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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Обработка изображений в формате PNG

Студент гр. 9383 _____ Никифоров П.А.

Преподаватель _____ Размочаева Н.В.

Санкт-Петербург
2020

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студент Никифоров П.А.

Группа 9383

Тема работы: Обработка изображений в формате PNG

Исходные данные:

Программа должна реализовывать весь следующий функционал по обработке png-файла

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).

Содержание пояснительной записки:

«Содержание», «Введение», «Задание работы», «Ход выполнения работы», «Заключение», «Список использованных источников».

Предполагаемый объем пояснительной записки:

Не менее 10 страниц.

Дата выдачи задания:

Дата сдачи реферата:

Дата защиты реферата:

Студент		Никифоров П.А.
Преподаватель		Размочаева Н.В.

АННОТАЦИЯ

В ходе выполнения курсовой работы была написана программа для обработки изображений в формате PNG. Инструменты обработки предоставлены в соответствии с заданием. Программа содержит в себе функции замены значения определенной компоненты формата RGB, смены одного цвета на другой, и разделения изображения на определенное количество равных частей с помощью линии заданной ширины. Для взаимодействия с пользователем реализован интерфейс командной строки.

SUMMARY

In the course of the course work was written program for image processing in PNG format. Processing tools are provided according to the assignment. The program contains functions of replacing the value of a specific component of the RGB format, changing one color to another, and dividing an image into a certain number of equal parts using a line of a given width. A command line interface is implemented for user interaction.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель работы

Реализация программы для обработки изображений в формате PNG на языке Си.

Основные задачи

Реализация консольного интерфейса для взаимодействия пользователя и программы.

Обеспечение стабильной работы программы, обработка исключительных ситуаций.

Методы решения

Разработка программы происходила на базе операционной системы Windows 10 в редакторе кода CLion 2020.1.

1. ЗАДАНИЕ

Программа должна реализовывать следующий функционал по обработке PNG-файла

- Заменяет все пиксели одного заданного цвета на другой цвет. Функционал определяется:
  - Цвет, который требуется заменить
  - Цвет на который требуется заменить
- 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
  - Разделяет изображение на N*M частей. Реализация: провести линии заданной толщины Функционал определяется:
  - Количество частей по “оси” Y
  - Количество частей по “оси” X
  - Толщина линии
  - Цвет линии

2. ХОД ВЫПОЛНЕНИЯ ПРОГРАММЫ

Программа работает на основе аргументов, полученных из командной строки. Для каждой отдельной функции предусмотрен свой флаг, обозначающий каким образом нужно обработать файл. В справке по работе с программой содержится информация о флагах и необходимых аргументах для обработки файлов. Считав и обработав аргументы, в случае корректного ввода программа передает структуру содержащую аргументы полученные от пользователя и номер операции в функцию, в которой файл с изображением помещается в структуру, и передается соответствующему методу для дальнейшей обработки. При успешной обработке файла изображение извлекается из структуры и возвращается в исходный файл.

3. СТРУКТУРА ПРОГРАММЫ

3.1. Интерфейс командной строки

Для взаимодействия с пользователем реализован интерфейс командной строки, с помощью которого вызывается функционал программы. Аргументы обрабатываются с помощью инструмента get_opt_long(), который принимает аргументы из командной строки и вносит содержимое в структуру globalArgs_t.

3.2. Обработка команд.

Список доступных команд и необходимых для них аргументов выводится при вызове программы с флагом -h. Флаги операций (—filter, —crop, —color) обрабатываются как логический значения, в случае наличия их в списке аргументов соответствующие флаги в программе устанавливаются в положительное значение. Для аргументов, подразумевающих определенные значения (-c, -v, —change, —to, —crop, —vert, —hor, —width, —linecolor) проводится проверка на вход в диапазон, позволяющий корректно обработать изображения, и в случае соответствия они вносятся в структуру globalArgs_t, хранящую необработанные значения аргументов. Если какое либо значение не может быть использовано, программа выводит соответствующее сообщение об ошибке.

