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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Обработка PNG изображения

Студент гр. 3343 _____ Пивоев Н. М.

Преподаватель _____ Государкин Я.С.

Санкт-Петербург

2024

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студент: Пивоев Никита

Группа: 3343

Тема: Обработка PNG изображения

Условия задания (Вариант 5.16):

Программа должна иметь следующие функции по обработке изображений:

1. Копирование заданной области. Флаг для выполнения данной операции: `--copy`. Функционал определяется:
   - Координатами левого верхнего угла области-источника. Флаг `--left_up`, значение задаётся в формате `left.up`, где left – координата по x, up – координата по y.
   - Координатами правого нижнего угла области-источника. Флаг `--right_down`, значение задаётся в формате `right.down`, где right – координата по x, down – координата по y.
   - Координатами левого верхнего угла области-назначения. Флаг `--dest_left_up`, значение задаётся в формате `left.up`, где left – координата по x, up – координата по y.
2. 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`).
3. Сделать рамку в виде узора. Флаг для выполнения данной операции: `--ornament`. Рамка определяется:

- Узором. Флаг `--pattern`. Обязательные значения: rectangle и circle, semicircles.
- 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.

4. 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).
   - Цветом линии для обводки. Флаг `--border_color` (работает аналогично флагу `--color`).
   - Толщиной линии для обводки. Флаг `--thickness`. На вход принимает число больше 0.

Дата выдачи задания: 18.03.2024

Дата сдачи реферата: 15.05.2024

Дата защиты реферата: 15.05.2024

АННОТАЦИЯ

В процессе написания курсовой работы создан проект на языке C с использованием библиотеки libpng, обрабатывающий PNG изображения. Для взаимодействия с программой добавлен интерфейс командной строки (CLI – command line interface). Программа реализует следующие возможности: копирование заданной области, замена цвета, рисование рамки в виде узора, обведение всех прямоугольников заданного цвета. Сборка проекта осуществляется с помощью утилиты make.

ВВЕДЕНИЕ

Цель работы заключается в изучении структуры PNG изображений, освоении работы с ними на языке программирования C. Необходимо разработать программу, выполняющую несколько функций по обработке, считыванию и записи изображений, а также взаимодействию с пользователем через интерфейс командной строки.

1. ОПИСАНИЕ СТРУКТУРЫ ПРОГРАММЫ

Описание структур:

1. *RGB* – структура, содержащая информацию о цвете пикселя.
2. *OptParams* – структура, в которую записывается информация о поданных флагах и аргументах и которая используется во всех заданиях.
3. *Png* – структура, описывающая изображение, его характеристики: высоту, ширину, глубину цвета, указатели на строки и т. д.

Описание функций:

1. *int main(int argc, char** argv)* – ключевая функция программы, вызывающая обработку командной строки и требуемую задачу.
2. *OptParams* parseCommandLine(int argc, char** argv)* – обрабатывает командную строку, осуществляет работу CLI.
3. *OptParams* initOptParams(OptParams* opt)* – заполняет структуру *opt* (options) пустыми, отрицательными данными.
4. *void printHelp()* – выводит информацию о флагах и авторе программы.
5. *void printInfo(Png* image)* – выводит информацию о изображении.
6. *char** parseArgs(char* arg, int size)* – преобразует и разделяет аргументы для записи в структуру *OptParams*.
7. *void raiseError(const char* message, int error)* – прекращает работу программы с ошибкой, выводит её.
8. *void checkExtraArgs(OptParams* opt)* – проверяет записанные флаги на наличие лишних, относящихся к другому заданию (например, задание –copy, подан дополнительный флаг –pattern, будет вызвана ошибка).
9. *void read_png_file(char* file_name, Png* image)* – считывает изображение, заполняет структуру *Png*, проверяет формат файла.
10. *void setColor(png_byte* ptr, RGB color)* – заменяет цвет в пикселе.
11. *void checkCopy(Png* image, OptParams* opt)* – проверяет параметры для задания *copy*.
12. *void copyArea(Png* image, OptParams* opt)* – выполняет копирование и вставку области.

13. *void checkColor(RGB color)* – проверяет поданную структуру *RGB* на правильность.

14. *void checkReplace(Png* image, OptParams* opt)* – проверяет параметры задания *color_replace*.

15. *void replaceColor(Png* image, OptParams* opt)* – заменяет все пиксели изображения определённого цвета на другой.

