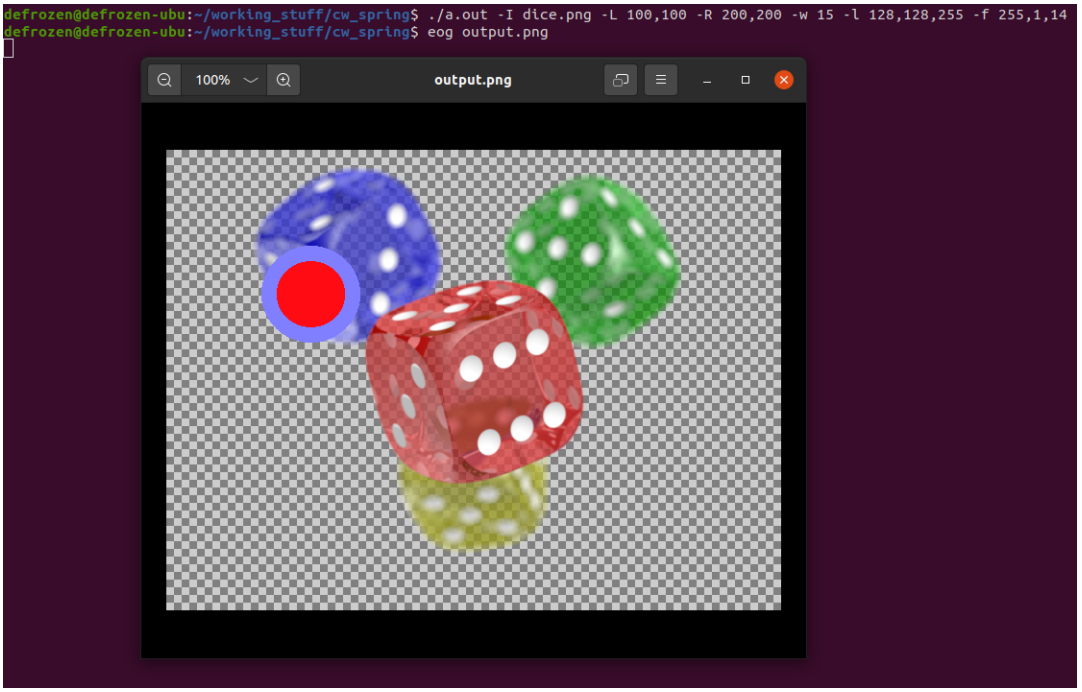
Функция разбирает введенные аргументы и записывает данные в файл конфигурации.

2. ТЕСТИРОВАНИЕ

2.1. Тестирование опции -C (--circle)