Последний аргумент по умолчанию должен быть файлом содержащим изображение в формате png, в ином случае программа выдает сообщение об ошибке. Если файл открывается, программа помещает адрес файла в структуру ProcessedArgs, содержащую аргументы для передачи в функции обработки изображения проверяет установленные флаги операций. Если флаг установлен, но для выполнения операции не хватает аргументов, программа выводит соответствующее сообщение об ошибке. В случае корректных аргументов, программа в случае необходимости производит манипуляции над ними, и помещает их в структуру ProcessedArgs, и передает эту структуру вместе с именованной константой обозначающей нужную операцию функции image_processing().

3.3. Чтение изображения

Функция image_processing(), получив структуру и номер операции, независимо от операции создает структуру Png и передает указатель на эту структуру и адрес изображения функции read_png_file(), где с помощью инструментов из библиотеки libpng информация из файла содержащего изображение помещается в структуру Png.

3.4. Обработка изображения

3.4.1. Фильтр RGB-компонент

Если код операции соответствует константе COMPONENT_VALUE_CHANGE, указатель на изображение и элементы структуры ProcessedArgs (component и value) передаются в функцию change_component() для обработки. Функция change_component() проверяет изображения на соответствие нужной цветовой кодировке (в случае несоответствия программа выдает сообщение об ошибке). Далее обходится каждый пиксель изображения и соответствующая компонента устанавливается в соответствующее значение.

3.4.2. Замена пикселей

Если код операции соответствует константе COLOR_CHANGE, указатель на изображение и элементы структуры ProcessedArgs (from и to) передаются в функцию change_color() для обработки. Функция change_component() проверяет изображения на соответствие нужной цветовой кодировке (в случае несоответствия программа выдает сообщение об ошибке). Далее обходится каждый пиксель изображения и цветовые значения пикселей того цвета, который нужно изменить, заменяются на указанные в новом цвете значения.

3.4.3. Разделение изображения

Если код операции соответствует константе CROP_BY_LINE, указатель на изображение и элементы структуры ProcessedArgs (vert, hor, line_width, line_color) передаются в функцию draw_line() для обработки. Функция draw_line() проверяет изображения на соответствие нужной цветовой кодировке (в случае несоответствия программа выдает сообщение об ошибке). Далее

вычисляются координаты положения центра линии по вертикали и горизонтали, и после этого обходится каждый пиксель изображения, в случае если пиксель находится в том месте, где должна проходить линия, его цвет устанавливается в выбранный пользователем цвет линии.

3.4.4 Запись изображения

После выполнения обработки указатель на структуру Png и адрес изображения передаются функции write_png_file(), где после проверок на возможные ошибки записи структура изображения возвращается в изначальный файл.

ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы была реализована стабильная программа для обработки изображений в формате PNG. Были получены навыки работы со сторонней библиотекой обработки изображений (libpng), реализации пользовательского интерфейса командной строки, работы с функцией getopt_long(), обработки изображений. В ходе реализации функционала соответствующего заданию был разработан объемный функционал по обработке аргументов полученных из командной строки, отслеживанию ошибок при данной обработке и обработке файлов в формате PNG.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Керниган Б. И Ритчи Д. Язык программирования Си. М.: Вильямс, 1978 288 с.

2. Грег Перри, Дин Миллер Программирование на C для начинающих, Эксмо 2014 369 с.

