**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*

1. 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*

1. 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*

1. 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);

}

}

}

}

}

}

}