16. *void checkOrnament(Png* image, OptParams* opt)* – проверяет параметры задания *ornament*.

17. *void createOrnament(Png* image, OptParams* opt)* – создаёт по краям изображения рамку из прямоугольников, круга (по центру) или полуокружностей.

18. *void checkRects(Png* image, OptParams* opt)* – проверяет параметры задания *filled_rects*.

19. *void fillRects(Png* image, OptParams* opt)* – ищет все прямоугольники заданного цвета и обводит их линией.

20. *void write_png_file(char* file_name, Png* image)* – записывает изменения в изображение, очищает память его указателей на строки.

Созданная программа разделена на модули, что хорошо сказывается на возможности развития программы в целом. Все функции распределены по соответствующим файлам, отвечающим за какую-либо группу действий. Программа собирается с использованием Makefile, что обеспечивает как легкость в редактировании зависимостей между модулями, так и удобство в управлении процессом компиляции. Разработанный код см. в приложении А.

ТЕСТИРОВАНИЕ

Рисунок 1 – изображение для тестирования

1. Задание *copy*:

   Аргументы запуска: ./cw —copy —left_up 30.50 —right_down 300.200 —dest_left_up 350.150 —input ornament_image.png —output out.png

Рисунок 2 – результат работы для задания *copy*

2. Задание *color_replace*:

Аргументы запуска: ./cw —color_replace —old_color 0.0.0 —new_color 255.0.255

--input ornament_image.png —output out.png

Рисунок 3 – результат работы для задания *color_replace*

3. Задание *ornament*:

   Аргументы запуска: ./cw —ornament —pattern semicircles —thickness 10 —count 5 —color 255.0.255 —input ornament_image.png —output out.png

Рисунок 4 – результат работы для задания *ornament*

4. Задание *filled_rects*:

   Аргументы запуска: ./cw —filled_rects —color 0.0.0 —border_color 255.0.255 —thickness 3 —input ornament_image.png —output out.png

Рисунок 5 – результат работы для задания *filled_rects*

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы была создан проект на языке программирования C с использованием библиотеки libpng, осуществляющий обработку PNG изображения, а именно: копирование заданной области, замена цвета, рисование рамки в виде узора, обведение всех прямоугольников заданного цвета. С помощью утилиты make реализована сборка проекта. Организация программы и выбор заданий осуществляется через CLI.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.h

```
#ifndef MAIN_H
#define MAIN_H

#include "struct.h"
#include "optParams.h"
#include "checkData.h"
#include "operatePng.h"

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <png.h>

#endif
```

Название файла: optParams.h

```
#ifndef FUNC_PARAMS_H
#define FUNC_PARAMS_H

#include "main.h"

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <getopt.h>

OptParams* initOptParams(OptParams* opt);
void printHelp();
void printInfo(Png* image);
char** parseArgs(char* arg, int size);
OptParams* parseCommandLine(int argc, char** argv);

#endif
```

Название файла: operatePng.h

```
#ifndef OPERATEPNG_H
#define OPERATEPNG_H

#include "main.h"

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <unistd.h>
#include <png.h>

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

```
void setColor(png_byte* ptr, RGB color);

void copyArea(Png* image, OptParams* opt);
void replaceColor(Png *image, OptParams* opt);
void createOrnament(Png* image, OptParams* opt);
void fillRects(Png* image, OptParams* opt);

#endif
```

Название файла: checkData.h

```
#ifndef CHECKDATA_H
#define CHECKDATA_H

#define FILE_ERROR 40

#define FILE_FORMAT_ERROR 41

#define MISSING_ARGUMENTS_ERROR 42

#define ARGUMENTS_ERROR 43

#define MULTIPLE_TASK_ERROR 44

#include "main.h"

#include <stdio.h>
#include <stdlib.h>
#include <png.h>

void raiseError(const char* message, int error);
void checkExtraArgs(OptParams* opt);
void checkCopy(Png* image, OptParams* opt);
void checkColor(RGB color);
void checkReplace(Png* image, OptParams* opt);
void checkOrnament(Png* image, OptParams* opt);
void checkRects(Png* image, OptParams* opt);

extern const char* fileTypeError;
extern const char* inputError;
extern const char* outputIsImputError;

extern const char* colorError;
extern const char* patternError;
extern const char* thicknessError;
extern const char* countError;

extern const char* argsError;
extern const char* taskError;