3. Стив Оолайн C Elements of Style M.: M&T books, 1992 456 с.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл: main.c

```
#include <stdio.h>
#include <unistd.h>
#include <getopt.h>
#include <stdlib.h>
#include <string.h>
#include "pngdance.h"
#include "structs.h"


Color color_pick(char* color_name);
int component_pick(char* component);


int component_pick(char* component) {
    if (strcmp(component, "Red") == 0) return 0;
    if (strcmp(component, "Green") == 0) return 1;
    return 2;
}

Color color_pick(char* color_name) {
    Color color = {0, 0, 0};
    if (strcmp(color_name, "White") == 0) {
        color.red = 255;
        color.green = 255;
        color.blue = 255;
    } else if (strcmp(color_name, "Red") == 0) {
        color.red = 255;
    } else if (strcmp(color_name, "Blue") == 0) {
        color.blue = 255;
    } else if (strcmp(color_name, "Green") == 0) {
        color.green = 255;
    }else if (strcmp(color_name, "Cyan") == 0) {
        color.green = 255;
        color.blue = 255;
    }else if (strcmp(color_name, "Magenta") == 0) {
        color.red = 255;
        color.blue = 255;
    }else if (strcmp(color_name, "Yellow") == 0) {
        color.red = 255;
        color.green = 255;
    }
    return color;
}


void printHelp(){
    printf("        PNGEDITOR\n");
```

```
    printf("Change RGB-component value:\n");
    printf("--filter -c <component> -v <value> <file> - change
component's value\n");
    printf("Change some color:\n");
    printf("--color --change <color> --to <color> <file>\n");
    printf("Crop image to N*M parts by line:\n");
    printf("--crop --vert <vertical> --hor <horizontal> --width
<line> --linecolor <color> <file>\n");
    printf("Arguments:\n");
    printf("<file> - PNG image file\n");
    printf("<sign>:\n");
    printf("<color>:\n");
    printf("    Black\n");
    printf("    Blue\n");
    printf("    Green\n");
    printf("    Cyan\n");
    printf("    Red\n");
    printf("    Magenta\n");
    printf("    Yellow\n");
    printf("    White\n");
    printf("<component>:\n");
    printf("    Red\n");
    printf("    Green\n");
    printf("    Blue\n");
    printf("<value>:\n");
    printf("    integer in range 0-255 for component's value\n");
    printf("<vertical> <horizontal>:\n");
    printf("    unsigned integers > 0\n");
    printf("<line>:\n");
    printf("    integer > 0\n");
    printf("-h -? --help - help\n");
}

int main(int argc, char* argv[]){
    char *opts = "c:v:h?";
    int colorFlag = 0;
    int filterFlag = 0;
    int cropFlag = 0;
    struct globalArgs_t arguments = {
        (char*) malloc(10*sizeof(char)),
        -1,
        NULL,
        (char*) malloc(10*sizeof(char)),
        (char*) malloc(10*sizeof(char)),
        -1,
        -1,
        -1,
        (char*) malloc(10*sizeof(char))
    };
    struct option longOpts[]={
        {"filter", no_argument, &filterFlag, 'f'},
```

```
        {"color", no_argument, &colorFlag, 1},
        {"change", required_argument, NULL, 2},
        {"to", required_argument, NULL, 3},
        {"vert", required_argument, NULL, 4},
        {"hor", required_argument, NULL, 5},
        {"width", required_argument, NULL, 6},
        {"linecolor", required_argument, NULL, 7},
        {"help", no_argument, NULL, 'h'},
        {"crop", no_argument, &cropFlag, 8},
        { NULL, 0, NULL, 0}
    };
    int opt;
    int value;
    int longIndex;
    int inputErrorFlag = 0;
    ProcessedArgs pr_args;
    char colors[8][10] = {"Red", "Green", "Blue", "Black",
"White", "Cyan", "Yellow", "Magenta"};
    opt = getopt_long(argc, argv, opts, longOpts, &longIndex);
    while(opt!=-1){
        int color = 0;
        switch(opt){
            case 'v':
                value = atoi(optarg);
                if (value < 0 || value > 255) {
                    inputErrorFlag = 1;
                    printf("Error: wrong value input\n");
                } else {
                    arguments.value = value;
                }
                break;
            case 'c':
                for (int i = 0; i < 3; i++)
                    if (strcmp(optarg, colors[i]) == 0) color = 1;
                if (color) {
                    strcpy(arguments.component, optarg);
                } else {
                    printf("Error: there is no such component!
\n");
                    inputErrorFlag = 1;
                }
                break;
            case 2:
            case 3:
            case 7:
                for (int i = 0; i < 8; i++)
                    if (strcmp(optarg, colors[i]) == 0) color = 1;
                if (color) {
                    if (opt == 2) strcpy(arguments.change,
optarg);
                    if (opt == 3) strcpy(arguments.to, optarg);
```

```
                    if (opt == 7) strcpy(arguments.line_color,
optarg);
                } else {
                    printf("Error: there is no such color as %s!
\n", optarg);
                    inputErrorFlag = 1;
                }
                break;
            case 4:
            case 5:
            case 6:
                value = atoi(optarg);
                if (value < 0) {
                    inputErrorFlag = 1;
                    printf("Error: wrong value input\n");
                } else {
                    if (opt == 4) arguments.vert = value;
                    if (opt == 5) arguments.hor = value;
                    if (opt == 6) arguments.line_width = value;
                }
                break;
            case 'h':
            case '?':
                printHelp();
                return 0;
        }
        longIndex = 0;
        opt = getopt_long(argc, argv, opts, longOpts, &longIndex);
        if (inputErrorFlag) break;
    }
    FILE *fp = fopen(argv[argc - 1], "r");
    if (fp == NULL) {
        printf("Error: couldn't open png file\n");
        return 1;
    } else {
        pr_args.image = (char*) malloc(100*sizeof(char));
        strcpy(pr_args.image, argv[argc-1]);
    }
    if (filterFlag) {
        if (strlen(arguments.component) > 0 && arguments.value !=
-1) {
            pr_args.component =
component_pick(arguments.component);
            pr_args.value = arguments.value;
            image_processing(pr_args, COMPONENT_VALUE_CHANGE);
            printf("Filter succsesfully applied\n");
        } else {
            printf("Filter impossible\n");
            return 1;
        }
    }
```

```
    if (colorFlag) {
        if (strlen(arguments.change) > 0 && strlen(arguments.to) >
0) {
            pr_args.from = color_pick(arguments.change);
            pr_args.to = color_pick(arguments.to);
            image_processing(pr_args, COLOR_CHANGE);
            printf("Color changing succeed\n");
        } else {
            printf("Color changing impossible\n");
            return 1;
        }
    }
    if (cropFlag) {
        if (strlen(arguments.line_color) > 0 &&
arguments.line_width != -1
        && arguments.hor != -1 && arguments.vert != -1) {
            pr_args.hor = arguments.hor;
            pr_args.vert = arguments.vert;
            pr_args.line_color = color_pick(arguments.line_color);
            pr_args.line_width = arguments.line_width;
            image_processing(pr_args, CROP_BY_LINE);
        } else {
            printf("Crop is impossible\n");
            return 1;
        }
    }
    return 0;
}
```