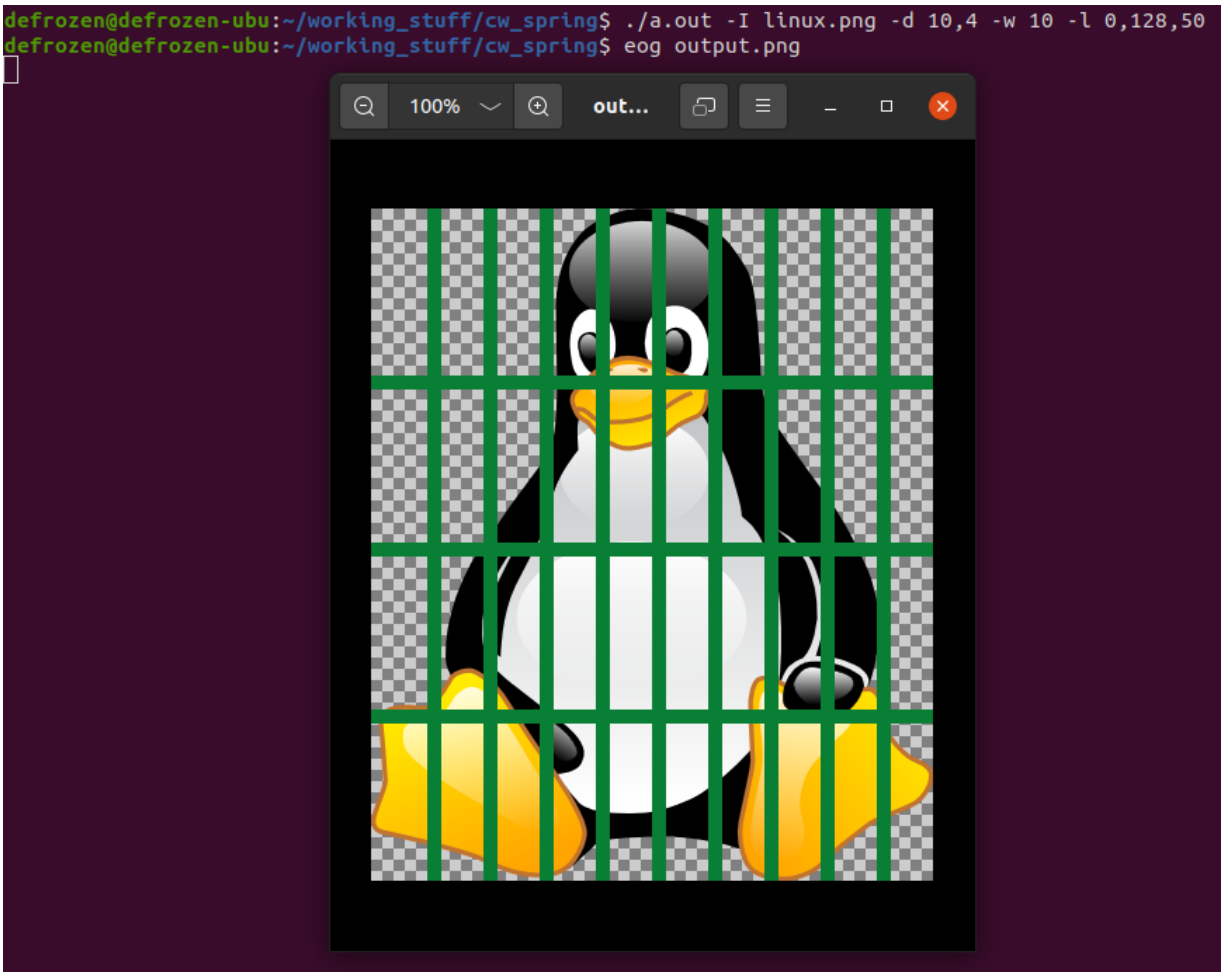
2.2. Тестирование опции -L -R (--lt –rb)



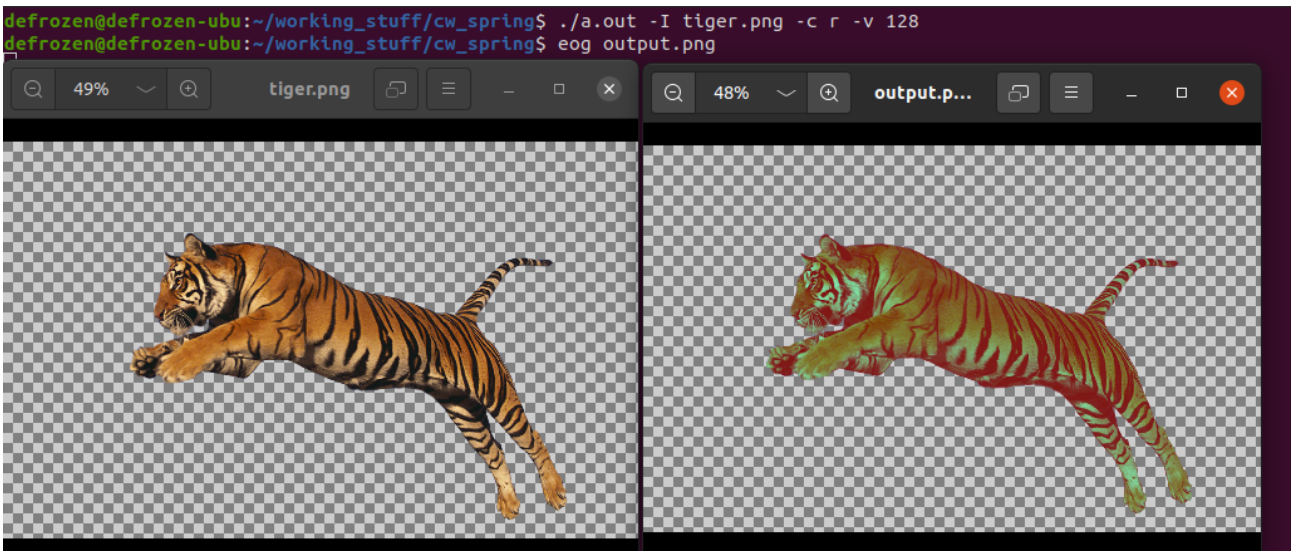
2.3. Тестирование опции -i (--info)

```
defrozen@defrozen-ubu:~/working_stuff/cw_spring$ ./a.out -I dice.png -i
Width: 640 pixels
Height: 480 pixels
Bit depth: 8
Color type: 6
```