#endif
```

Название файла: struct.h

```
#ifndef STRUCT_H
#define STRUCT_H

#include <png.h>
```

```
#include <stdbool.h>

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

typedef struct {
    char* input;
    char* output;
    bool info;

    bool copy;
    int copy_left;
    int copy_up;
    int copy_right;
    int copy_down;
    int copy_dest_left;
    int copy_dest_up;

    bool color_replace;
    RGB old_color;
    RGB new_color;

    bool ornament;
    int ornament_pattern;
    int ornament_count;

    bool filled_rects;
    RGB rects_border_color;

    RGB color;
    int thickness;
} OptParams;

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

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

#endif
#endif
```

Название файла: main.c

```
#include "../include/main.h"

int main(int argc, char** argv) {
    OptParams* opt = parseCommandLine(argc, argv);
    checkExtraArgs(opt);
```

```
    Png image;
    read_png_file(opt->input, &image);

    if (opt->info)
        printInfo(&image);

    if (opt->copy) {
        checkCopy(&image, opt);
        copyArea(&image, opt);
    }

    if (opt->color_replace) {
        checkReplace(&image, opt);
        replaceColor(&image, opt);
    }

    if (opt->ornament) {
        checkOrnament(&image, opt);
        createOrnament(&image, opt);
    }

    if (opt->filled_rects) {
        checkRects(&image, opt);
        fillRects(&image, opt);
    }

    write_png_file(opt->output, &image);
    free(opt);
    return 0;
}
```

Название файла: optParams.c

```
#include "../include/optParams.h"

OptParams* initOptParams(OptParams* opt) {
    opt->input = NULL;
    opt->output = NULL;
    opt->info = false;

    opt->copy = false;
    opt->copy_left = -1;
    opt->copy_up = -1;
    opt->copy_right = -1;
    opt->copy_down = -1;
    opt->copy_dest_left = -1;
    opt->copy_dest_up = -1;

    opt->color_replace = false;
    opt->old_color.r = opt->old_color.g = opt->old_color.b = -1;
    opt->new_color.r = opt->new_color.g = opt->new_color.b = -1;

    opt->ornament = false;
    opt->ornament_pattern = -1;
    opt->ornament_count = -1;

    opt->filled_rects = false;
```

```
    opt->rects_border_color.r = opt->rects_border_color.g = opt-
>rects_border_color.b = -1;

    opt->color.r = opt->color.g = opt->color.b = -1;
    opt->thickness = -1;
    return opt;
}

void printHelp() {
    printf("Course work for option 5.16, created by Pivoev Nikita.\n"
        "Usage: ./cw [FLAGS]\n\n"
        "Flags:\n"
        "-h --help: Вывод справочной информации.\n"
        "-i --input: Изменение входного файла.\n"
        "-o --output: Изменение выходного файла.\n"
        "--info: Вывод информации о изображении.\n\n"
        "--copy: Копирование области.\n"
        "--left_up: Координаты левого верхнего угла изображения.\n"
        "--right_down: Координаты правого нижнего угла изображения.\n"
        "--dest_left_up: Координаты левого верхнуго угла области-
назначения.\n\n"
        "--color_replace: Замена цвета всех пикселей на другой.\n"
        "--old_color: Старый цвет для замены.\n"
        "--new_color: Новый цвет для замены.\n\n"
        "--ornament: Создание рамки в виде узора.\n"
        "--pattern: Тип узора.\n"
        "--color: Цвет узора.\n"
        "--thickness: Ширина.\n"
        "--count: Количество узоров.\n\n"
        "--filled_rects: Поиск всех залитых прямоугольников заданного
цвета.\n"
        "--color: Цвет искомых прямоугольников.\n"
        "--border_color: Цвет линии обводки.\n"
        "--thickness: Толщина линии для обводки.\n");
}

void printInfo(Png* image) {
    printf("Image settings:\n"
    "Height: %d.\n"
    "Width: %d.\n"
    "Color type: %d.\n",
    image->height, image->width, image->color_type);
}

char** parseArgs(char* arg, int size) {
    char** parsedArgs = malloc(sizeof(char*)*50);
    for (int i = 0; i < size; ++i)
        parsedArgs[i] = malloc(strlen(arg));

    int currentLength;
    int currentElement;
    for (int i = 0; i < strlen(arg); ++i) {
        if (arg[i-1] == '.' && i > 0) {
            parsedArgs[currentElement++][currentLength-1] = '\0';
            currentLength = 0;
        }
        parsedArgs[currentElement][currentLength++] = arg[i];
    }
```