Файл: pngdance.c

```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <stdarg.h>
#include "structs.h"


#define PNG_DEBUG 3
#include <png.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;
};
```

```
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("Error: file could not be opened\n");
        return;
    }


    fread(header, 1, 8, fp);
    if (png_sig_cmp((png_const_bytep) header, 0, 8)){
        printf("Error: file is not recognized as a PNG\n");
        return;
    }


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

    if (!image->png_ptr){
        printf("Error: png_create_read_struct failed\n");
        return;
    }

    image->info_ptr = png_create_info_struct(image->png_ptr);
    if (!image->info_ptr){
        printf("Error: png_create_info_struct failed\n");
        return;
    }

    if (setjmp(png_jmpbuf(image->png_ptr))){
        printf("Error: error during init_io\n");
        return;
    }

    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))){
        printf("Error: error during read_image\n");
        return;
    }

    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");
    if (!fp){
        printf("Error: 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){
        printf("Error: png_create_write_struct failed\n");
        return;
    }

    image->info_ptr = png_create_info_struct(image->png_ptr);
    if (!image->info_ptr){
        printf("Error: png_create_info_struct failed\n");
        return;
    }

    if (setjmp(png_jmpbuf(image->png_ptr))){
        printf("Error: error during init_io\n");
        return;
    }
```

```
    png_init_io(image->png_ptr, fp);


    /* write header */
    if (setjmp(png_jmpbuf(image->png_ptr))){
        printf("Error: 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))){
        printf("Error: error during writing bytes\n");
        return;
    }

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


    /* end write */
    if (setjmp(png_jmpbuf(image->png_ptr))){
        printf("Error handling: 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 change_component(struct Png *image, int component, int value)
{
    int x,y;
    if (png_get_color_type(image->png_ptr, image->info_ptr) ==
PNG_COLOR_TYPE_RGB){
        printf("Error: 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){
        printf("Eerror: 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]);
            ptr[component] = value;
        }
    }
}

void draw_line(struct Png *image, int vert, int hor, int width,
Color color) {
    int x,y;
    if (png_get_color_type(image->png_ptr, image->info_ptr) ==
PNG_COLOR_TYPE_RGB){
        printf("Error: 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){
        printf("Error: color_type of input file must be
PNG_COLOR_TYPE_RGBA\n");
        return;
    }
    int each_x = image->width/hor;
    int each_y = image->height/vert;
    int x_start, x_end, y_start, y_end;
    if (width * hor > image->width || width * vert > image-
>height) {
        printf("Error: too much pieces or too wide line\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]);
            x_start = each_x - width/2;
            x_end = each_x + width/2;
            y_start = each_y - width/2;
            y_end = each_y + width/2;
```

```
            if (x >= x_start && x <= x_end && x_end < image-
>width) {
                ptr[0] = color.red;
                ptr[1] = color.green;
                ptr[2] = color.blue;
            }
            if (y >= y_start && y <= y_end && y_end < image-
>height) {
                ptr[0] = color.red;
                ptr[1] = color.green;
                ptr[2] = color.blue;
            }
            if (x == x_end + 1) each_x += image->width/hor;
            if (y == y_end + 1) each_y += image->height/vert;
        }
        each_x = image->width/hor;
    }
}


void change_color(struct Png *image, Color color, Color colorto) {
    int x,y;
    if (png_get_color_type(image->png_ptr, image->info_ptr) ==
PNG_COLOR_TYPE_RGB){
        printf("Error: 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){
        printf("Error: 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 (ptr[0] == color.red && ptr[1] == color.green &&
ptr[2] == color.blue) {
                ptr[0] = colorto.red;
                ptr[1] = colorto.green;
                ptr[2] = colorto.blue;
            }
        }
    }
}
```

```
int image_processing(ProcessedArgs arguments, int operation) {
```

```
    struct Png image;
    read_png_file(arguments.image, &image);
    if (operation == COMPONENT_VALUE_CHANGE) {
        change_component(&image, arguments.component,
arguments.value);
    } else if (operation == COLOR_CHANGE) {
        change_color(&image, arguments.from, arguments.to);
    } else if (operation == CROP_BY_LINE) {
        draw_line(&image, arguments.vert, arguments.hor,
arguments.line_width, arguments.line_color);
    }
    write_png_file(arguments.image, &image);
    return 0;
}
```