2.4. Тестирование опции -d (--divide)



2.5. Тестирование опции -c (--color)



ЗАКЛЮЧЕНИЕ

В ходе выполнения работы была написана программа, выполняющая считывание, обработку PNG изображения, а также сохранение полученного результата в новый файл. Программа проводит проверку корректности поданных на вход аргументов и выводит соответствующие сообщения. Взаимодействие пользователя и утилиты осуществляется через CLI (Command line interface).

ПРИЛОЖЕНИЕ А

Название файла: cw.c

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

void printHelp(){
	printf("-I --input - filename to open\n");
	printf("-O --output - filename to save\n");
	printf("-i --info - prints general information about picture
\n\n");
	printf("\tTo draw a circle:\n\n");
	printf("-C --circle, arg - radius, amount of pixels\n");
	printf("-p --pos - coordinates of center !MUST BE WITHIN PICTURE
OR ELSE WON'T WORK!\n");
	printf("\t\tOR\n");
	printf("-L --lt in /X,Y/ format for right top corner AND -R --rb
in /X,Y/ format for right bottom corner\n");
	printf("WARNING: corners must be WITHIN picture AND form a
square!\n\n");
	printf("-w, --width\n");
	printf("-l --linecolor - in format /R,G,B/\n");
	printf("-f --fill - in format /R,G,B/\n\n");
	printf("\tTo change the overall color of picture:\n");
	printf("-c --color - in format /R/ or /G/ or /B/\n");
	printf("-v --value - sets the value of color, in range
0-255\n\n");
	printf("\tTo divide the picture in frames\n");
	printf("-d --divide - args in format /X,Y/, should be less than
height or width divided by 2\n");
	printf("-w, --width\n");
	printf("-l --linecolor - in format /R,G,B/\n");

};

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 Configs{
	int cfg_error_flag;
	int info_flag;
	char* name;
	char* output_name;
	// CIRCLE
```

```
        int circle;
        char* position;
        char* left_top_corner;
        char* right_bot_corner;
        int width;
        char* linecolor;
        char* fill;
        //COLOR CHANGING
        char color;
        int var;
        //DIVIDING
        char* div;
        //char* linecolor
        //int width
};

void optsProcessing (int argc, char* argv[], struct Configs *config){
        char *opts = "hI:O:iC:p:w:l:f:c:v:d:L:R:";
        struct option longOpts[]={
        {"help", no_argument, 0, 'h'},
        {"input", required_argument, 0, 'I'},
        {"output", required_argument, 0, 'O'},
        {"info", no_argument, 0, 'i'},
        {"circle", required_argument, 0, 'C'},
        {"pos", required_argument, 0, 'p'},
        {"width", required_argument, 0, 'w'},
        {"linecolor", required_argument, 0, 'l'},
        {"fill", required_argument, 0, 'f'},
        {"color", required_argument, 0, 'c'},
        {"value", required_argument, 0, 'v'},
        {"divide", required_argument, 0, 'd'},
        {"lt", required_argument, 0, 'L'},
        {"rb", required_argument, 0, 'R'},
        {0, 0, 0, 0}
        };
        int opt;
        int longIndex;
        opt = getopt_long(argc, argv, opts, longOpts, &longIndex);
        while(opt != -1){
        switch(opt){
                case 'h':
                printHelp();
                exit(1);
                break;
                case 'I':
                config->name = optarg;
                break;
                case 'C':
                config->circle = atoi(optarg);
                break;
                case 'p':
                config->position = optarg;
                break;
                case 'w':
                config->width = atoi(optarg);
                break;
                case 'f':
```

```
                config->fill = optarg;
                break;
                case 'l':
                config->linecolor = optarg;
                break;
                case 'c':
                if(strlen(optarg) == 1){
                        config->color = optarg[0];
                }else{
                        printf("Wrong input format of color\n");
                        printHelp();
                }
                break;
                case 'v':
                config->var = atoi(optarg);
                break;
                case 'd':
                config->div = optarg;
                break;
                case 'O':
                config->output_name = optarg;
                break;
                case 'L':
                config->left_top_corner = optarg;
                break;
                case 'R':
                config->right_bot_corner = optarg;
                break;
                case 'i':
                config->info_flag = 1;
                break;