```
    parsedArgs[currentElement][currentLength] = '\0';
    return parsedArgs;
}

OptParams* parseCommandLine(int argc, char** argv) {
    OptParams* opt = malloc(sizeof(OptParams));
    initOptParams(opt);

    opterr = 0;
    const char* short_options = "hi:o:";
    static struct option long_options[] = {
        {"help", 0, NULL, 'h'},
        {"input", 1, NULL, 'i'},
        {"output", 1, NULL, 'o'},

        {"info", 0, NULL, 310},

        {"copy", 0, NULL, 410},
        {"left_up", 1, NULL, 411},
        {"right_down", 1, NULL, 412},
        {"dest_left_up", 1, NULL, 413},

        {"color_replace", 0, NULL, 420},
        {"old_color", 1, NULL, 421},
        {"new_color", 1, NULL, 422},

        {"ornament", 0, NULL, 430},
        {"pattern", 1, NULL, 431},
        {"count", 1, NULL, 432},
        //color
        //thickness

        {"filled_rects", 0, NULL, 440},
        {"border_color", 1, NULL, 441},
        //color
        //thickness

        {"color", 1, NULL, 450},
        {"thickness", 1, NULL, 451},
        {0, 0, 0, 0}
    };

    int value;
    char** args;
    while ((value = getopt_long(argc, argv, short_options, long_options,
NULL)) != -1) {
        switch (value) {
        case 'h': //--help
            printHelp();
            break;
        case 'i': //--input
            opt->input = optarg;
            break;
        case 'o': //--output
            opt->output = optarg;
            break;
        case 310: //--info
            opt->info = true;
```

```
            break;

        case 410: //--copy
            if (opt->color_replace || opt->ornament || opt-
>filled_rects){
                raiseError(taskError, 44);
            }
            opt->copy = true;
            break;
        case 411: //--left_up
            args = parseArgs(optarg, 2);
            opt->copy_left = strtol(args[0], NULL, 10);
            opt->copy_up = strtol(args[1], NULL, 10);
            break;
        case 412: //--right_down
            args = parseArgs(optarg, 2);
            opt->copy_right = strtol(args[0], NULL, 10);
            opt->copy_down = strtol(args[1], NULL, 10);
            break;
        case 413: //--dest_left_up
            args = parseArgs(optarg, 2);
            opt->copy_dest_left = strtol(args[0], NULL, 10);
            opt->copy_dest_up = strtol(args[1], NULL, 10);
            break;

        case 420: //--color_replace
            if (opt->copy || opt->ornament || opt->filled_rects) {
                raiseError(taskError, 44);
            }
            opt->color_replace = true;
            break;
        case 421: //--old_color
            args = parseArgs(optarg, 3);
            opt->old_color.r = strtol(args[0], NULL, 10);
            opt->old_color.g = strtol(args[1], NULL, 10);
            opt->old_color.b = strtol(args[2], NULL, 10);
            break;
        case 422: //--new_color
            args = parseArgs(optarg, 3);
            opt->new_color.r = strtol(args[0], NULL, 10);
            opt->new_color.g = strtol(args[1], NULL, 10);
            opt->new_color.b = strtol(args[2], NULL, 10);
            break;

        case 430: //--ornament
            if (opt->copy || opt->color_replace || opt->filled_rects){
                raiseError(taskError, 44);
            }
            opt->ornament = true;
            break;
        case 431: //--pattern
            if (strcmp(optarg, "rectangle") == 0)
                opt->ornament_pattern = 0;
            else if (strcmp(optarg, "circle") == 0)
                opt->ornament_pattern = 1;
            else if (strcmp(optarg, "semicircles") == 0)
                opt->ornament_pattern = 2;
            else
```

```
                raiseError(patternError, 43);
            break;
        case 432: //--count
            opt->ornament_count = strtol(optarg, NULL, 10);
            break;

        case 440: //--filled_rects
            if (opt->copy || opt->color_replace || opt->ornament) {
                raiseError(taskError, 44);
            }
            opt->filled_rects = true;
            break;
        case 441: //--border_color
            args = parseArgs(optarg, 3);
            opt->rects_border_color.r = strtol(args[0], NULL, 10);
            opt->rects_border_color.g = strtol(args[1], NULL, 10);
            opt->rects_border_color.b = strtol(args[2], NULL, 10);
            break;