Файл: pngdance.h

```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <stdarg.h>
#include "structs.h"
#pragma once

#define PNG_DEBUG 3
#include <png.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;
};

void read_png_file(char *file_name, struct Png *image);
void write_png_file(char *file_name, struct Png *image);
void process_file(struct Png *image);
int image_processing(ProcessedArgs arguments, int operation);
void change_component(struct Png *image, int component, int
value);
void change_color(struct Png *image, Color color, Color colorto);
void draw_line(struct Png *image, int n, int m, int width, Color
color);
```

Файл:structs.h

```
#define COLOR_CHANGE  1
#define COMPONENT_VALUE_CHANGE  2
#define CROP_BY_LINE  3
#pragma once
```

```
typedef struct{
    int red;
    int green;
    int blue;
} Color;

struct globalArgs_t {
    char* component;
    int value;
    FILE* image;
    char* change;
    char* to;
    int vert;
    int hor;
    int line_width;
    char* line_color;
} globalArgs;

typedef struct {
    Color from;
    Color to;
    char* image;
    int component;
    int value;
    int vert;
    int hor;
    int line_width;
    Color line_color;
} ProcessedArgs;
```

Файл: `Makefile`

```
all: main

main: main.c pngdance.c
    gcc pngdance.c main.c -lpng -o pngedit
```