        }
        opt = getopt_long(argc, argv, opts, longOpts, &longIndex);
        }
        argc -= optind;
        argv += optind;
};

void 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){
        printf("No such file!\n");
        exit(1);
        }

        fread(header, 1, 8, fp);

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

        image->info_ptr = png_create_info_struct(image->png_ptr);
```

```
	if (setjmp(png_jmpbuf(image->png_ptr))){
	// Some error handling: error during init_io
	printf("error during init_io\n");
	exit(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
	printf("error during read_image\n");
	exit(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);
}

void write_png_file(char *file_name, struct Png *image) {
	int x,y;
	/* create file */
	FILE *fp = fopen(file_name, "wb");

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

	image->info_ptr = png_create_info_struct(image->png_ptr);

	if (setjmp(png_jmpbuf(image->png_ptr))){
	// Some error handling: error during init_io
	printf("error during init_io\n");
	exit(1);
```

```
	}

	png_init_io(image->png_ptr, fp);


	/* write header */
	if (setjmp(png_jmpbuf(image->png_ptr))){
	// Some error handling: error during writing header
	printf("error during writing header\n");
	exit(1);
	}

	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
	printf("error during writing bytes\n");
	exit(1);
	}

	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
	printf("error during end of write\n");
	exit(1);
	}

	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 printInfo(struct Png* image){
	printf("Width: %d pixels\n", image->width);
	printf("Height: %d pixels\n", image->height);
	printf("Bit depth: %d\n", image->bit_depth);
	printf("Color type: %d\n", image->color_type);
}

void checkColor(int R, int G, int B){
	int errFlag = 0;
	if(R < 0 || R > 255 ){
```

```
       printf("Red color channel value is incorrect. Check the input
data.\n");
       errFlag = 1;
       };
       if(G < 0 || G > 255 ){
       printf("Green color channel value is incorrect. Check the input
data.\n");
       errFlag = 1;
       };
       if(B < 0 || B > 255 ){
       printf("Blue color channel value is incorrect. Check the input
data.\n");
       errFlag = 1;
       };
       if(errFlag){
       exit(1);
       };
}

void drawPixel(struct Png *image, int x, int y, int color_R, int
color_G, int color_B){
       if(x >= 0 && y >=0 && x < image->width && y < image->height){
       png_byte *row = image->row_pointers[y];
       png_byte *ptr = &(row[x*4]);
              ptr[0] = color_R;
              ptr[1] = color_G;
              ptr[2] = color_B;
              ptr[3] = 255;
       }
}

void drawOutline(struct Png *image,int rad, int x, int y, int line_R,
int line_G, int line_B, int** mask){
       int currX = 0;
    int currY = rad;
    int delta = 1 - 2 * rad;
    int error = 0;
    while(currY >= 0) {
       drawPixel(image, x+currX, y+currY, line_R, line_G, line_B);
       mask[y+currY][x+currX] = 1;
       drawPixel(image, x+currX, y-currY, line_R, line_G, line_B);
       mask[y+currY][x-currX] = 1;
       drawPixel(image, x-currX, y+currY, line_R, line_G, line_B);
       mask[y-currY][x+currX] = 1;
        drawPixel(image, x-currX, y-currY, line_R, line_G, line_B);
       mask[y-currY][x-currX] = 1;
        error = 2 * (delta + currY) - 1;
        if(delta < 0 && error <= 0) {
              ++currX;
              delta += 2 * currX + 1;
              continue;
        }
        error = 2 * (delta - currX) - 1;
        if(delta > 0 && error > 0) {
              --currY;
              delta += 1 - 2 * currY;
              continue;
```

```
        }
        ++currX;
        delta += 2 * (currX-currY);
        --currY;
    }
}

void Fillment(struct Png* image, int** mask, int x, int y, int fill_R,
int fill_G, int fill_B){
       if(x >= 0 && x < image->width && y >=0 && y < image->height){
       png_byte *row = image->row_pointers[y];
       png_byte *ptr = &(row[x*4]);
       if (x >= 0 && x <= image->width && y>=0 && y <= image->height &&
mask[y][x] == 0){
              drawPixel(image, x, y, fill_R, fill_G, fill_B);
              mask[y][x] = 1;
              Fillment(image, mask, x+1, y, fill_R, fill_G, fill_B);
              Fillment(image, mask, x, y+1, fill_R, fill_G, fill_B);
              Fillment(image, mask, x-1, y, fill_R, fill_G, fill_B);
              Fillment(image, mask, x, y-1, fill_R, fill_G, fill_B);
       };
       };
}