        case 450: //--color
            args = parseArgs(optarg, 3);
            opt->color.r = strtol(args[0], NULL, 10);
            opt->color.g = strtol(args[1], NULL, 10);
            opt->color.b = strtol(args[2], NULL, 10);
            break;
        case 451: //--thickness
            opt->thickness = strtol(optarg, NULL, 10);
            break;

        case '?': //missing args
            raiseError(argsError, 42);
            break;

        default:
            break;
        }
    }
    free(args);

    if (argc == 2 && (strcmp(argv[1], "--help") == 0 || strcmp(argv[1],
"-h") == 0))
        exit(0);

    if (opt->input == NULL && optind == argc - 1) {
        opt->input = malloc(strlen(argv[argc - 1]) + 1);
        strncpy(opt->input, argv[argc - 1], strlen(argv[argc - 1]) + 1);
    }

    if (opt->input == NULL)
        raiseError(inputError, 40);

    if (opt->output == NULL) {
        opt->output = malloc(strlen("out.png") + 1);
        opt->output = "out.png";
    }

    if (strcmp(opt->input, opt->output) == 0)
        raiseError(outputIsImputError, 40);
```

```
    return opt;
}
```

Название файла: checkData.c

```
#include "../include/checkData.h"

const char* fileTypeError = "Error: Invalid file type!";
const char* inputError = "Error: input is missing!";
const char* outputIsImputError = "Error: Output matches imput!";

const char* colorError = "Error: Invalid color!";
const char* patternError = "Error: Invalid pattern!";
const char* thicknessError = "Error: Invalid thickness!";
const char* countError = "Error: Invalid pattern count!";

const char* argsError = "Error: Missing required arguments or extra flags
given!";
const char* taskError = "Error: More than one task provided!";

void raiseError(const char* message, int error) {
    printf("%s\n", message);
    exit(error);
}

void checkExtraArgs(OptParams* opt) {
    int task = 0;

    if (opt->copy == true || opt->copy_left != -1 || opt->copy_up != -1
|| opt->copy_right != -1 || opt->copy_down != -1 || opt-
>copy_dest_left != -1 || opt->copy_dest_up != -1)
        task = 1;

    if (opt->color_replace == true || opt->old_color.r != -1 || opt-
>new_color.r != -1) {
        if (task != 0)
            raiseError(argsError, 43);
        task = 2;
    }

    if (opt->ornament == true || opt->ornament_pattern != -1 || opt-
>ornament_count != -1) {
        if (task != 0)
            raiseError(argsError, 43);
        task = 3;
    }

    if (opt->filled_rects == true || opt->rects_border_color.r != -1) {
        if (task != 0)
            raiseError(argsError, 43);
        task = 4;
    }

    if (opt->color.r != -1 || opt->thickness != -1) {
        if (task == 0 || task == 1 || task == 2)
            raiseError(argsError, 43);
    }
```

```
}

void checkCopy(Png* image, OptParams* opt) {
    if (opt->copy_left == -1 || opt->copy_up == -1 || opt->copy_right ==
-1 || opt->copy_down == -1 || opt->copy_dest_left == -1 || opt-
>copy_dest_up == -1)
        raiseError(argsError, 43);

    if (opt->copy_left < 0)
        opt->copy_left = 0;

    if (opt->copy_left > image->width - 1)
        opt->copy_left = image->width - 1;

    if (opt->copy_up < 0)
        opt->copy_up = 0;

    if (opt->copy_up > image->height - 1)
        opt->copy_up = image->height - 1;

    if (opt->copy_right < 0)
        opt->copy_right = 0;

    if (opt->copy_right > image->width - 1)
        opt->copy_right = image->width - 1;

    if (opt->copy_down < 0)
        opt->copy_down = 0;

    if (opt->copy_down > image->height - 1)
        opt->copy_down = image->height - 1;

    if (opt->copy_dest_left < 0)
        opt->copy_dest_left = 0;

    if (opt->copy_dest_left > image->width - 1)
        opt->copy_dest_left = image->width - 1;

    if (opt->copy_dest_up < 0)
        opt->copy_dest_up = 0;

    if (opt->copy_dest_up > image->height - 1)
        opt->copy_dest_up = image->height - 1;

    if (opt->copy_right < opt->copy_left || opt->copy_down < opt->copy_up)
{
        int x = opt->copy_right;
        int y = opt->copy_down;
        opt->copy_right = opt->copy_left;
        opt->copy_down = opt->copy_up;
        opt->copy_left = x;
        opt->copy_up = y;
    }
}

void checkColor(RGB color) {
    if (color.r < 0 || color.g < 0 || color.b < 0)
        raiseError(colorError, 43);
```