void strtokForTwo(char* mergedPos, int* x, int* y){
       char* pch;
       pch = strtok(mergedPos, ",");
       if(pch != NULL){
              *x = atoi(pch);
       }else{
              printf("Wrong input format!\n\n");
              printHelp;
              exit(1);
       };
       pch = strtok(NULL, ",");
       if(pch != NULL){
              *y = atoi(pch);
       }else{
              printf("Wrong input format!\n\n");
              printHelp();
              exit(1);
       };
       pch = strtok(NULL, ",");
       if(pch != NULL){
              printf("Wrong input format!\n\n");
              printHelp();
              exit(1);
       }
}

void strtokForThree(char* mergedColors, int* R, int* G, int* B){
       char* pch = strtok(mergedColors, ",");
       if(pch != NULL){
       *R = atoi(pch);
       }else{
       printf("Wrong input format!\n");
       printHelp();
```

```
        return;
        };
        pch = strtok(NULL, ",");
        if(pch != NULL){
        *G = atoi(pch);
        }else{
        printf("Wrong input format!\n");
        printHelp();
        return;
        };
        pch = strtok(NULL, ",");
        if(pch != NULL){
        *B = atoi(pch);
        }else{
        printf("Wrong input format!\n");
        printHelp();
        return;
        };
        pch = strtok(NULL, ",");
        if(pch != NULL){
        printf("Wrong input format!\n\n");
        printHelp();
        return;
        }
}

void drawCircle(struct Png *image, struct Configs cfg){
        int* mask[image->height];
        for(int i = 0; i < image->height; i++){
        mask[i] = malloc(image->width*sizeof(int));
        };
        int x;
        int y;
        char* pch;
        if(cfg.circle != 0){
        char* mergedPos = malloc(101*sizeof(char));
        strcpy(mergedPos, cfg.position);
        strtokForTwo(mergedPos, &x, &y);
        free(mergedPos);
        }else{
        int LT_x;
        int LT_y;
        int RB_x;
        int RB_y;
        char* mergedPosLT = malloc(101*sizeof(char));
        strcpy(mergedPosLT, cfg.left_top_corner);
        strtokForTwo(mergedPosLT, &LT_x, &LT_y);
        free(mergedPosLT);
        char* mergedPosRB = malloc(101*sizeof(char));
        strcpy(mergedPosRB, cfg.right_bot_corner);
        strtokForTwo(mergedPosRB, &RB_x, &RB_y);
        free(mergedPosRB);
        if(RB_x - LT_x != RB_y - LT_y){
        printf("Entered coordinates do not form a square! Check your
entered coordinates!\n");
        return;
        }
```

```
        x = LT_x + (RB_x - LT_x) / 2;
        y = LT_y + (RB_y - LT_y) / 2;
        cfg.circle = (RB_x - LT_x) / 2;
        }
        if(x <= 0 || x >= image->width || y <= 0 || y >= image->height){
               printf("Center of circle is outside of picture.\n");
               printHelp();
               return;
        };
        int line_R;
        int line_G;
        int line_B;
        char* mergedColors = malloc(12*sizeof(char));
        strcpy(mergedColors, cfg.linecolor);
        strtokForThree(mergedColors, &line_R, &line_G, &line_B);
        free(mergedColors);
        if(line_R == 0 && line_G == 0 && line_B == 0){
        line_R = 1;
        line_B = 1;
        line_G = 1;
        }
        checkColor(line_R, line_G, line_B);
        int fillFlag = 0;
        int fill_R;
        int fill_G;
        int fill_B;
        if (strcmp(cfg.fill, "") != 0){
        fillFlag = 1;
        char* mergedFill = malloc(12*sizeof(char));
        strcpy(mergedFill, cfg.fill);
        strtokForThree(mergedFill, &fill_R, &fill_G, &fill_B);
        free(mergedFill);
        if(fill_R == 0 && fill_G == 0 && fill_B == 0){
               fill_R = 1;
               fill_B = 1;
               fill_G = 1;
        }
        checkColor(fill_R, fill_G, fill_B);
        }
        // DRAWING CIRCLE
        drawOutline(image, cfg.circle, x, y, line_R, line_G, line_B,
mask);
        if(cfg.width > 2){
        drawOutline(image, cfg.circle-cfg.width, x, y, line_R, line_G,
line_B, mask);
        if(x < image->width && y < image->height && x > 0 && y >0){
               if(x <= (image->width / 2)){
               Fillment(image, mask, x+cfg.circle-1, y, line_R, line_G,
line_B);
               };
               if(x > (image->width / 2)){
               Fillment(image, mask, x-cfg.circle+1, y, line_R, line_G,
line_B);
               };
        };
        };
        if (fillFlag == 1){
```

```
        Fillment(image, mask, x, y, fill_R, fill_G, fill_B);
        }

}

void colorChanger(struct Png *image, char color, int val){
        if(val < 0 || val > 255){
        printf("Wrong value input format!\n");
        printHelp();
        }
        int x,y;
        color = tolower(color);
        if(color != 'r' && color != 'g' && color != 'b'){
        printf("No such RGB channel. Check the input data.\n");
        exit(1);
        }
        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 (color == 'r'){
                ptr[0] = val;
                };
                if (color == 'g'){
                ptr[1] = val;
                };
                if (color == 'b'){
                ptr[2] = val;
                };
        };
        };
};

void divideImage(struct Png *image, struct Configs cfg){
        int amount_x, amount_y;
        char* mergedAmounts = malloc(10*sizeof(char));
        strcpy(mergedAmounts, cfg.div);
        strtokForTwo(mergedAmounts, &amount_x, &amount_y);
        free(mergedAmounts);
        if(amount_x == 0){
        amount_x = 1;
        };
        if(amount_y == 0){
        amount_y = 1;
        }
        if(amount_x > image->width/2 || amount_y > image->height/2){
        printf("Too much parts. Choose amounts less than width and
height.\n");
        exit(1);
        }
        int line_R;
        int line_G;
        int line_B;
        char* mergedColors = malloc(12*sizeof(char));
        strcpy(mergedColors, cfg.linecolor);
        strtokForThree(mergedColors, &line_R, &line_G, &line_B);
        free(mergedColors);
```

```
        if(line_R == 0 && line_G == 0 && line_B == 0){
        line_R = 1;
        line_B = 1;
        line_G = 1;
        }
        checkColor(line_R, line_G, line_B);
        int step_x = image->width / amount_x;
        int step_y = image->height / amount_y;
        for (int i = 0; i < image->height; i++){
        int curr_x = 0;
        for (int j = 0; j < image->width; j++){
               if(j % step_x == 0 && j != 0){
               drawPixel(image, j, i, line_R, line_G, line_B);
               curr_x++;
               for(int k = j; k < j+cfg.width; k++){
                      drawPixel(image, k, i, line_R, line_G, line_B);
               }
               if (curr_x == amount_x-1){
                      break;
               }
               }
        }

        };
        int curr_y = 0;
        for( int i = 0; i < image->height; i++){
        if (i % step_y == 0 && i != 0){
               for(int j = 0; j < image->width; j++){
               drawPixel(image, j, i, line_R, line_G, line_B);
               for(int k = i; k < i+cfg.width; k++){
                      drawPixel(image, j, k, line_R, line_G, line_B);
               }
               }
               curr_y++;
        }
        if(curr_y == amount_y-1){
                      break;
        }
        }
}

int main(int argc, char* argv[]){
       struct Configs cfg = {0, 0, "", "output.png", 0, "1,1", "", "",
0, "0,0,0", "", 0, 255, ""};
       optsProcessing(argc, argv, &cfg);
       if(!strcmp(cfg.name, "")){
       printf("No file on input\n");
       printHelp();
       return 0;
       }
       if(cfg.cfg_error_flag == 1){
       return 0;
       }
       struct Png image;
       read_png_file(cfg.name,&image);
       if(cfg.info_flag == 1){
       printInfo(&image);
```

```
        exit(0);
        };
        if(cfg.circle != 0 || (strcmp(cfg.left_top_corner, "")) &&
(strcmp(cfg.right_bot_corner, ""))){
        drawCircle(&image, cfg);
        write_png_file(cfg.output_name, &image);
        return 0;
        };
        if(cfg.color != 0){
        colorChanger(&image, cfg.color, cfg.var);
        write_png_file(cfg.output_name, &image);
        return 0;
        };
        if(strcmp(cfg.div, "")){
        divideImage(&image, cfg);
        write_png_file(cfg.output_name, &image);
        return 0;
        }
        printf("You have entered less opts than required for
processing!\n");
        return 0;

}
```