```
    if (color.r > 255 || color.g > 255 || color.b > 255)
        raiseError(colorError, 43);
}

void checkReplace(Png* image, OptParams* opt) {
    checkColor(opt->old_color);
    checkColor(opt->new_color);
}

void checkOrnament(Png* image, OptParams* opt) {
    if (!(opt->ornament_pattern == 0 || opt->ornament_pattern == 1 ||
opt->ornament_pattern == 2))
        raiseError(patternError, 43);
    checkColor(opt->color);
    if (opt->thickness <= 0)
        raiseError(thicknessError, 43);
    if (opt->ornament_count <= 0)
        raiseError(countError, 43);
}

void checkRects(Png* image, OptParams* opt) {
    checkColor(opt->color);
    checkColor(opt->rects_border_color);
    if (opt->thickness <= 0)
        raiseError(thicknessError, 43);
}
```

Название файла: operatePng.c

```
#include "../include/operatePng.h"

void read_png_file(char *file_name, Png *image) {
    png_byte header[8];

    FILE *fp = fopen(file_name, "rb");
    if (!fp){
        printf("Cannot read file: %s!\n", file_name);
        exit(40);
    }

    fread(header, 1, 8, fp);
    if (png_sig_cmp(header, 0, 8)){
        printf("Probably, %s is not a png!\n", file_name);
        exit(41);
    }

    /* проверка сигнатуры png*/
    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(40);
    }

    image->info_ptr = png_create_info_struct(image->png_ptr);
    if (!image->info_ptr){
        printf("Error in png info-structure!\n");
```

```
        exit(40);
    }

    if (setjmp(png_jmpbuf(image->png_ptr))){
        printf("Error during init_io!\n");
        exit(40);
    }

    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);

    /* чтение файла */
    if (setjmp(png_jmpbuf(image->png_ptr))){
        printf("Error during read_image!\n");
        exit(40);
    }

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

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

    fclose(fp);
}


void write_png_file(char *file_name, Png *image) {
    FILE *fp = fopen(file_name, "wb");
    if (!fp){
        printf("Cannot read file: %s!\n", file_name);
        exit(40);
    }

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

    if (!image->png_ptr){
        printf("Error in creating png structure!\n");
        exit(40);
    }

    image->info_ptr = png_create_info_struct(image->png_ptr);
    if (!image->info_ptr){
        printf("Error in png info-structure!\n");
```

```
        exit(40);
    }

    if (setjmp(png_jmpbuf(image->png_ptr))){
        printf("Error during init_io!\n");
        exit(40);
    }

    png_init_io(image->png_ptr, fp);


    /* write header */
    if (setjmp(png_jmpbuf(image->png_ptr))){
        printf("Error during header writing!\n");
        exit(40);
    }
    /* Настройка png изображения */
    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);


    /* Запись байтов */
    if (setjmp(png_jmpbuf(image->png_ptr))){
        printf("Error during writing bytes!\n");
        exit(40);
    }

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

    /* Конец записи */
    if (setjmp(png_jmpbuf(image->png_ptr))){
        printf("Error during end of writing!\n");
        exit(40);
    }

    png_write_end(image->png_ptr, NULL);

    /* Очистка памяти */
    for (int y = 0; y < image->height; ++y)
        free(image->row_pointers[y]);
    free(image->row_pointers);

    fclose(fp);
}

void setColor(png_byte* ptr, RGB color) {
    ptr[0] = color.r;
    ptr[1] = color.g;
    ptr[2] = color.b;
}

/*Задание 1 --copy*/
void copyArea(Png* image, OptParams* opt) {
    int color = 3;
```

```
    int currentY = 0;
    int currentX = 0;
    int sizeY = opt->copy_down - opt->copy_up;
    int sizeX = opt->copy_right - opt->copy_left;
    RGB canvas[sizeY][sizeX];

    /*Копирование области*/
    for (int y = opt->copy_up; y < opt->copy_down; ++y) {
        png_byte* row = image->row_pointers[y];

        for (int x = opt->copy_left; x < opt->copy_right; ++x) {
            png_byte* ptr = &(row[x * color]);
            canvas[currentY][currentX].r = ptr[0];
            canvas[currentY][currentX].g = ptr[1];
            canvas[currentY][currentX++].b = ptr[2];
        }
        currentX = 0;
        ++currentY;
    }

    /*Закраска*/
    for (int y = 0; y < sizeY; y++) {
        png_byte* row = image->row_pointers[opt->copy_dest_up + y];
        if (opt->copy_dest_up + y > image->height - 1)
            break;

        for (int x = 0; x < sizeX; x++) {
            if (opt->copy_dest_left + x > image->width - 1)
                break;
            png_byte* ptr = &(row[(opt->copy_dest_left + x) * color]);
            setColor(ptr, canvas[y][x]);
        }

    }
}

/*Задание 2 --color_replace*/
void replaceColor(Png* image, OptParams* opt) {
    int color = 3;
    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 * color]);

            if (opt->old_color.r == ptr[0] && opt->old_color.g == ptr[1]
&& opt->old_color.b == ptr[2])
                setColor(ptr, opt->new_color);
        }
    }
}

/*Задание 3 --ornament*/
void createOrnament(Png* image, OptParams* opt) {
    int color = 3;
    switch (opt->ornament_pattern) {
    case 0: { //rectangle
        int start;
```

```
        int maxY = 0;
        int maxX = 0;

        //обработка линий
        for (int y = 0; y < image->height; ++y) {
            int yValue = y / opt->thickness;
            if (y > image->height / 2)
                yValue = (image->height - 1 - y) / opt->thickness;

            start = opt->thickness * yValue;
            for (int x = start; x < image->width - start; ++x) {
                png_byte* ptr = &(image->row_pointers[y][x * color]);

                if (yValue % 2 == 0 && yValue / (opt->ornament_count * 2)
< 1) {
                    setColor(ptr, opt->color);
                    if (y < image->height / 2)
                        maxY = yValue;
                }
            }
        }

        //обработка рядов
        for (int x = 0; x < image->width; ++x) {
            int xValue = x / opt->thickness;
            if (x > image->width / 2)
                xValue = (image->width -1 - x) / opt->thickness;

            start = opt->thickness * xValue;
            for (int y = start; y < image->height - start; ++y) {
                png_byte* ptr = &(image->row_pointers[y][x * color]);

                if (xValue % 2 == 0 && xValue / (opt->ornament_count * 2)
< 1) {
                    setColor(ptr, opt->color);
                    if (x < image->width / 2)
                        maxX = xValue;
                }
            }
        }

        break;
    }
    case 1: { //circle
        int centerY = image->height / 2;
        int centerX = image->width / 2;
        int radius = (centerX > centerY) ? centerY : centerX;

        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 * color]);
                if (sqrt(pow(x - centerX, 2) + pow(y - centerY, 2)) >
radius)
                    setColor(ptr, opt->color);
            }
        }
```

```
        break;
    }

    case 2: { //semicircles
        double width = (double)(image->width - opt->ornament_count * opt-
>thickness) / (opt->ornament_count * 2);
        double height = (double)(image->height - opt->ornament_count *
opt->thickness) / (opt->ornament_count * 2);
        int radiusX = ceil(width);
        int radiusY = ceil(height);

        int count = 0;
        int middleX[opt->ornament_count * 4];
        int middleY[opt->ornament_count * 4];

        /*Поиск центров полуокружностей*/
        /*Верхние центры*/
        int current =  opt->thickness / 2 + radiusX - 1;
        for (int i = 0; i < opt->ornament_count; ++i) {
            middleX[count] = current;
            middleY[count++] = 0;
            current += radiusX + opt->thickness + radiusX;
        }

        /*Левые центры*/
        current = opt->thickness / 2 + radiusY - 1;
        for (int i = 0; i < opt->ornament_count; ++i) {
            middleX[count] = 0;
            middleY[count++] = current;
            current += radiusY + opt->thickness + radiusY;
        }

        /*Правые центры*/
        current = opt->thickness / 2 + radiusY - 1;
        for (int i = 0; i < opt->ornament_count; ++i) {
            middleX[count] = image->width - 1;
            middleY[count++] = current;
            current += radiusY + opt->thickness + radiusY;
        }

        /*Нижние центры*/
        current = opt->thickness / 2 + radiusX - 1;
        for (int i = 0; i < opt->ornament_count; ++i) {
            middleX[count] = current;
            middleY[count++] = image->height - 1;
            current += radiusX + opt->thickness + radiusX;
        }


        /*Закраска*/
        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 * color]);

                for (int i = 0; i < opt->ornament_count*4; ++i) {
```

```
                    int length = sqrt(pow(x - middleX[i], 2) + pow(y -
middleY[i], 2));

                    if ((middleX[i] == 0 || middleX[i] == image->width -
1) && length >= radiusY && length <= radiusY + opt->thickness)
                        setColor(ptr, opt->color);

                    if ((middleY[i] == 0 || middleY[i] == image->height -
1) && length >= radiusX && length <= radiusX + opt->thickness)
                        setColor(ptr, opt->color);
                }
            }
        }
        break;
    }

    default:
        break;
    }
}

/*Задание 4 --filled_rects*/
void fillRects(Png* image, OptParams* opt) {
    int color = 3;
    int canvas[image->height][image->width];

    /*Замена всех нужных цветов на 1, остальных на 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 * color]);
            if (opt->color.r == ptr[0] && opt->color.g == ptr[1] && opt-
>color.b == ptr[2])
                canvas[y][x] = 1;
            else
                canvas[y][x] = 0;
        }
    }

    /*Увеличивание счётчика поданного цвета, если сверху него такой же
цвет*/
    for (int y = 1; y < image->height; ++y) {
        png_byte* row = image->row_pointers[y];

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


    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 * color]);
```

```
            if (canvas[y][x] > 0) {

                int check = 1;
                int sizeY = 0;
                int currentY = y;
                int sizeX = 0;
                int currentX = x;

                /*Вычисление размеров прямоугольника*/
                while (canvas[currentY][currentX] > 0) {
                    ++sizeX;
                    if (currentX == image->width - 1)
                        break;

                    ++currentX;
                }
                --currentX;

                while (canvas[currentY][currentX] > 0) {
                    ++sizeY;
                    if (currentX == image->height - 1)
                        break;

                    ++currentY;
                }
                --currentY;

                /*Проверка нижних пикселей прямоугольника*/
                /*Ввиду особенности алгоритма проверка других пикселей не
нужно*/
                for (int i = 0; i < sizeX - 1; ++i) {
                    if (currentX - 1 == 0)
                        continue;

                    if (canvas[currentY][currentX] !=
canvas[currentY][currentX-1])
                        check = 0;
                    --currentX;
                }

                currentY = currentY - sizeY + 1;
                --currentX;
                --currentY;

                /*Проверка на обособленность прямоугольника сверху и
снизу*/
                for (int i = 0; i < sizeX + 1; ++i) {
                    if (!(currentY < 0 || currentX < 0 || currentX >
image->width - 1)) {
                        if (canvas[currentY][currentX] > 0)
                            check = 0;
                    }

                    if (!(currentY > image->height - 1 || currentX < 0 ||
currentX > image->width - 1)) {
                        if (canvas[currentY + sizeY + 1][currentX] > 0)
                            check = 0;
                    }
```

```
                    ++currentX;
                }


                currentX = currentX - sizeX - 1;
                ++currentY;

                /*Проверка на обособленность прямоугольника слева и
справа*/
                for (int i = 0; i < sizeY + 1; ++i) {
                    if (!(currentY < 0 || currentX < 0 || currentY >
image->height - 1)) {
                        if (canvas[currentY][currentX] > 0)
                            check = 0;
                    }

                    if (!(currentY > image->height - 1 || currentY < 0 ||
currentX > image->width - 1)) {
                        if (canvas[currentY][currentX + sizeX + 1] > 0)
                            check = 0;
                    }
                    ++currentY;
                }

                if (currentY != 0) {
                    if (canvas[y-1][x] > 0)
                        check = 0;
                }
                --currentY;

                /*Закраска*/
                if (check) {
                    --sizeX;
                    --sizeY;
                    int x0 = x - opt->thickness;
                    int y0 = y - opt->thickness;
                    int x1 = x + sizeX + opt->thickness;
                    int y1 = y + sizeY + opt->thickness;

                    for (int i = y0; i <= y1 && i < image->height; ++i) {
                        png_byte* row = image->row_pointers[i];

                        for (int j = x0; j <= x1 && j < image->width; ++j)
{
                            canvas[i][j] = -1;
                            png_byte* ptr = &(row[j * color]);

                            if (i < 0 || j < 0) {
                                continue;
                            }

                            if ((j < x) || (j > x + sizeX) || (i < y) ||
(i > y + sizeY))
                                setColor(ptr, opt->rects_border_color);
                        }
                    }
                }
            }
```

```
            }
        }
    }
```